Spike2 version 11 for Windows

Spike2 version 11 for Windows

The Spike2 online help as a manual

---

by Cambridge Electronic Design Limited

Spike2 version 11 for Windows

Printed: October 2024 in Cambridge, England

Revision History

11.00 October 2024

Published by:

Cambridge Electronic Design Ltd
Technical Centre
139 Cambridge Road
Milton
Cambridge
CB24 6AZ
UK

Telephone: Cambridge (01223) 420186
International: +44 1223 420186
Email: info@ced.co.uk
Home page: ced.co.uk

Acknowledgements

Some curve fitting functions are based on routines in Numerical Recipes: The Art of Scientific Computing, published by Cambridge University Press and are used by permission.

The XML library used to save and restore resources is based on pugixml (http://pugixml.org); pugixml is Copyright (C) 2006-2023 Arseny Kapoulkine.

Table of Contents

1: Spike2 version 11

Spike2 version 11

With Spike2 version 11 and a modern CED 1401 interface (Power1401, Power1401 mk II, Power1401-3, Micro1401 mk II, Micro1401-3 or Micro1401-4), you can capture and analyse waveform, event and marker data and output precisely timed pulses and voltages using the familiar and easy to use Windows environment. If you have a 1401*plus* or the original micro1401 you can capture date with Spike2 version 7, which is also on the installation medium. You can extend your data capture, or capture data without a 1401 using Talker interfaces.

You can arrange the windows to display the data within them to best advantage and cut and paste the results to other applications. Alternatively, you can obtain printer hard copy directly from the application. When you close a data file, Spike2 saves the screen format and channel display settings. When you open a file, Spike2 restores the configuration, so it is easy to resume work where you stopped in a previous session.

You can analyse sections of data by reading off values at and between cursors, or by applying the built-in functions, for example waveform averaging, digital filtering, spike detection, histogram formation and power spectra. More ambitious users can automate both data capture and analysis with scripts and the output sequencer.

New features in version 11

We have tried very hard to keep version 11 of Spike2 compatible with version 10. It reads data files from all previous versions. Resource files are compatible; some resource formats are extended to support new features. Scripts that ran with version 10 should work unchanged with version 11 unless they used some of the new built-in functions as names. New features in version 11 include:

- The Sampling configuration can now hold a list of pre-set TextMark data items that can be applied during sampling.
- Improvements to interactive use of TextMark data during sampling..
- TextMark data captured from serial line inputs can trigger the output sequencer and arbitrary waveform output.
- Overdraw list dialog supports addition of arbitrary trigger times.
- 3D Overdraw adjustment by dragging.
- Info windows can now display recent TextMark text or Marker channel codes.
- Active Horizontal cursors have two new modes: Median and Median + factor * size.
- Graphical sequencer improvements including a visual indication that a Wait command interrupts the time base and a time out when waiting for a digital input change.

There are many other improvements and more are planned. You can find a full list of new features, bug fixes and changes in the Revision History. Licensed users of Spike2 version 11 can download updates of version 11 from our Web site ced.co.uk as they become available.

Hardware required

The minimum supported system for Spike2 version 11 is a computer running Windows 10 on an x86 or x64 CPU with at least 4 GB of memory. It may run on older operating system, in virtual machines or on different CPU types using emulation, but we do not test or support this. The more powerful the processor and the more memory your system has, the better Spike2 runs. If your CPU is 32-bit only (very old), it must support the SSE2 instruction set. If your CPU is 64-bit, the Spike2 installer will install a 64-bit version of Spike2 unless you choose Custom install and select a 32-bit version.

To sample data, you need a CED Power1401 or a Micro1401 (mk II, -3 or -4) or a suitable Talker. See the *Owners Handbook* that came with your 1401 for hardware installation instructions. Spike2 comes with all required 1401 drivers and will install the Try1401 test program to verify correct 1401 operation.

File icons

The various file types in Spike2 have similar icons so that you can recognise them in directory listings. If your file icons do not display correctly, run Spike2 once as Administrator to allow it to register the file types. To do this, right-click the Spike2 application and select Run as Administrator. You will be asked if it is OK for Spike2 to make changes to the system (which you must allow). This will then trigger a rebuild of all the system icons (which can be slow). After this, the icons should be correct.

This is the Spike2 application icon that you double-click to launch Spike2.

The icon on the left is for Spike2 data files. The icon on the right is for saved result views. The central icon is for an XY file. If you double-click one of these it will launch the Spike2 application (if it is not already running) and open the file.

Spike2 can output sequences of pulses, sine waves and voltage levels as it samples data. Output sequence files have this icon.

This icon is for Spike2 script files. A Spike2 script can automate data capture and analysis operations and extend the capabilities of the Spike2 program.

This is the icon for Spike2 grid files. Grids are useful for storing and presenting tables of information. Columns can be left, right or centre aligned.

The icon on the left is for CED configuration files; these hold all the information needed to sample a new data file. The icon on the right is for CED resource files; these are usually associated with a data file and control how it is displayed.

Direct access to the raw data

Some users may wish to write their own applications that manipulate the Spike2 data files directly. A C library *Spike2: Son data storage library* is available from CED together with documentation sufficient for an experienced C programmer to use it. This library is for the 32-bit library, as used up to version 7 of Spike2. The library documentation is also available as a PDF file on the Spike2 distribution CD. To install it, select Custom Install and check the Additional documentation box.

The library used by Spike2 version 11 is written in C++ and includes the old 32-bit library. The interface has similar functions (in most cases there is a one to one mapping between the old functions and the new), but it is written taking advantage of the features of C++ and can be used to manipulate both the old and new files. There is a MatLab interface to the new format available on our web site. This interface can also be used from other languages that can access external DLLs.

Printed manuals

Since Spike2 version 8, we no longer provide printed manuals. This is to save paper (the manual is more than 1000 pages), reduce shipping costs and to reflect the fact that few users read them as the on-line and context sensitive help is much easier to use. You can get help in Spike2 in most situations where the system is waiting for input from you by pressing the `F1` key or clicking a Help button. There are two PDFs of the on-line help on the distribution disk. These are mechanically generated from the on-line help (that is, it is not hand-optimised for reading as a manual). One version, `spike11.pdf`, is intended for use in a PDF reader and includes highlighted links, the other, `spike11print.pdf` (the version you are reading), is intended for printing and does not include highlighted links (so there may be invitations to find more information "here" that make no sense).

Upgrading from earlier versions

Sampling configurations from Spike2 version 7 or earlier write 32-bit `smr` files for backwards compatibility. Use the Sampling Configuration dialog Resolution tab to change to the new format. Although we allow sampling to old `smr` files, the program is optimised for the 64-bit `smrx` file format. Sampling to the 32-bit format is slower as we convert data from 64-bit to 32-bit. Unless you have pressing reasons for sticking with the 32-bit format we encourage you to migrate to the 64-bit format.

Scripts that explicitly use `smr` file extensions (32-bit files) will need modification to work with `smrx` files. However, if a file with a `smr` extension fails to open as a 32-bit file we try to open it as a 64-bit file, so a short term fix is to change the 64-bit file extensions to `smr`.

Feature	64-bit smrx file	32-bit smr file
Maximum file length in clock ticks	More than 10^{18}	2×10^9
Maximum file length with 1 µs tick	255 thousand years	35 minutes 47 seconds
Maximum file size in bytes	16 EB (16 million TB)	2 GB or 1 TB (big file)
Maximum channels in a file	65534 (2000 in Spike2)	451
Big file data search time proportional to	Log(file size)	file size
Waveform channel gap overhead	16 bytes	0-32746 bytes, average 16374

This table compares some of the features of the two filing systems. The 64-bit system is designed with future developments in mind, the idea being that it is the basis of a format that can remain stable for many years. Particular emphasis has been placed on coping with very large files and long run times.

Old-style configuration and resource files

Spike2 originally used binary configuration and resource files, stored in files with the extensions `s2c` and `s2r`. In version 7.11 (in February 2013) we stopped writing the binary format and changed to using XML format files with extensions `s2cx` and `s2rx`, which gave us more flexibility. Spike2 versions 7 and 8 continued to read the old format, but only wrote the new format. Spike2 versions 9 onwards no longer read the old format; if you must read the old formats, install version 8 and use it to write files with the new format. To convert configuration files, load the old configuration into version 8, sample data with it, then save the configuration. To convert resource files, open the associated data file, then close it.

On-line Help

There is on-line Help available in the program, usually activated by the `F1` key or by clicking a Help button. The help is generally context-sensitive, which makes it easier to use than this manual (even if used in a PDF reader) as it will often open at exactly the information you require. The manual you are reading is mechanically generated from the on-line Help and although there are differences between the on-line help and this manual, the main emphasis has been on making the on-line help as useful as possible. The result is that sometimes, the order of information in this manual will suffer.

CED Software Licences

CED software is protected by both United Kingdom Copyright Law and International Treaty provisions. Unless you have purchased additional licences as described below, you are licensed to run one copy of the software. Each copy of the software is identified by a serial number that is displayed by the Help menu About Spike2… command. You may make archival copies of the software for the sole purpose of back up in case of damage to the original. You may install the software on more than one computer as long as there is **No Possibility** of it being used at one location while it is being used at another. If multiple simultaneous use is possible, you must purchase additional software licences.

Additional software licences

The original licensee of a CED software product can purchase additional licences to run multiple copies of the same software. CED does not supply additional software media. As these additional licences are at a substantially reduced price, there are limitations on their use:

1. The additional licences cannot be separated from the original software and are recorded at CED in the name of the original licensee.
2. All support for the software is expected to be through one nominated person, usually the original licensee.
3. The additional licensed copies are expected to be used on the same site and in the same building/laboratory and by people working within the same group.
4. When upgrades to the software become available that require payment, both the original licence and the additional licences must be upgraded together. If the upgrade price is date dependent, the date used is the date of purchase of the original licence. If some or all of the additional licences are no longer required, you can cancel the unwanted additional licences before the upgrade.
5. If you are the user of an additional licence and circumstances change such that you no longer meet the conditions for use of an additional licence, you may no longer use the software. In this case, with the agreement of the original licensee, it may be possible for you to purchase a full licence at a price that takes into account any monies paid for the additional licence. Contact CED to discuss your circumstances.
6. If you hold the original licence and you move, all licences are presumed to move with you unless you notify us that the software should be registered in the name of someone else.

Installation

Your installation media is serialised to personalise it to you. Please do not allow others to install unlicensed copies of Spike2.

You can run the installation by opening the folder Spike11 on the installation media, then open the `disk1` folder and run `setup.exe`.

During installation

You can have multiple Spike2 versions on the same system as long as they are in different folders. The installation program will propose a standard location for Spike2 which differs for each major version (`Spike9`, `Spike10`, `Spike11`...). You can modify the folder name and, if you wish, you can install in a non-standard location (not recommended). If you want to keep a previous minor revision of Spike2, we suggest you modify the proposed folder name by appending the minor revision, for example `Spike10_01`. You must personalise your copy with your name and organisation.

The installation program copies the Spike2 program plus help, demonstration, example and tutorial files. It also copies and installs all required 1401 support (device drivers and control panels). In rare cases you may need to install the drivers manually; the installation program will tell you if this is the case and give you detailed instructions. Your system may require a restart after installation to get all drivers up to date.

The installation process creates several folders for use by Spike2. Some of these folders are created for the user account under which Spike2 was installed. If you run Spike2 from another account, Spike2 attempts to create the missing folders, but if it fails you may need Administrator assistance.

Due to driver signing requirements in 64-bit versions of Windows 10, if you install Spike2 onto a new Windows 10 system that has never seen a 1401 driver before, the driver must be signed for the new version of Windows. We cannot sign drivers until Microsoft releases the tools to do this, so you may find that a previous Spike2 installation will install the program, but the 1401 driver will not work. You can get the latest 1401 support from our web site.

Custom install

To install without 1401 support, or to copy additional documentation or to exclude example Talker support, choose Custom installation. You can run the installation multiple times to the same folder to apply different Custom options. Installations will not delete files that you have created in data directories.

After Installation

If you are new to Spike2, please work through the Getting Started tutorial. Where you go next depends on your requirements. The *Spike2 Training Course Manual* is more descriptive than this help file, which is organised as reference material. The *Training Course* manual covers all versions of Spike2 and you will occasionally need to refer to this on-line Help for version 11 specific details. If you prefer printed documentation, there is also a PDF version of this manual.

Where is Spike2 installed?

Unless you specifically choose a folder for installation, Spike2 version 11 is installed into:

```
"Program Files"\CED\Spike11
```

`"Program Files"` is typically:

```
C:\Program Files
```

but could be on a different drive. If you install a 32-bit version of Spike2 on a 64-bit system, the folder is:

```
C:\Program Files (x86)
```

This folder path is protected; you need Administration rights to modify files in the Spike2 installation folder.

Updating and removing Spike2

You can update your copy of Spike2 to the latest release from our Web site: ced.co.uk. You can only update a correctly installed and licensed copy of Spike2 within the same major version. There are full instructions for downloading the update on the Web site. If your copy is not the latest major version you can upgrade to the latest major version by purchasing an Upgrade.

Once you have downloaded the Spike2 update, you will find that the update program is very similar to the original installation, except that you must already have a properly installed and licensed copy of Spike2 for Windows on your computer.

Updates will include both bug fixes and new features. If we know your email address we will notify you of new releases. You can also register for this service on our web site. To stop emails, reply to them and ask to be removed from the list.

Support and development policy

We actively develop the latest major release of Spike2, and support (that is make our best efforts to fix bugs) in the latest major release and the previous major release. We may choose to make changes to older versions for reasons of backwards and forwards compatibility. We will always answer questions and give advice on working with older versions; however in some cases the advice might be that upgrading to the latest version is the best way forward.

Removing Spike2

To remove Spike2: open the system Control Panel, select Add/Remove Programs, select CED Spike2 version 11 and click Remove. This removes files installed with Spike2; you will not lose files you created.

Versions of Spike2

This manual describes Spike2 for Windows version 11. We have generated the following versions of "Spike2 for …", listed in more or less chronological order of release:

DOS	Data and output sequencer files are still readable with later versions of Spike2.
Macintosh 68k	Data files, scripts and output sequences are compatible with later versions.
Windows version 2	This version ran on Windows 3.1 and 3.11.
Macintosh PowerPC	This was equivalent to Spike2 for Windows version 2. The last Macintosh version.
Windows version 3	The last version to sample with the standard 1401.
Windows version 4-6	These versions of Spike2 do not support the standard 1401.
Windows version 7	The last version to sample with a 1401*plus* or the original micro1401 (Micro1).
Windows version 8	Introduced the new 64-bit `smrx` filing system. It runs in Windows XP service pack 3 onwards.
Windows version 9	Allowed up 2000 data channels, more arbitrary waveforms, Talker-only sampling. Windows 7 onwards.
Windows version 10	Added Info windows, real-time data processing and derived channel, variable length gating...
Windows version 11	The version described in this manual. It runs in Windows 10 onwards.

You can read about all the version of Spike2 on our web site, here.

We continue development of the latest released version, adding new features and improving the existing code. We actively support (fix bugs) in the latest released version and the one before that. We may choose to fix bugs in earlier versions, and may even add features for forwards compatibility. We will always offer help, workarounds and advice for users of older versions, but that advice may be: 'you should upgrade to the latest version' - especially if the problem is related to changes in the operating system.

Spike2 version references

We mark features that do not exist in earlier versions of Spike2 with [`M.nn`], where `M` is the major version and `nn` the minor version of Spike2 in which it first appeared. You can assume that all higher versions have this feature. If a feature is marked with two or more versions, for example [8.17, 9.03] this means it was added at versions 8.17 and 9.03 and is present in all version 10, 11 and later releases.

In the case of script documentation, if you use a marked feature, your script will not work on earlier versions.

You can find the full list or revisions, fixes and changes in the current major version of Spike2 here. The list of previous major versions of Spike2 is here.

CED 1401 interfaces

The 1401 family of interfaces are intelligent peripherals that generate and receive waveform, digital and timing signals. Using their own processors, clocks and memory, under the control of the host computer, they make complex real world jobs easy to control. In 2022 there are eleven family1401 family members: standard 1401, 1401*plus*, micro1401, Micro1401 mk II, Micro1401-3, Power1401, Power1401 625, Power1401 mk II, Power1401-3, Power1401-3a, Micro1401-4. We place a lot of emphasis on software compatibility; it is easy to write programs that can drive all the family members.

Obsolete and Modern 1401s

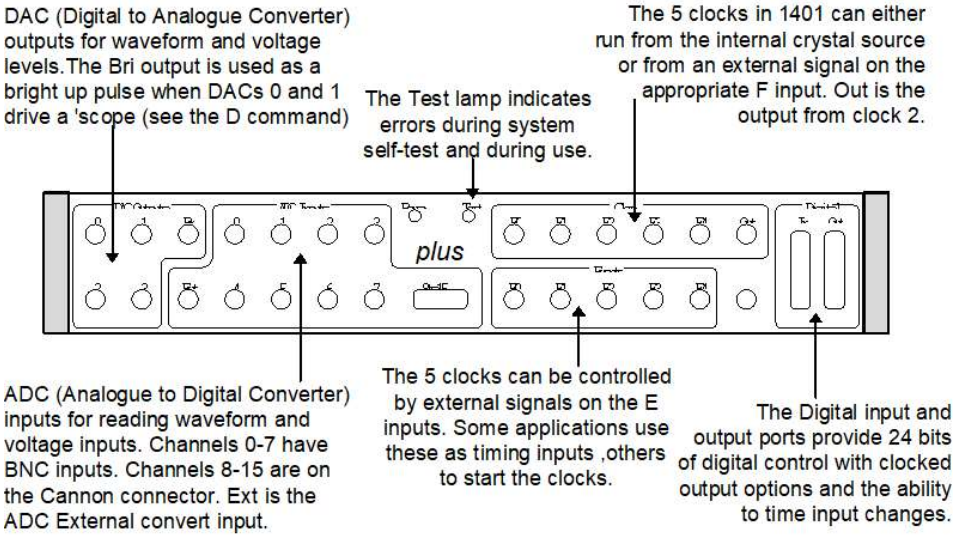
The standard 1401 and 1401plus are obsolete and not supported by Spike2, though a few still survive more than 30 years after issue. They interfaced to the host computer via a proprietary CED interface card that was implemented initially with an ISA interface, and later a PCI interface. The term Modern 1401s refers to all units after these.

The standard 1401 (first available in 1984, now obsolete)

The standard 1401 had a 4 MHz, 8-bit processor, a 12 µs ADC (Analogue to Digital Converter) for sampling 16 channels of waveform data with a separate processor (the Z8 channel sequencer) for automatic channel changing and burst generation, 4 DACs (Digital to Analogue Converters) for waveform output, five clocks, event inputs, digital input and output with clock links and a memory space of around 60 kB for data and commands.

The last Spike2 version to support the standard 1401 was Spike2 version 3.

Front panel of 1401 and 1401plus



In addition, there were several option cards that included:

- MassRAM card for 2 or 8 MB of extra data storage memory and faster sampling
- Expansion of the 16 ADC channels to 32 channels
- Programmable 8 channel event detector
- Programmable gain and filter card options
- Fixed gain and filter cards

The 1401plus (first available in 1991, now obsolete)

The 1401*plus* used a 20 MHz 32-bit processor for 20-40 times more processing power and increased the data space from 59 kB of the standard 1401 to more than 900 kB (16 MB with expanded memory). It was hardware compatible with the standard 1401.

The 1401*plus* supported the same options as the standard 1401 except for the MassRAM, which was emulated by a 1401*plus* with expanded memory. It used the same analogue card as the standard 1401 with the Z8 channel sequencer. However from 1993 it was fitted with a more advanced analogue card 'Issue-M' Channel Sequencer

with a fast 3 µs ADC complete with ADC-silo and high-performance hardware sequencer. There were also analogue card options with 2.5 and 10 µs 16-bit ADC and 4 16-bit DACs for higher accuracy.

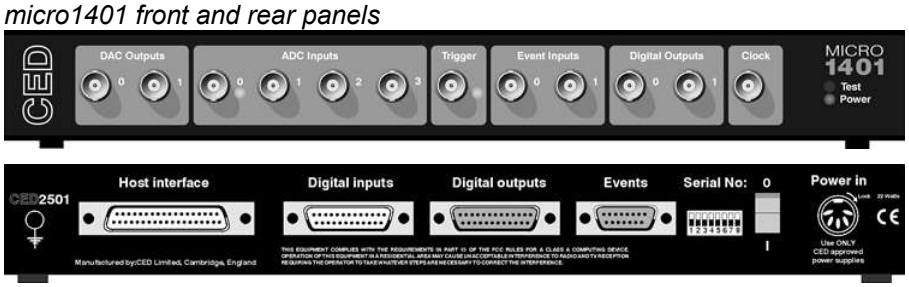
The last Spike2 version to support the 1401*plus* was Spike2 version 7.

The micro1401 (first available in 1996)

The micro1401 (micro1) had the speed and almost all the features of a 1401*plus* with the issue-M analogue card, packed into a much smaller space. Some sacrifices were made in the basic unit; there are only 4 ADC channels and 2 DAC channels as standard. However, there are benefits too: it is small and easily portable, all inputs have LED indicators to show when inputs or outputs are in use, interrupt driven commands generally run faster than 1401*plus*, trigger inputs (as seen by the user) are easier to understand and the unit can be expanded with more channels. It also has the option of a USB interface.

The last Spike2 version to support the micro1401 was Spike2 version 7.

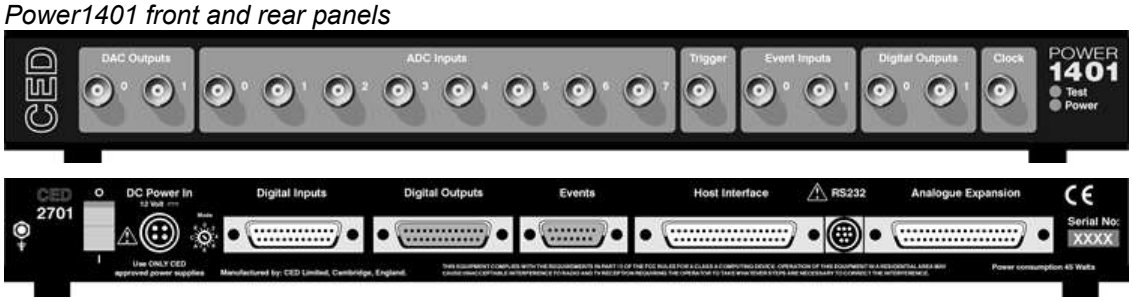
micro1401 front and rear panels



The Power1401 (first available in 2000, Power1401 625 in 2004)

The Power1401 (Power1) took the best features of the 1401*plus* and the micro1401 and added a more powerful processor (up to 30 times faster than the micro1401 or the *plus*), a 16-bit analogue section and up to 256 MB of memory. Like the *plus*, it has 4 DAC channels and 16 ADC channels as standard, like the micro it has a small chassis and LED indicators. It also supports both the standard 1401 interface (ISA or PCI bus) and USB 1. The software and hardware configuration is held in flash memory and can be updated without opening the unit. The Power1401 625 was a revision in 2004 with USB 2 and faster multi-channel sampling.

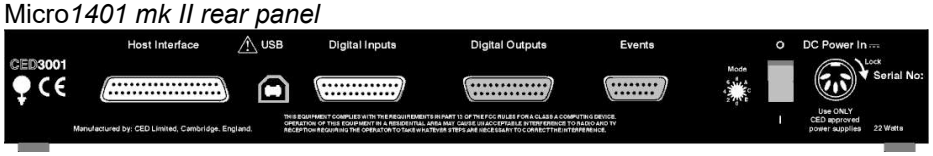
Power1401 front and rear panels



The Micro1401 mk II (first available in 2001)

The Micro1401 mk II (Micro2) looks like the original micro1401 from the front, but it takes much of the internal structure from the Power1401. The processor is more than three times faster than the micro1401, it has a 16-bit 500 kHz ADC, a memory size of 1 or 2 MB and has firmware stored in flash memory for easy update without opening the box. It supports both the standard CED interface and USB.

Micro*1401 mk II rear panel*



The Power1401 mk II (first available in 2007)

The Power1401 mk II (Power2) is similar to the Power1401, but with a processor some 3 times faster, up to 1 GB of memory and a faster multi-channel sample rate. It has a USB 2 interface.

The Micro1401-3 (first available in 2009)

The Micro1401-3 (Micro3) is very similar in appearance to the mk II, but has a faster processor, 4 MB of base memory and 16-bit DACs in place of the 12-bit DACs of the mk II. It has a USB 2 interface.

The Power1401-3 (first available in 2012, -3a in 2016)

The Power1401-3 (Power3) is similar to the Power1401 mk II, but with a faster processor, up to 2 GB of memory and a USB interface that has about twice the throughput. The Power1401-3a was available from 2016 and has the same processor with a faster ADC block and FPGA and provision for more DACs. It uses the same loadable commands as the Power1401-3.

The Micro1401-4 (first available in 2019)

The Micro1401-4 (Micro4) is similar in appearance to the Micro1401-3, but uses a different and much faster processor (typically 3 to 8 times faster) and 32 MB of memory. It has an option to fit 2 extra DACs.

Software compatibility

All members of the 1401 family use the same software interface. It is easy to write applications that will run with any 1401. The language support libraries are written to conceal differences between family members; however, applications that wish to take advantage of 1401-specific features are also supported.

Downloadable commands

The basic capabilities of each 1401 can be extended by loading '1401 commands' into each 1401 to add additional functionality. The commands used by Spike2 are held in the 1401 folder inside the folder holding the Spike2 program. The loadable commands for the 1401 types have different file extensions:

1401 type	Micro1	Power1	Micro2	Power2	Micro3	Power3(a)	Micro4
Extension	`arm`	`arn`	`aro`	`arp`	`arq`	`arr`	`ars`

The commands used by Spike2 version 11 are:

Name	Purpose
`DADC`	Used when setting up signal conditioners to monitor the waveform input.
`MEMDAC`	Replay of waveform data through the 1401 DACs when not sampling data.
`SP11DAC`	Arbitrary waveform output through the 1401 DACs during data capture.
`SP11DIG`	The main command that controls the 1401 during sampling and handles data transfer between the 1401 and the PC.
`SP11FCH`	Setting up WaveMark (spike shape) data capture.
`SP11PUL`	Implements the output sequencer.
`SP11TMP`	Implements real-time WaveMark (spike shape) template matching.

For example, the command used for the output sequencer in the Micro4 is: `PathToSpike2\1401\SP10PUL.ars`

Nomenclature

In this document '1401' refers to all 1401 family types. To be specific we use 'micro1401', 'Micro1401 mk II', 'Micro1401-3', 'Power1401', 'Power1401 625', 'Power1401 mk II' and Power1401-3. As shorthand, we also use 'micro1', 'Micro2', 'Micro3', 'Micro4', 'Power1', 'Power 625', 'Power2', 'Power3' and 'Power3a'. Micro1401 means mk II, -3 and -4, micro1401 (lower case) is the original.

2: Getting started with Spike2

Getting started with Spike2

This tutorial teaches you the basic operations that manipulate Spike2 data files. Spike2 is a large program with many features; this short tutorial will get you started finding your way around.

Demonstration script

There is a demonstration script supplied with Spike2, called `BaseDemo.s2s`, that will give you an introduction to interacting with Spike2 data files using the keyboard and mouse.

Use the Help menu Getting started… command to load and run the `BaseDemo` script. If this command is disabled, it means that Spike2 could not locate the script; re-installing Spike2 should restore it. You can control the script by clicking buttons or using the keyboard. Please remember to click OK or press the keyboard `Enter` key to keep the demonstration moving. If you prefer to just watch the demonstration, start it and click the Options button. Set the Auto OK time to 2 (this is how long to wait, in seconds, before the demonstration moves on).

If you are running the demonstration version of Spike2, you will find that this script has already been loaded for you. You will only need to select it from the list when you use the Script menu Run Script command.

The demonstration script loops back to the start, so you will need to interrupt it to continue with the tutorial. Hit the Exit! button on the toolbar at the top of the screen to stop it running.

There are video tutorials for various aspects of Spike2 on our web site: http://ced.co.uk/tutorials/introduction

If you are not a fan of demonstration scripts or videos, you can get an introduction by reading on...

Opening a file to view

In this step you will open the demonstration data file that was shipped with Spike2. Follow these steps:

1. Open the File menu and select the Open... command
2. Navigate to `My Documents` folder and within that there should be a `Spike11` folder (`Spike2 Demo` in the demonstration version) and within that `Data;` open `Data` and double click on `demo.smr` (if you have moved files since your installation you may have to search around to find this file, if all else fails, use any Spike2 file). Spike2 can read data files with the extensions `.smr` (an old format file with 32-bit times and a maximum size of 1 TB) and `.smrx` (a new format file with 64-bit times and a maximum size limited by the operating system).
3. A new window will open. Arrange the help window and this new window so that you can see both.
4. To follow the tutorial, you should see a window holding at least one data channel and with a horizontal scroll bar at the bottom. If this is not the case your file is in a mess!

Spike2 displays the file in the state in which it was last saved (as long as it can find a file with the same name and the extension `.s2rx`). You are looking at the raw data in the file. We call this a time view because it displays a time history of the data and the axis at the bottom is in seconds.

Clean up a messy file

You can tidy up by following these instructions:

1. Click on the window titled `demo.smr`
2. From the View menu select the Standard Display option
3. If the fonts used seem too large or too small open the View menu Font dialog and select Times New Roman, Regular 10 point
4. If the colours are grotesque open the View menu and select the Change Colours... option and click Reset for a standard colour scheme or choose a better combination. As a last resort, the View menu Use Black And White converts everything into a black foreground and a white background.

Data channels and channel numbers

There are several *data channels* displayed in the window. These channels can hold different types of data: channel 1 holds a waveform, channels 2 and 3 holds events and channel 31 holds keyboard markers.

Selecting channels

You select channels by clicking on the channel number. The channel number is usually to the left of the channel, but can also be on the right if this is selected in the View menu Show/Hide Channel dialog. If the channel number has been hidden you can click the region a few pixels wide in the place where the channel number would be if it were displayed. Spike2 highlights the channel number. Hold down the `Shift` key and click on a channel to select all channels between it and the last selection. Hold down `Ctrl` to select discontinuous channels. Several commands work on a list of selected channels (for example y axis display optimisation).

The mouse pointer changes when you move it over a channel number to remind you that you can select the channel.

Deselect and select all channels

If any channel is selected, you can deselect all channels by clicking in the empty screen area below the channel number, title and y axis. The mouse pointer includes a # when you are over this region to remind you it is active. If you click here with no channel selected, all channels are selected. You can also use the `Ctrl+A` key combination to select all channels, or if all channels are selected to clear all channels.

Channel modified indicator

If the channel number is displayed in red, this means that the channel data has a Channel process or a Marker Filter applied to it or is not displaying the first marker code. If you hover the mouse over the channel y axis area, a tool tip will appear that has more information about the modified status of the channel. From [10.02] this also works for hovering over the channel number area.

Overdrawn (grouped) channel identifier

If you overdraw channels the channel numbers take the main colour used to draw the channel data, which can help you to know which channel is which (but only if you give the channel different colours).

Zoom buttons

The bottom window edge holds four buttons and a scroll bar. The scroll bar controls movement through the file. If you resize the window, the same data is drawn, scaled to the window. The two buttons to the left of the scroll bar change the time range.

This button halves the time range (zoom in). The left edge of the display remains fixed. You can zoom in until the display is one time unit wide.

This button doubles the time range (zoom out). The left edge of the window does not move unless the start plus the new width exceeds the length of the file. If the new width exceeds the file length, the entire file is displayed.

X axis short cut keys

The following short cut keys combinations can be used to navigate the x axis:

Key	Operation
`Left arrow`	Scroll 1 pixel left.
`Right arrow`	Scroll 1 pixel right.
`Shift+Left`	Scroll several pixels left.
`Shift+Right`	Scroll several pixels right.
`Ctrl+Left`	Scroll half a screen left.

Ctrl+Right	Scroll half a screen right.
End	Scroll to the end of the data.
Home	Scroll to the start of the data.
Ctrl+E	Expand (zoom out) the data view around the left edge of the window.
Ctrl+R	Reduce (zoom in) the data view around the left edge of the window.
Ctrl+U	Expand (zoom out/Up) the data view around the centre of the window.
Ctrl+I	Reduce (zoom In) the data view around the centre of the window.

Time views only

Key	Operation
Ctrl+Shift+Left	Search selected event channels for the previous event that is nearest to the centre of the screen and make it the centre or make a sound if there are no selected channels or no more events.
Ctrl+Shift+Right	Search selected channels for the next event that is nearest to the centre of the screen and make it the centre of the screen or make a sound if there are no selected channels or no more events.
Alt+Shift+Left	If Triggered display mode is enabled, steps to the previous event or by a page.
Alt+Shift+Right	If Triggered display mode is enabled, steps to the next event or by a page.

The full list of keyboard commands in a data view can be found in the General information chapter.

Cursor buttons

Vertical

This button adds a vertical cursor to the display (up to 10 vertical cursors can be present in a window). A cursor is a dashed line used to mark positions. You remove cursors with the Cursor menu Delete option. You can add vertical cursors in five ways.

1. Click and release the button to add a cursor in the centre of the window.
2. The Cursor menu New cursor command adds a cursor in the centre of the window.
3. Click and release the right mouse button at the position where you want the new cursor. When the context menu appears, select New Cursor and a cursor will appear at the mouse click position.
4. The short cut keys Ctrl+0, Ctrl+1 through Ctrl+9 place cursors 0 to 9 in the centre of the window.
5. From a script you can use the CursorNew() or CursorSet() commands to create cursors. Script users have more control over cursors and can set custom cursor labels.

You can also use a short cut key to scroll the x axis to locate a cursor. Ctrl+Shift+0 through Ctrl+Shift+9 will scroll so that the selected cursor is as near to the centre of the window as possible. You can read more about vertical cursors in the Cursor menu chapter.

Beware that Microsoft have grabbed Ctrl+Shift+0 for IME language from Vista onwards. You can get instructions to defeat this in the *Technical support*, Frequently asked questions.

Horizontal

There are also horizontal cursors. A horizontal cursor belongs to a channel and has a value that matches the channel y axis. You can use them in dialogs that expect a y axis value and they are often used as part of a script. You can create them interactively from the cursor menu or from the right mouse click context menu or by clicking the horizontal cursor button. Click the button to add a new horizontal cursor to the lowest channel with a y axis on the screen. If channels are grouped, the head channel of the group gets the cursor. You can read more about horizontal cursors in the Cursor menu chapter.

Cursor style and pointers

Click on the cursor button so that at least one vertical cursor is visible. The mouse pointer changes over a cursor or a cursor label:

This indicates that you can drag the cursor. If you drag beyond the window edge, the window scrolls. The further beyond the edge, the faster the scroll. Dragging hides the label unless the `Ctrl` key is down or you drag the label. If a horizontal cursor is outside the range of the y axis of a channel, you can still see the cursor label and even drag it sideways. Hold down `Ctrl` and click and drag to bring it back into the channel area.

If you position the mouse pointer over the cursor label, the pointer changes to a 4-headed arrow to indicate that you can drag both the cursor and the label. This can be useful when preparing an image for publication and you need the cursor label to be clear of data. If you move the pointer to one side, or hold down the shift key, the pointer becomes a two-headed vertical arrow and you can drag the label, but not the cursor. If a cursor is off the edge of the display area and you can still see the label, hold down `Ctrl` and click and drag to move it back into view.

There are four labelling styles for the cursor: no label, position, position and cursor number, and number alone. You select the style with the Cursor menu Label mode option.

Active cursor modes

Type `Ctrl+0` to centre cursor 0 on the screen. Drag any other cursors away from cursor 0. Now right click on cursor 0 and select Cursor 0->Active mode... from the context menu to open the Cursor mode dialog.

All vertical cursors can be *Static* or *Active*. A static cursor stays where you leave it. An active cursor can reposition itself by searching for user-defined data features.

Set cursor 0 to Peak find on channel 1. Set Amplitude to 0.1, Minimum Step to 0 and click the OK button. Now try the `Ctrl+Shift+Right` key combination (`Right` is the right arrow key). Each time you press these keys, cursor 0 seeks the next peak on channel 1 that is at least 0.1 y axis units high. `Ctrl+Shift+Left` moves cursor 0 in the opposite direction. You can set any vertical cursor into active mode. When cursor 0 moves, any other active cursors apply their search method in rising cursor number order.

There are a wide variety of search methods that can be used to locate data features. You can read more about active cursors in the Cursor menu active cursors description, or by pressing the `F1` key when the Cursor mode dialog is open. For now, leave cursor 0 in Peak search mode and move on to the next step.

Automatic measurements to XY view

Now use the Analysis menu Measurements->XY view command to open the Settings for XYPlot dialog.

This dialog lets us take a set of measurements over a time range and generate a graph which you can print, save or copy as either a picture or as a table into other applications. It is likely that the dialog settings will be suitable for this demonstration, but check that the Cursor 0 stepping region holds: Channel=1 Sinewave, Method=Peak find, Amplitude=0.1 and Minimum step = 0. Make sure that the Ignore cursor step if field is blank and that the User check positions box is not checked.

Check in X Measurements that: Type=Time at Point, Time=Cursor(0). Check in Y Measurements that: Type = Value at Point, Channel=1 Sinewave, Time=Cursor(0), Width=0.

Finally check that Points=0 and then click the New button. The Process XYPlot dialog opens so that you can set the region of the data file to analyse. It should hold a sensible start and end time range, so just click the Process button.

Spike2 will generate a new window that displays a graph of the x and y measurements. In this case you will get a graph of peak amplitudes against time in seconds. You can adjust the appearance of the new XY view (remove joining lines, change the markers used for each data point , and the like), but we are here for a quick tour, so close the XY view (click in the X box at the top right).

To find out more about the measurement system you can read more in the Analysis menu Measurements description or press the `F1` key in the Settings for XYPlot dialog.

Zoom in on an area

Move the mouse pointer to the waveform channel. Click the left mouse button and drag a rectangle round a waveform feature and release the button. The window displays the area within the dragged rectangle. If the rectangle covers more than one channel, only the time axis changes. If your rectangle lies within a channel and has zero width, only the y (vertical) axis changes. If you drag past the right or left edge of the window, the view scrolls sideways.

The mouse pointer changes to a magnifying glass when you hold the left mouse button down in the data channel area to show that you are about to magnify the data.

If you hold down the `Ctrl` key and left click, the mouse pointer is the zoom out symbol and the rectangle holding the channel area shrinks to the rectangle you drag

Whichever method you use to scale the data, you can return to the previous display using the Edit menu Undo command or the keyboard short-cut `Ctrl+Z`. If you release the mouse button with the pointer in the same position from which you started the drag, the display does not change.

On screen measurements

If you hold down the `Alt` key before you click and drag, Spike2 displays the size of the dragged rectangle next to the mouse pointer and does not zoom the display. With the mouse button held down, release `Alt`, then use the `C` key to copy the current measurement to the clipboard or the `L` key to copy it to the Log view.

Zoom a channel

You can zoom one channel to use the entire display area!

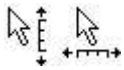
Double click anywhere in the waveform channel. The channel will expand to occupy the entire window.

Double click in the waveform channel again and your previous display is restored.

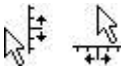
You can also change the display to show a channel and all duplicates of it by holding down the `Ctrl` key, then double-clicking on a channel. This is useful with sorted spikes, where duplicates of a channel are used to display spike classes.

Double clicking a channel works with Time view (and Result view) channels. Before Spike2 version [10.12] it worked only for channels that have a y axis.

Using x and y axes to scroll and zoom



When the cursor is over the tick marks of an axis, you can drag the axis. This maintains the current axis scaling and the window moves to keep pace with the mouse pointer. You can do this with most x and y axes in Spike2. This is particularly useful for y axes as they do not have a vertical scroll bar. The window does not update until you release the mouse button. If you hold down the `Ctrl` key, the window will update continuously.



When the cursor is over the axis numbers, a click and drag changes the axis scaling. The effect depends on the position of zero on the axis. If the zero point is visible, the scaling is done around the zero point; the zero point is fixed and you drag the point you clicked towards or away from zero. If the zero point is not visible, the fixed point is the middle of the axis and you drag the point you clicked towards and away from the middle of the axis

In a time view, result view, or XY view, you can drag the y axis so as to invert the axis. You can prevent this happening by setting an option in the Edit Preferences command Display tab. You are not allowed to invert the X axis. You are not allowed to invert or scroll the y axis in the spike shape dialogs.

Spike2 hides horizontal cursors when you drag the y axis. This makes the drag operation a bit faster, but is mainly done because when filled cursors were set, dragging the y axis (particularly during data sampling) could cause the screen area behind a label to become a mess until the drag ended.

X Range dialog

Click this toolbar button or double-click the time (x) axis of the display to open the X Range dialog. Experiment with the time axis. You can type new positions or use the pop-up menus next to each field to access cursor positions and the maximum time in the file. The most important dialog controls are:

Units — In a time view you can choose between an x axis in seconds, in hours, minutes and seconds and as time of day. Time of day works for files sampled with versions of Spike2 from 4.03 onwards that save the start of sampling time in the file.

Left — Sets the window start time. You can enter times as seconds, for example 3665.2, or in a time format as 61:05.2 (61 minutes, 5.2 seconds) or as 1:01:05.2 (1 hour, 1 minute, 5.2 seconds). The time format extends up to days, so 1:1:1:1 is 1 day, 1 hour, 1 minute and 1 second. At the moment, times are relative to the start of the file, even with time of day mode selected.

In addition to typing times, or selecting a time from the drop-down list, you can type in expressions using the maths symbols + (add), - (subtract), * (multiply) and / (divide). You can also use round brackets. For example, to display from 1 second before cursor 1 to one second past cursor 1 set Left to `Cursor(1)-1` and Right to `Cursor(1)+1`. The Draw button is disabled if you type an invalid expression, or if the Right value is less than or equal to the Left value or if the new range is the same as the current range.

Right — Sets the window end time using the same format as the Left field.

Width — Shows the window width. You can either set the left and right positions, or the left position and the width. Check the box to keep the width the same when the Left field changes.

Show All — Expands the time axis to display all the file and closes the dialog.

Draw — If the axis range had been edited, use this to redraw the data to match the new range.

Close — This closes the dialog; it does not update the axis range.

Cancel — This undoes all changes made with the dialog and closes it.

The Large tick spacing and Tick subdivisions fields let you customise the axis. Values that would produce an illegible axis are ignored. Changes to these fields cause the axis to change immediately; you do not need to click Draw. See the X Axis Range dialog documentation for full details of all dialog items.

Drop-down menus

A drop-down menu item is marked in a dialog by a triangle pointing downwards () that displays a list of possible values for the field when you click the mouse on it. You can select one of the items in the list, or in some cases you can type in your own value.

Y Range dialog

Click this toolbar button or double-click on the y axis of the waveform channel to open the Y Range dialog. This dialog changes the y axis range of one or more channels. The Top and Bottom fields set the values to display at the ends of the axis. Experiment with the y axis range.

Channel — A pop-up menu from which you can select any channel with a y axis, or all channels with y axes, or all selected channels.

Optimise — This button changes the y axis ranges of the set channel(s) to fit the data and closes the dialog. You can optimise the y axis without opening this dialog. The `Ctrl+Q` key combination optimises all selected channels, or all channels if no channel is selected.

Show All — This shows the full range of a waveform channel and closes the dialog. Sampled waveform data is stored on disk as 16-bit integers with a numeric range of -32768 to 32767. The y axis is set to display the full range. For other channels it will display a sensible range.

Draw — If the axis range had been edited or the channel locking fields have changed, use this to redraw the data to match the new settings. It does not close the dialog.

Close — This closes the dialog; it does not update the axis range.

Cancel — This undoes all changes made with the dialog and closes it.

The Lock axes and Group offset fields are hidden unless you have selected a channel that is part of an overdrawn group. These fields are disabled unless the current channel is the first of a group. If you check the Lock axes box, all the grouped channels use the y axis of the first channel in the group to set their display range. The Group offset field sets a vertical display offset, in y axis units, to apply between channels in the group. You would use this if you wanted to draw many channels with the same mean level on the same axis and waned to separate the channels vertically. The offset only applies to the visual display, not to any measurement made on the data.

The Large tick spacing and Tick subdivisions fields customise the axis. Values that would produce an illegible axis are ignored. Changes to these fields are applied immediately; you do not need to click Draw.

If no channel has a y axis, open this dialog from the View menu. See the Y Axis Range dialog for full details.

Channel draw mode dialog

Data files hold two basic channel types: waveform and event. Waveform channels hold a list of values representing the waveform amplitude at successive time intervals. Event channels hold the times at which something happened (and more data, depending on the channel type).

Open the View menu Channel Draw Mode dialog. Experiment with different drawing modes for channel 3. Click Draw to update the display without closing the dialog. Click OK to close the dialog.

Show and Hide channels

Open the View menu Show/Hide Channel dialog. This sets the channels to display in your window. With up to 2000 channels in a file plus memory, duplicate and virtual channels, this ability is quite important if you are to see any detail!

The list on the left of the dialog holds all the channels that can be displayed. Check the box next to a channel to include it in the display list. You can turn all channels on and off with the buttons at the bottom of the dialog. There are also controls to filter the list of displayed channels with regular expressions based on the channel number, title and type.

You can also show and hide the channel number, axes, background grid and the horizontal scroll bar in the window from this dialog and position the y axis on the right and choose to draw axes as scale bars.

Click the Draw button to see the result of your changes without closing the dialog, or click OK to close the dialog and see the changes. If you make a complete mess of your window you can use the View menu Standard Display command to clean things up.

Channel order

Make sure that the demo file is the current window and use the View menu Standard Display command to tidy things up. Click on the Keyboard channel number (31) and drag it down over the other channel numbers.

As the mouse pointer passes over each channel, a horizontal line appears above or below the channel. This horizontal line shows where the selected channel will be dropped. Drag until you have a horizontal line below channel 1 and release the mouse button. Channel 31 will now move to the bottom of the channel list. Type `Ctrl+Z` or use the Edit menu Undo to remove your change.

You can move more than one channel at a time. Spike2 moves all the channels that are selected when you start the drag operation. For example, hold down `Ctrl` and click on the channel 3 number. Keep `Ctrl` down and click and drag the channel 2 number. When you release, both channels will move. The mouse pointer shows a tick when you are in a position where dropping will work.

The usual Spike2 channel order is with low numbers at the bottom of the screen. If you prefer low numbers at the top of the screen, open the Edit menu Preferences and check Standard Display shows lowest numbered channel at the top, then use the View menu Standard Display command.

Channel spacing

Change the channel 3 drawing mode to Mean frequency. Hold down the `Shift` key and move the mouse over the data area. Hold the `Shift` key down and click. Drag up and down and release the mouse.

When you click with `Shift` down, the mouse jumps to the nearest channel boundary and you can change the boundary position by dragging. With `Shift` down, you can move the edge up and down as far as the next channel edge. You can undo changes or use Standard Display to restore normal sizes.

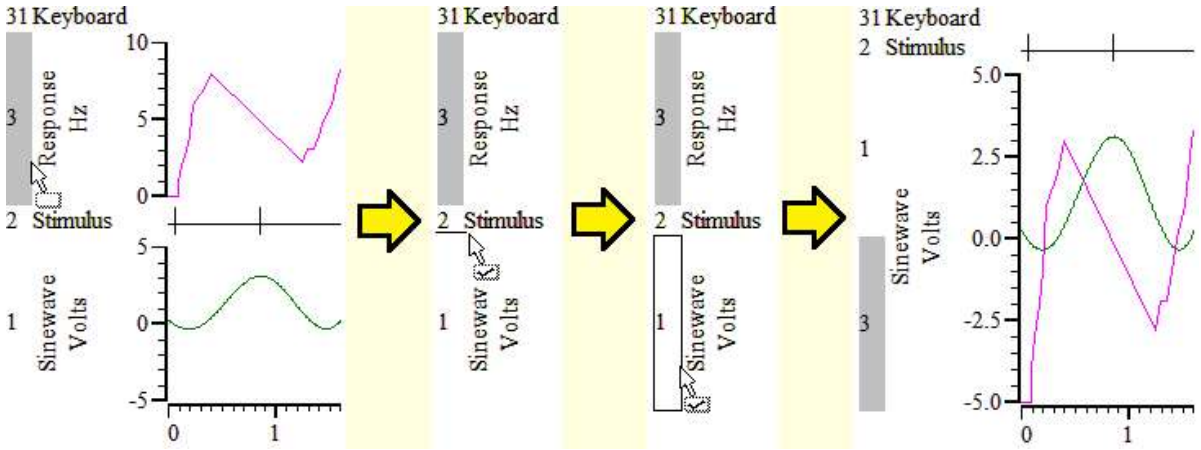
If you add `Ctrl`, all channels with a Y axis are scaled. If there are no channels with a Y axis above or below the drag point, then all channels scale. You can force all channels to scale by lifting your finger off the `Shift` key (leaving `Ctrl` down) after you start to drag the boundary.

Channel grouping (overlay)

You can group (overlay) data channels with a Y axis in a time view or a result view. This is done interactively by dragging channel numbers on top of each other or from a script by using the `ChanOrder()` command. When you drag channels, you can drop them on top of another channel with a Y axis or on top of a channel group. As you drag, a hollow rectangle appears around suitable dropping zones. You can also drop between channels when a horizontal line appears.

To do this in the `demo.smr` example file:

Click the "3" of channel 3 and drag it on top of the "1" of Channel 1 and release.



Steps in overdrawing channels

Channels 1 and 3 now share the same space with the channel numbers stacked up next to the Y axis. The visible Y axis is for the top channel number in the stack. In this case, this is channel 1, which is the *head* of the group. Only a channel with a y axis can be the head of a group. The other channels in the group are referred to as *members* of the group. You can promote another channel in the stack to the head by double-clicking the channel number (as long as the channel has a y axis). The channels retain their own y axes and scaling. You can remove a channel by dragging the channel number to a new position.

Grouped channels are drawn such that the head channel with the visible Y axis is drawn last. If you have a channel that fills in areas, such as a sonogram or an event channel drawn as rate mode or in state mode, put it at the bottom of the stack, as it will mask channels below it in the stack. If overdrawn channels have been given their own channel primary colours, the channel numbers for the overdrawn channels are displayed in the channel primary colour.

Before Spike2 version [10.07] only channels with a y axis could be grouped. From [10.07] you are allowed to group any channels, but only channels with a y axis can be the head of a group. This change allows the background of a channel to be coloured by a Marker channel drawn in State mode.

Common (locked) y axis

If you want all grouped channels to draw at the same scale as the displayed y axis, double-click the Y axis scale to open the Y axis dialog and check the Lock axes box. If you have very similar channels, they will tend to lie on top of each other. You can use the Group offset field to set the vertical displacement between adjacent channels in y axis units. The head of the group has no offset, the first member has the offset, the second has twice the offset, and so on. If you have included channels without a y axis in a group, you will likely want to place them last so the channels with y axes are equally spaced.

Effect of grouping on cursors

Grouping channels has no effect on vertical cursors.

Horizontal cursors are hidden if the y axis they belong to is hidden. This means that if you drag a channel with horizontal cursors into a group, the horizontal cursors will vanish. You can make them appear if you lock the Y axis axes (as then the y axis they belong to is logically visible).

Effect of grouping on context menu

When channels are grouped, the context menu that appears when you right-click in the channel data area is for the topmost channel in the group. From version [10.10], you can see context menus for 'underneath' channels by right-clicking on the channel number.

Tidy up

Use the View menu Standard Display command to tidy things up before you continue.

Cursor values

Make sure there are cursors in the window, then click this toolbar button or use the Cursor menu Display Y Values option to open the Cursor Values dialog. The columns show the cursor times and values. For channels without a y axis, the value is the next event time after the cursor. If you move cursor or alter a channel display mode, the values change. If you add or remove channels in the display, they are added or removed in the dialog.

The Time zero and Y zero check-boxes select relative rather than absolute measurements; the radio buttons set the reference cursor. Reference cursor values are unchanged; values at other cursors have the reference value subtracted.

You can copy dialog fields to the clipboard. Click and drag for multiple selections. Click a top or left hand cell to select an entire column or row. Hold down `Ctrl` to make non-contiguous row or column selections.

Cursor regions

Click this toolbar button or use the Cursor menu Cursor Regions command. Experiment with changing cursor positions and channel display types. The regions dialog looks at the data values between cursors. There are several modes set by the drop down list at the bottom of the dialog:

Area	The area between a waveform and the y axis 0.0 level or the number of events in the region.
Mean	The average level of a waveform, or the number of events divided by the region width.
Slope	The gradient of the least-squares best fit line to the waveform, no meaning for events.
Sum	The sum of waveform values between the cursors or the number of events in the region.
Area(scaled)	This mode is the same as Area, except that when a Zero region is set, the value in the zero column is scaled to allow for the relative column widths before being subtracted.
Curve area	Each data point makes a contribution to the area of its amplitude above a line joining the end points multiplied by the x axis distance between the data points.
Modulus	Each data point makes a contribution to the area of its absolute amplitude value multiplied by the x axis difference between data points. This is equivalent to rectifying the data, then measuring the area. If a zero region is specified, the amount subtracted from the other regions is scaled by the relative width of the regions.
Maximum	The maximum value found between the cursors.

Minimum	The minimum value found between the cursors.
Peak to Peak	The difference between maximum and minimum values found between the cursors.
SD	The standard deviation from the mean of the values between the cursors. If there are no values between the cursors the field is blank.
RMS	The value shown is the RMS level of the values found between the cursors. If there are no values between the cursors the field is blank.
Extreme	The value shown is the maximum absolute value found between the cursors. Thus if the maximum value was +1, and the minimum value was -1.5, then this mode would display 1.5.
Peak	The value shown is the maximum value found between the cursors measured relative to a baseline formed by joining the two points where the cursors cross the data.
Trough	The value shown is the minimum value found between the cursors measured relative to a baseline formed by joining the two points where the cursors cross the data.

You can also make relative measurements by checking the Zero region box and choosing a reference region.

Result views

Windows holding raw, unprocessed data are called *Time views*. There is another type of data window, called a *Result view*, that holds the result of analysing time view data. These windows hold one or more channels of data, each channel has the same number of data points and x axis are suitable for waveforms or histograms. Channels can also have event times associated with them for raster displays. There are two steps in the analysis:

1. You set the type of analysis, the channels to analyse, the number of points or bins to generate and any other parameters required. This creates a new, empty result window.
2. You set a time region of the time window and Spike2 calculates the result and adds it to the result view.

You may repeat step 2 as many times as is required to accumulate results from different sections of data. The new window behaves like a time view containing a single channel of data. Result views can also be created from a script.

Make an interval histogram

Close all windows except the original time view of `demo.smr`. Then:

1.  Click this button on the toolbar or use the Analysis menu New Result View command to select Interval Histogram.
2. Set the channel to analyse. The channel list in the pop-up menu includes event channels only. Channel 3 of the `demo` file is the best for this example.
3. Set the number of bins in the histogram, we suggest 150.
4. Set the width of each bin, in seconds. Bins must be at least one Spike2 clock tick wide. The bin width is rounded to a multiple of this clock tick. We suggest 0.01 seconds (this is 10 milliseconds).
5. Leave the last field set to 0.0 seconds. This is the minimum interval that appears in the new window.
6. Click the New button to create a new window and open the Process dialog.

Process dialog

Now set the region of the data file to analyse:

1. Set the start time for analysis to 0
2. Set the end time to View(-1).MaxTime() (this is the time of the last data item in the file), this is in the pop-up menu.
3. Check the Optimise Y axis after process box to scale the y axis to the data automatically.
4. Click Process to analyse the data and display the result.

The Clear result view before process check box sets the result window contents to zero before you analyse the data, otherwise each new result is added to the previous one. The Settings button takes you back to the previous step.

Repeating and extending a process

You can add more data into the result or change the process settings and process again:

1. Recall the process dialog by selecting the Process command from the Analysis menu.
2. Click the Process button again. The data in the result window will double in size (as long as you have not checked the Clear result view before process box).
3. Recall the process settings dialog by selecting the Process settings command from the Analysis menu.
4. Change the number of bins to 400, and the time resolution to 0.004 then click the Change button and confirm that you wish to continue.
5. Select a region for processing with the new settings.

Result view drawing modes

Experiment with the Channel Draw Mode command in the View menu.

There are four different drawing styles available for result windows: Histogram, Line, Dots and SkyLine. These styles are self-explanatory. The various analysis routines that create result windows will select an appropriate style.

See the View menu Result view drawing modes documentation for more detailed information.

Result view cursors

Experiment with cursors in this new window.

You will find that the cursors behave in a very similar manner to the original time window and that you can use the Cursor menu Cursor Values and Cursor Regions commands in the same way.

XY Views

XY views have a wide variety of uses, from displaying user-defined graphs to drawing pictures. XY views have the following features:

- One x axis and one y axis shared between all data channels in the XY view, so all the channels share the same space; you can overdraw one channel with another.
- Up to 2000 data channels allowed in the view (the limit was 256 before Spike2 version 9).
- Channels are numbered consecutively from 1. If you delete a channel, the channel numbers of any higher-numbered channels change. For example, if you have three channels (1, 2, 3) and you delete channel 2, the old channel 3 becomes channel 2.
- Each data channel is a list of (x,y) data points. The number of data points in a channel is limited only by available memory and drawing time. However, you can limit the number of data points on a channel, in which case new data points replace the oldest data points.
- The data points can be drawn with markers at each data point. The range of marker styles currently includes: dots, boxes, plus signs, crosses, circles (Windows NT only), triangles, diamonds, horizontal lines and vertical lines. The size of the markers can also be set, and they can be made invisible.
- The data points can be joined with solid, dotted or dashed lines, and the line thickness can be varied. You can also choose to join the last point in a channel to the first point to make a loop.
- You can sort the order of the data points in a channel by x, by y or by order of insertion in the channel. This is only important if the data points are joined.

- The colour of the lines and markers can be chosen. If no colour is set, the same colour as for a waveform channel in a time view is set.
- You can process data from a time view to generate an XY view with multiple channels; see Measurements to XY views.

As we do not display separate y axes for each channel, there is no selectable channel number field; you cannot select a channel in an XY view.

Make XY view

The simplest way to create an XY view is to take some measurements from a time view:

1. Close any extra views so that only the `demo` time view remains.
2. Select the Analysis menu Measurements->XY View... option or click on the Measurements icon in the toolbar and select XY View... from the pop-up menu.
3. A complicated dialog will appear (see here for details or click the Help button). The default state of this dialog will find all the peaks on channel 1 of the demo file and generate an XY view with the peak positions as the x value and the peak amplitude as the y value. Accept this by clicking the New button at the bottom right.
4. The familiar process dialog will open. It will be set to process all the time range in the demo file, so click the Process button.
5. You will get a graph that is fairly convincing, but may have a few negative values that do not 'feel' like peaks. We can fix these...
6. Click the Settings... button on the Process dialog to go back to the previous dialog.
7. The problem is that we have accepted the default value in the Cursor 0 stepping dialog region Amplitude field (0). This accepts the smallest upwards wobble of the data as a peak. For now, just edit the `Amplitude` field to `0.1` and click the Change button.
8. The output data is removed; this is done because the analysis has changed. The Process dialog opens again and you should click Process to generate the data again.
9. This time, the dubious points will not be present.

You can modify the drawing style of the XY view. The easiest way to do this is to right-click in the view and experiment with the context menu that appears.

The Measurements system uses the Active Cursor system to iterate through a time view. Each successful active cursor iteration results in a new (x,y) data value. The x and y values that are selected in the X measurements and Y measurements regions of the XY Measurements Settings dialog.

Instead of measuring to an XY view you can measure to a RealMark channel in a Time view. This can be more convenient when you need to see the results in the context of the time view data.

Other sources of information

There are video tutorials for various aspects of Spike2 on our web site: http://ced.co.uk/tutorials/introduction that demonstrate specific features in detail.

If you have worked through the Getting started tutorial, you have the basic skills required to make use of Spike2 for interactive data analysis. You can find more detailed information in the following sections:

Sampling data	This describes the different data types Spike2 supports and how to configure Spike2 to sample them. It also covers arbitrary waveform output (output through the 1401 DACs of previously sampled waveforms or waveforms generated by the script language).
Data output during sampling	This describes the output sequencer, which allows you to produce precisely timed digital pulses, voltage ramps and steps and cosine waves. You can also detect changes in external signals in real time and respond to them in much less than a millisecond.

File menu	This describes all the commands in the File menu: Opening and saving files and configurations, exporting data from Spike2 and Printing.
Edit menu	This describes the commands in the Edit menu including the Preferences (well worth a visit).
View menu	This describes the View menu, which covers all aspects of what you see in each data window.
Analysis menu	This describes how you can generate result views from time view data, how to take measurements to a new XY view, how you generate Memory channel (for example by picking peaks from waveforms), the marker filter, spike shape creation and editing and the digital filter.
Window menu	Describes the commands in the window menu.
Cursor menu	Covers the use of vertical and horizontal cursors, how to set active cursors and how to use the Cursor regions and Cursor values dialogs.
Sample menu	Describes the commands in the sampling menu including off-line waveform output and the use of TextMark data during sampling.
Script menu	This is a brief overview of the commands in the script menu. There is a separate manual *The Spike2 script language* which has a detailed description of the language. You should also see the *Spike2 Training Course Manual* for more details of selected script topics.
Spike shapes	This chapter has a more detailed explanation of the use of Spike2 to sample spike shapes and the use of the template system to identify spike shapes.
Digital filtering	This describes the digital filter dialog in detail, and also has a more technical section which discusses the use of digital filters.
XY views	This has a brief introduction to XY views from the point of view of a script writer and introduces a script that can generate "waterfall" displays.
Signal Conditioners	Spike2 can control programmable signal conditioners such as the CED 1902, the Axon CyberAmp and the Power1401 with the ADC gain option fitted.
Utilities	There are additional programs provided with Spike2 to fix damaged data files and to check out your 1401.

To learn about using the script language for analysis you should read about the script language and investigate the example scripts provided with Spike2:

Script language Introduction	Basic introduction to the script language including a "Hello world" script and how to record simple actions and then edit them into a more useful script.
Script window and debugging	Working with the script window and the associated script commands and how to use the built-in script debugging system.
Script language syntax	An explanation of the script language structure, keywords, variable types, data arrays and basic programming structures together with simple programming examples.
Functions by topic	Spike2 has more than 500 built-in script functions. This section groups them together by function. For example, if you want to find all the script functions that relate to sampling, then start here.
Alphabetic function list	This is the full list of all built-in functions arranged alphabetically. Each function comes with a description and some also include examples of use.

The *Spike2 Training Course Manual* is another resource you can use. It contains getting started chapters on many topics, and is an invaluable reference for the script writer as it has worked examples that cover many common requirements.

3: General information

General information

This covers topics that are important throughout Spike2 and miscellaneous items that don't fit anywhere else.

View types and files

We often refer to *Views* when we describe features of Spike2. The types of views that an interactive user of Spike2 has to deal with have a window on the screen with an associated menu and a data file on disk. These views come under the categories of Data based views and Text based views. If you write scripts you will also have to deal with two other types of views: External views (with no associated window) and Other). There is a more technical description of views and view handles in the Script language section.

Data based view

There are three types of data views in Spike2:

Time views (.smrx, *.smr)*

These display data that is being sampled or has been sampled by Spike2 or imported into Spike2. They are created by the File New command (to create a new file for sampling based on the current sampling configuration), or by opening an existing file or by importing a foreign format file. You can also create a time view from the script language and use the script language to fill the channels.

Result views (.srf)*

Result views are generated based on Time views and hold channels of similar data that can be displayed as waveforms or histograms. You can open saved result views with the File menu Open command and create new ones when the current view is a Time or Result view from the Analysis menu New Result View command. You can also create them from the script language.

XY views (.sxy)*

XY views are generated from analysis of Time views or by the script language. They display multiple channels of data points defined by (x,y) co-ordinates. The data can be drawn as dots, lines, closed curves or as histograms, allowing a wide variety of displays to be created. Existing XY files are opened by the File menu Open command.

Text based views

Text-based views correspond to standard text files. From Spike2 version 8 onwards, we expect text files saved on disk to be in UTF-8 format, but we will attempt to read text in an range of formats and translate as appropriate. We save text files in UTF-8. If you are firmly of the ASCII text persuasion, do not worry as this is a subset of UTF-8. We distinguish the varieties of text file we support by the file extension used to store them on disk. The Text file view types that Spike2 supports are:

The Log view

The log view is unusual in that it does not have an associated data file, though you can save it as a text file. The Log view is used as a convenient place for Spike2 to display messages and there is a dedicated script command, `PrintLog()`, to make it easy to write to from the script language, regardless of the current view.

Text views (.txt)*

These views are created by opening an existing `.txt` file or by creating a new text file with the File New command or from the script language with `FileOpen(name$, 1,...)` or `FileNew(1,...)` commands. They display the text in the file and the text can be added to it with the script `Print()` command. You can choose to display a gutter margin which can hold bookmarks, and a line number margin, which is normally hidden. Text views are normally used from the script language to build output files that the user can view. Script users can also create external text files, which are faster but have no display window.

Script views (.s2s)*

These views are created by opening an existing script file or by creating a new one with the File New command or from the script language with the `FileOpen(name$, 3,...)` or `FileNew(3,...)` commands. Scripts are used to automate Spike2 operations. Most Spike2 interactive functionality can be controlled by a script, and there is a lot of functionality that is only available from a script.

Script views display the text in a window. Script views highlight your script files based on the syntax of the script language, which can make scripts easier to understand and write. They are also used to debug scripts. They have additional folding margin which shows the structure of the script and allows you to fold away blocks of code based on the script structure.

Output sequencer views (*.pls)

These views are created by opening an existing output sequence file or by creating a new one with the File New command. Output sequences are used when sampling data with a 1401 interface to sequence DAC and digital outputs. They can also be used to respond to inputs very quickly, enabling real-time responses to external signals. Like scripts, output sequences support syntax highlighting and folding.

There is an alternative way to create output sequences using a graphical editor. The output of the graphical editor is converted to a text sequence before it is used.

Grid views (*.s2gx)

These were added at version 8.05. They are mainly intended for use from the script language, but can be used interactively. They provide a spreadsheet-style grid that can be used to store data and to build tables for printing. They are not intended for vast quantities of data. You can read more about them here.

External views

These views have data files and view handles, but no display windows. They are used for text and binary data files and are only available to users of the script language.

External text files (.txt, *.*)*

These files are opened and created with the `FileOpen(name$, 8,...)` script command. You can read data from the files with `Read()` and write data with `Print()`; the files are accessed sequentially, that is you will read from the start of the file through to the end or write from the start to the end or append to the end. Because there is no associated screen image to update when you write or to scroll through on a read, external text files can be much faster to use than a text view with an associated window, particularly if the file is very big.

External binary files (.*)*

These files are opened and created with the `FileOpen(name$, 9,...)` script command. You can read and write real, integer and text data to these files in a wide variety of formats. The files allow random access. See the `BRead()`, `BReadSize()`, `BWrite()`, `BWriteSize()` and `BSeek()` commands for details. You can read or write just about any type of file as a binary file, but to do it usefully you need to know the exact file format. The example for the `BWriteSize()` command shows you how to create a `.bmp` file using the script language. The example for `Spline2D()` has an example of generating a bitmap file.

Other view types

These view types are for objects that are controllable by the script using view handles, but that do not fit the view types listed above. These are things like Info windows, control bars, the Sampling status view, the Spike2 application window, the spike sorting window, the multimedia windows and so on.

Data file topics

Spike2 clock tick
Data file versions
Data channel types
Channel search expressions
Channel lists
Dialog expressions
Data view keyboard shortcuts
Overdrawing data in Spike2
Interrupting drawing
Recycle Bin

Spike2 clock tick

All items in a Spike2 `.smrx` or `.smr` data file exist at an integer multiple of a basic time unit, the underlying *clock tick* or *time resolution* for the file. This unit is typically very small, and usually lies between 1 and 100 microseconds. A 64-bit smrx data file can run for 8 x 10^{18} of these clock ticks, which at 1 microsecond per tick would be 255 thousand years. Put another way, if you use a 64-bit `smrx` data file, the length of the file is not a consideration and you should probably run at 1 microsecond per tick unless there is a pressing reason to run slower.

However, if you use a 32-bit smr data file, it can only run for a little more than 2 billion (2 x 10^9) of these clock ticks. At a clock tick of 1 microsecond, a file is limited to around 36 minutes. A 10 microsecond clock tick file can run for almost 6 hours. At 100 microseconds the file could run for more than 2 days.

The value of the clock tick is usually some multiple of microseconds or tenths of microseconds when the original source of the data is a CED 1401 data acquisition unit. However, when data is imported or a data file is created by a script, the clock tick could be any value.

The clock tick used when sampling is set in the Resolution tab of the Sampling configuration dialog. From the script language, the clock tick is set by the `FileNew()` command when creating a file. The `FileTimeBase()` script command can be used to read the clock tick and it can also be changed to adjust the time base of a file (for instance when a EEG data is played back from a recorder at several times the normal rate).

Data file versions

Spike2 data is stored in files with the extensions .smr (for the original 32-bit times format) and .smrx (for the 64-bit times format).

The 32-bit file format was designed around 1987 and has been much extended since. It was limited to 2 GB in size until version 9, which allowed sizes up to 1 TB. There were 9 major version changes to the 32-bit format; we are on the second version of the 64-bit format. We have always made sure that more modern versions of Spike2 will read older file versions. Occasionally we are asked why the 32-bit file extension was `.smr` and not, for example, `.spk`. It stands for `S`on of `MR`ate - `MRate` was a CED MS-DOS program that was the predecessor of Spike2 in that it sampled both waveforms and event times.

The 64-bit format was released with Spike2 version 8 in 2013. We strongly recommend that you use the new, 64-bit `.smrx` version of the library for all new work.

Unless you are interested in history, or you have an old version of Spike2 and cannot read a data file, you can stop reading this page now.

If you are interested in the technical details of the library, or need to interface to it, there is documentation and header files on the CED web site (follow links to **Downloads** and then **Son library**). You can read the file version with the `FileInfo()` script command and from the View menu File Information... command.

Version Change history

1 The original SON filing system, written in Pascal, supported waveform and `Event` data only.

2 New `Marker` data type. Added extra space to the file header for future expansion. The library was much faster reading large data files as it remembered the last accessed data.

3 Added the `FilterMarker` function. Changed the meaning of the waveform channel divide to prevent an apparent change of sampling rate if a waveform channel was added or deleted. This effectively removed the use of the time per ADC value stored in the file.

4 Added the WaveMark data type. Changed the use of `FilterMarker`. The library now caches the current data block to avoid re-reading when the required block is already in memory.

5 Added the `TextMark` and `RealMark` data types and the `MaxTime` and `ChanMaxTime` functions. C library versions written for the IBM PC in DOS and Windows and for the Macintosh. The version 5 libraries write version 3 and 4 files if the data does not need new features.

In 1998, we extended the C version to support read-only files and to allow more data to be written per call. We added lookup tables to speed up data access in long files and write buffering to remove the need for `FastWrite`, allow peri-triggered sampling, and speed up writing).

6 Documented for C/C++ use only with information on use as a DLL by other languages. Added the `RealWave` channel type. WaveMark channels have multiple traces. Waveform rate divider is now 32-bits. Added support for time and date stamp, basic time unit and an application identifier. Files still compatible with versions 3, 4 or 5 if they do not use version 6 features.

7 You can choose to round the sizes of WaveMark and TextMark extended marker types up to a multiple of 4 bytes so that we can build the library on systems that insist on aligned data access.

8 Altered storage of channel numbers to avoid the bit used to hold the initial state for a level channel, allowing the maximum number of channels to be increased to 451. Extended the lookup table so that the table for a channel starts small and grows up to a limit (larger than the old fixed size).

9 Added support for big files (up to 1 TB), rewrote the internal lookup table system to improve speed and saves the table as part of the file in Big file mode. Macintosh (big-endian format) support was removed. Linux support is added. Pointers to disk space that were previously byte offsets (but multiples of 512) are now block pointers (a pointer value 10 means offset 5120). This increases the maximum file size by a factor of 512. There are changes to the file header to support a lookup table on disk and to channel headers to track the block counts. There were also type changes to allow compiling as 64-bit code on Windows and Linux.

256 0x100 **64-bit times**. A complete redesign of the filing system to remove the time and disk space limitations of the original while leaving a system that is similar enough to the original to store the same data and behave in similar ways. Times are stored as 64-bits and the files can theoretically be of any size up to 2 to the power 64 bytes (around 10 to the power 19 bytes).

Practical considerations, such as the need to copy files in reasonable time will limit file sizes to a small number of TB for the next few years. Times are stored to 64-bit accuracy, but some operations within Spike2 use floating point numbers to manipulate file times, and a floating point number has some 53 bits of precision, so this also limits the available maximum time. However, with 1 microsecond ticks, you can still sample for tens of years before this becomes an issue.

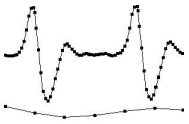
You can access both 32-bit and 64-bit files using the same API, so although there is software effort required to use the new format, once done you can read and write both old and new formats with the same code. Also, if the son64.dll file is placed in the same folder as a recent son32.dll file, old code that reads old 32-bit files can also read a new file (as far as the first 32-bits of clock ticks).

257 0x101 This fixed a bug where a data file, usually created by importing a file with a lot of channels and with quite long channel comments, titles and units, could end up being unreadable. You can recover a file in this state with S64Fix, but you may lose some of the text strings. There is no change to the file format; the version number change is to help us recover the file.

Data channel types

In a Spike2 time view there are two basic data types: Waveforms and Events. These are then further sub-divided into more types. You can always find out the type of a channel by hovering the mouse over the channel y axis area and a tool tip will pop up displaying (amongst other things) the channel type.

Waveform and RealWave



Waveforms are stored as a list of data values that represent a signal amplitude. The time gaps between the samples are all the same, and this is known as the *sampling interval*. The reciprocal of the sampling interval is known as the *sampling rate*. So a signal with a sampling interval of 0.01 seconds (or 10 milliseconds) has a sampling rate of 100 Hz. The sampling interval must be an integer multiple of the clock tick for the file. For example, if the underlying clock tick was 1 microseconds waveform sample intervals of 1, 2, 3, 4...n microseconds are possible. Put another way, sampling rates of 1000 kHz, 500 kHz, 333.3 kHz, 250 kHz... 1000/n kHz are possible.

Spike2 stores waveforms on disk as either 16-bit integers in the range -32768 to 32767 (a Waveform channel) or 32-bit floating point values (a RealWave channel). Both these channel types have an associated *scale* and *offset* value plus a text string that defines the user units to use for the y axis when displaying the data. For a Waveform (integer) channel, the scale and offset are used to convert the stored integer values into user units.

This is arranged so that a scale value of 1.0 and an offset value of 0.0 map the input of a 1401 device with a ±5 Volt range into Volts. That is:

user units = integer value * *scale* / 6553.6 + *offset*

RealWave channels are stored in user units. They use the scale and offset values whenever a RealWave channel needs to be converted to an integer value for use as a Waveform. In this case:

integer value = (user units - *offset*) * 6553.6 / *scale*

If the result of this would exceed 16-bit integer range (-32768 to 32767), it is limited to -32768 or 32767.

Waveform and RealWave channels can have gaps in them where there is no data. The very first data point of the channel determines the possible positions of all subsequent points as all points lie at a time that is:

time of first point + n * sampling interval

where n is an integer value. You can read about sampling waveform data in the *Sampling data* chapter.

Effect of Scale factor on Text output to Clipboard, File and Spreadsheet

When outputting Waveform data, the number of decimal places is set so that the equivalent of half a bit change is visible. This means that for a scale factor of 1.0, we display 5 decimal places. For every change of a factor of 10 in the scale factor, the number of decimal places changes by 1:

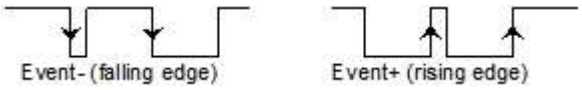
Channel scale factor	100.0	10.0	1.0	0.1	0.01
Decimal places	3	4	5	6	7

From version [10.20], when formatting RealWave data for text output, the number of decimal places used is also set based on the scale factor. Prior to this version, 6 decimal places was always used.

Event

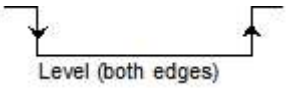
The basic concept of an event is that it is a time stamp. The time stamp is a 64-bit number that is the count of the number of underlying clock ticks from the start of the file to the event. The time of the event in seconds is:

time in seconds = time stamp * underlying clock tick period



Spike2 has two event types that are exactly equivalent: Event+ and Event-. The + and - relate to how the data was captured. The plus means that the time was from a signal rising through a trigger level, the minus means that the signal was falling through a level. This is only important when data is captured. You can read about sampling event data in the *Sampling data* chapter.

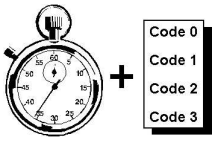
Level



There is a third simple event type, Level. In the 32-bit filing system, this was stored in exactly the same way as Event data and each data block had a flag that indicated the direction of the first edge in the block. This had the advantage of being compact, but the disadvantage that deleting an edge caused all following edges (but only in the block) to invert and caused all kinds of logical problems. The new son64 system implements Level channels as Marker data using marker code 00 for a low-going edge and code 01 for a high-going edge.

When stored in a memory channel, we still treat Level data as a list of times to make it easy to insert and delete edges and to maintain backwards compatibility with Spike2 version 7. However, when we write Level data to a son64 file we save it in Marker format.

Marker

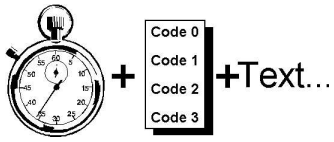


Spike2 extends the event concept by attaching data to each event. The simplest form of attached data is the Marker, which adds 4 8-bit marker codes (in the range 0-255 decimal, 00-FF hexadecimal) to each event time. These codes can then be used to filter the event data. For example, the keyboard marker channel that Spike2 always adds to a sampled data file stores the keys pressed during data capture in the first marker code. If you associate a particular key with a recurrent event during sampling, you can filter the channel to show only events with that particular key code. Marker data stored in 64-bit `.smrx` files also has a 32-bit integer value stored with each marker. Currently, all sampled data sets this extra data to 0, but file importers can set it and it is accessible with the `LastTime()`, `NextTime()`, `ChanData()` and `MarkEdit()` script commands.

As a Marker is exactly like an event, anything that you can do with an event channel can be done with an (optionally filtered) Marker channel. You can read about sampling Marker data in the *Sampling data* chapter. You can also create such channels as memory channels and then save them to make a permanent channel.

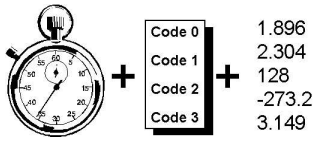
TextMark

A TextMark channel is exactly the same as a Marker (so anything that can be done with a Marker can be done with a TextMark), except that it has a text string attached. There is a maximum size of string set for each TextMark channel. You can set a TextMark channel for use during sampling. You can also create such channels as memory channels and then save them to make a permanent channel.



RealMark

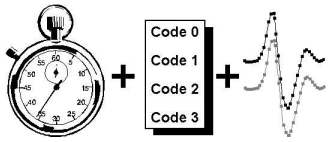
A RealMark channel is exactly the same as a Marker, but with the additional ability to store a list of real numbers. RealMark channels can be created as the result of an active cursor measurement processor they can be created as a memory channel and saved to a permanent channel. An example of the use such a channel would be store a list of blood gas parameters that are sampled from time to time during an experiment.



Since version [9.02] you can attach separate title and units to each item and from [10.18] you can duplicate a RealMark channels as a waveform for each item.

WaveMark

A WaveMark channel is exactly the same as a Marker, but also stores 1, 2 or 4 short waveforms (stored as 16-bit integers). These are normally used to store Spike shapes. The marker codes are usually used to store codes that represent the result of sorting the spike shapes into different classes. You can read more about sampling spike shapes in the *Spike sorting* chapter .



Channel search expressions

In most places where you can type a *channel list*, you can also type a *channel search expression*. This allows you to select channels with Titles, Units or Comments that match a regular expression.

A channel search expression is a string of the form `"what=regexpr"`, where `what` specifies what we search and `regexpr` is an ECMAScript regular expression that specifies the text that must be matched (found within the expanded `what` string) to include this channel. The text of `what` is scanned and each `T`, `U` and `C` (case is important) in `what` is replaced by the Title, Units and Comment of a channel. Then the resulting string is searched by the regular expression. If `what` is empty, it is treated as `T` (title search).

This is a very powerful feature; however, it requires you to understand how regular expression syntax is used.

Examples

`"T_U=y_V"` would match channels with a title that ended with `y` and units that start with `V` (assuming that you did not use the underline characters in the title or units). The text to be searched for each channel is the channel Title followed by underscore followed by the channel Units. The idea of the underscore is to use an unusual character to mark the end of the title and the start of the units as we want to match the end of the title. If your title might hold an underscore, you would need to use a different character as a separator. For example, if the Title was `"Funny"` and the Units string was `"Volts"`, this would expand to `"Funny_Volts"`, which is matched as it contains `"y_V"`.

`"=^G\\d+ x"` would match channel titles like `"G2 x"` or `"G29 x"`. In this case we have not supplied a `what`, so the channel title is searched. We need two backslash characters as backslash is an escape character. The regular expression that is used is `"^G\d+ x"`. This means: find a `G` at the start of the text followed by 1 or more decimal digits followed by a space and an `x`.

Channel lists

In many places where you are prompted for a channel, you can select a channel from a drop down list, or you can type a channel list. A channel list is a list of channel numbers or channel ranges separated by commas. A channel range is two channel numbers separated by two periods or a hyphen, for example `4..7`, which is equivalent to channels 4, 5, 6 and 7. The channel range `7-4` is equivalent to channels 7, 6, 5 and 4. The following channel list:

```
1,3..5,7
```

means channels 1, 3, 4, 5 and 7. In most cases, Spike2 checks channel lists and removes channels that are not suitable for the operation. For example, if you open the `demo.smr` file supplied with Spike2, select an Interval histogram and type a channel list of `1..32` and then click on another field, Spike2 reformats the list to 2,3,31 as these are the only suitable channels. It is not an error for a channel list to include unsuitable channels, however it is an error for a channel list to include no suitable channels.

You can replace channel numbers with symbolic names using `m` for a memory channel or `v` for a virtual channel. For example `m1` for the first memory channel or `v2a` for the first duplicate of virtual channel 2. If you specify a range of duplicates of the same channel, the range expands by duplicates, not by channel numbers. For example: `2..2c` means 2, `2a, 2b, 2c`. We allow up to 52 duplicates of a channel using upper case `A..Z` for the second set of 26. So `2..2C` is the same as `2..2z, 2A..2C`.

Channel lists can also be used in script commands, for example: `ChanShow("1..4")`. Script commands that will accept this format describe the argument as `cSpc`. You can find more information about channel specifiers in the script language documentation.

Anywhere that you can you a channel list you can also use a channel search expression to match channel titles, units and comments.

Dialog expressions

Many dialogs in Spike2 accept an expression in place of a number. These expressions can be divided into two types: numeric expressions and view-based expressions.

Numeric expressions

A numeric expression is composed of numeric values, the arithmetic operators `+`, `-`, `*` and `/`, the logical operators `<`, `<=`, `=`, `>=`, `<>` and `?`, arithmetic functions and round brackets `(` and `)`. The result of a logical comparison is 1 if the result is true and 0 if the result is false. You may not have come across `?` (the ternary operator), which is used as:

```
expr1 ? expr2 : expr3
```

The symbols `expr1`, `expr2` and `expr3` stand for expressions. The result is `expr2` if `expr1` evaluates to a non-zero value and `expr3` if `expr1` evaluates to zero.

If you write expressions involving more than one operator, for example `1+2*3` you need to know if this is evaluated as `(1+2)*3` or as `1+(2*3)`. This is determined by the operator precedence, described below (where you will find the answer).

Arithmetic functions

Dialog fields that allow expressions will also accept the following functions. In these examples, `x` stands for a numeric value or valid dialog expression. [9.02]

`Abs(x)`	If `x` is negative, return `-x`, otherwise `x`.
`Cos(x)`	Return the cosine of `x` (`x` is in radians).
`Sin(x)`	Returns the sine of `x` (`x` is in radians).
`Min(x1,x2{,x3...})`	Returns the most negative of the arguments.
`Max(x1,x2{,x3...})`	Returns the most positive of the arguments.
`Sqr(x)`	Returns `x * x`.

`Sqrt(x)`	Returns the square root of `x`, or 0 if `x` is negative.

View-based time/x-axis expressions

These expressions follows the rules for numeric expressions and allow references to positions along the x axis. If a dialog field is documented as allowing expressions, and the field supplies an x axis position (for example a time), then you can use the following:

`BinSize()`	In a time view, this returns the time resolution of the file, in seconds. In a result view it returns the bin width, in seconds. In an XY view it returns 1. New at version [10.04].
`BinSize(n)`	In a time view, if `n` is a waveform channel it returns the sample interval of the channel as displayed, in seconds. In all other cases it returns the same value as `Binsize()`. New at version [10.04].
`Cursor(n)`	Where `n` is 0 to 9 returns the position of the cursor. If the cursor does not exist or the position is invalid, the expression evaluation fails.
`C0` to `C9`	This is shorthand for `Cursor(0)` to `Cursor(9)`.
`CursorX(n)`	Where `n` is 0 to 9 returns the position of the cursor before the move to the current position. If the cursor does not exist the expression evaluation fails. [8.09]
`CX0` to `CX9`	This is shorthand for `CursorX(0)` to `CursorX(9)`, and returns the previous position of cursors 0 through 9. [8.09]
`XLow()`	The left hand end of the visible x axis in x axis units (seconds in a time view, else axis units).
`XHigh()`	The right hand end of the visible x axis in x axis units.
`MaxTime()`	The right hand end of the x axis in x axis units. It is not valid in an XY view. In a time view that holds WaveMark data, this will be slightly more than the value of `MaxTime()` returned by the script language as it includes the width of the widest WaveMark data item (so it can be displayed).
`MaxTime(n)`	The time of the last data item on channel `n` in a time view.

Beware: Most of these have an identically named equivalent in the script language that operates in the same way in Time and XY views. However, in a Result view, the script language cursor-related routines return bin numbers whereas the dialog expressions *always* return x axis units.

You can combine these expressions with any of the operators or arithmetic functions (it is up to you to ensure that what you use makes sense). For example:

```
Cursor(2)>Cursor(1) ? Cursor(2) : Cursor(1)
```

evaluates the position of the rightmost of cursors 1 and 2.

View-based channel/y-axis expressions

If a dialog field is documented as allowing expressions and the field supplies a y axis position associated with a specific channel, then you can use the following command.

`HCursor(n)`	Where n is 1 to 9 returns the position of the horizontal cursor. If the cursor does not exist or the position is invalid, the expression evaluation fails.
`H1` to `H9`	This is shorthand for `HCursor(1)` to `HCursor(9)`.
`HCursorX(n)`	Where n is 1 to 9 returns the position of the cursor before the move to the current position. If the cursor does not exist the expression evaluation fails.[8.09]
`HX1` to `HX9`	This is shorthand for `HCursorX(1)` to `HCursorX(9)`, and returns the previous position of cursors 1 through 9. [8.09]
`YLow()`	The bottom end of the visible y axis.
`YHigh()`	The top end of the visible y axis.
`At(t{,c})`	The value on the relevant channel at time t, which can be a view-based time expression.
`Mean(t1,t2{,c})`	The mean level on the relevant channel between times `t1` and `t2`, both or either of which can be view-based time expressions.
`SD(t1,t2{,c})`	The standard deviation of the values on the relevant channel between times `t1` and `t2`, both or either of which can be view-based time expressions.

`Meas(m,t1,t2{,c})` Any available measurement (selected by `m`) on the relevant channel between times `t1` and `t2`, both or either of which can be view-based time expressions. See the `ChanMeasure()` script function for a list of the possible values of `m` and the corresponding measurements.

Several of these commands with `{,c}` allow you to specify a channel number to override the channel to be used. The curly braces mark optional arguments; you don't type them! For example, to use the `At()` command at the time of cursor 1 on channel 3 you would type: `At(C1,3)` but to use the default channel you would type `At(C1)`.

You can add `View(-1).` before these expressions to force the expression to be evaluated for the time view linked to the current view, for example `View(-1).Cursor(0)`. This is required when the current view is a result view and you wish to access timing information from the time view that the result view is based on.

Times as numbers

All times are in units of seconds. However, where a time is typed into a dialog you can use `{{{days:}hours:}minutes:}seconds` where the seconds may include a decimal point and items enclosed in curly brackets are optional. Each colon promotes the number to the left of the colon from seconds to minutes to hours to days. Times may only contain numbers and colons, white space is not allowed. One decimal point is allowed at the end of the time to introduce fractional seconds. We also allow a number with no colons to be followed by `ms` or `us` (with no spaces) to interpret the time in milliseconds or microseconds. So the following are equivalent: `1.6`, `00:00:01.6`, `1600ms`, `1600000us`.

Time of Day format

From [10.11] we also allow you follow a number (usually in the `days:hours:minutes:seconds` format) with `tod` (no spaces between the number and `tod`). This allows you to type in a time that matches the value displayed on a time view x axis in Time of Day mode. Examples: `1:13:27.5tod`, `12:00:00tod`. The Cursor Position dialog (activated from the vertical cursor context menu) uses this format when the x axis is in Time of Day mode. Adding this suffix offsets the number to remove the time of day of the start of the file from the typed in value so the result is an offset in seconds from the start of the file.

Operator precedence

In the table, `LHS` means the value of the expression to the left of the operator as far as the next operator of same or lower precedence, `RHS` means the value of the expression on the right up to the next operator of the same or lower precedence. Where operators have the same precedence, evaluation is from left to right. The precedence order from highest (Level 1) to lowest (Level 5) is:

Level		Name	Return value
1	`()`	Brackets	Everything inside a pair of brackets is evaluated before considering the effect of an adjacent operator.
2	`*` `/`	Multiply Divide	`LHS` multiplied by `RHS` `LHS` divided by `RHS`. It is an error for `RHS` to be zero
3	`+` `-`	Add Subtract	`LHS` plus `RHS` `LHS` minus `RHS`

4	`<`
	`<=`
	`=`
	`>=`
	`>`
		Less than
		Less or equal
		Equal
		Greater or equal
		Greater than
			If `LHS` less than `RHS` then 1 else 0
			If `LHS` less than or equal to `RHS` then 1 else 0
			If `LHS` equal to `RHS` then 1 else 0
			If `LHS` greater than or equal to `RHS` then 1 else 0
			If `LHS` greater than `RHS` then 1 else 0

5	`?`	Ternary operator	`LHS ? A : B` has the value `A` if `LHS` is not 0, and `B` is it is 0. Put spaces around the colon to distinguish it from a time.

`1+2*3-4` has the value 3 because multiply has a higher precedence than add. Put another way, the operator with the highest precedence (lowest level) is used first, so `2*3` is evaluated, leaving `1+6-4`. The remaining operators both have the same level, so are evaluated left to right, first the `1+6` leaving `7-4`, and thus the result is `3`. Use brackets to force other orderings: `(1+2)*(3-4)`, for example.

Script language compatibility

The expressions are compatible with the script language except for the use of `C0` to `C9` and `H1` to `H9` (and the `CnX` and `HnX`) as shorthand for `Cursor(0)` to `Cursor(9)` and `HCursor(1)` to `HCursor(9)` and the use of colons, `ms` and `us` to denote times. If you use these in a script you will get syntax errors. However, you can use these constructs in strings passed as expressions to `CursorActive()` or `MeasureChan()`.

Data view keyboard shortcuts

The following short cut key combinations can be used in a time, result view or XY view except for `Ctrl+Shift+Left/Right` and `Alt+Left/Right`, which are only available in a time view. Items marked *Always available* can be used, regardless of the current view.

Key	Operation
`Left arrow`	Scroll 1 pixel left.
`Right arrow`	Scroll 1 pixel right.
`Shift+Left`	Scroll several pixels left.
`Shift+Right`	Scroll several pixels right.
`Ctrl+Left`	Scroll half a screen left.
`Ctrl+Right`	Scroll half a screen right.
`Ctrl+Shift+ Left/Right`	Time views only. If cursor 0 is active, search for the next/previous feature and scroll the screen to make it visible.
`Alt+Left Alt+Right`	Time views only. Search selected event channels for the previous/next event that is nearest to the screen centre and make it the screen centre or make a sound if there are no selected channels or no more events.
`Alt+Shift+ Left/Right`	Time views only. Jump to the next display trigger point (only if the display trigger is enabled). This is equivalent to the manual stepping buttons at the bottom left of the view.
`Home/End`	Scroll to the start/end of the data.
`Ctrl+n`	Where `n` is 0 to 9. Fetch vertical cursor 0 to 9. If the cursor does not exist it is created. Cursor 0 exists only in time views.
`Ctrl+Shift+n`	Where `n` is 0 to 9. Centre the window on the cursor, if it exists. Beware that Microsoft have grabbed `Ctrl+Shift+0` for IME language from Vista onwards. You can get instructions to defeat this in the *Technical support*, Frequently asked questions.
`Ctrl+Shift+A`	Fetch all vertical cursors.
`Alt+n`	Where `n` is 1 to 9. Fetch horizontal cursor `n` to the first visible channel with a Y axis.
`Ctrl+A`	Select all channels. If all channels are selected, deselect all channels. Equivalent to a click in the bottom left-hand corner of a Time or Result view.
`Ctrl+Alt+A`	Abort sampling; unless no data has been captured or you have been running for only a very short time you will be asked to confirm this.
`Ctrl+C`	Copy the image of the view to the clipboard.
`Ctrl+E`	Expand (zoom out) the data view around the left edge of the window.
`Ctrl+G`	Open the Graphical Sequence editor during sampling. Currently you cannot use this option until sampling has started and the initial graphical sequence is running.
`Ctrl+I`	Reduce (zoom In) the data view around the centre of the window.
`Ctrl+Shift+I`	Open the View Information dialog for the current view (where supported).
`Ctrl+K`	Show the x axis dialog.
`Ctrl+N`	Open a new data file choice dialog. Always available.
`Ctrl+O`	Open the file open dialog. Always available.
`Ctrl+P`	Print the current view. Always available.
`Ctrl+Q`	Optimise selected channels in the current view. If none selected, optimise all channels.
`Ctrl+L`	Open the Evaluate window to run script commands. Available unless a script is running.
`Ctrl+R`	Reduce (zoom in) the current view around the left edge of the window.
`Ctrl+Shift+R`	Reset sampling. You will be queried unless no data has been captured or you have been running for a very short time. Always available when sampling.

Ctrl+S	Save the current view (where supported). Equivalent to the File menu Save command.
Ctrl+Alt+S	Start sampling (if a data view is ready to sample) or stop sampling if already sampling data.
Ctrl+T	Create a TextMark during sampling (if the TextMark channel exists), open the TextMark review dialog offline.
Ctrl+U	Expand (zoom out/Up) the current view around the centre of the window.
Ctrl+Shift+W	Toggle writing data to disk on all channels. Always available when sampling.
Ctrl+Y	Show the y axis dialog.
Ctrl+Z	Undo the last undoable operation for the current view.
Ctrl+Break	Break out of long drawing or calculation operations.

Overdrawing data in Spike2

There are several ways to overdraw data in Spike2:

- Overdraw Time or Result view data channels so that they share the same vertical space and optionally share the same y axis. Use this to compare two or more channels, or to group similar channels. You can apply a per-channel vertical shift to visually separate channels within a group.
- Overdraw all the WaveMark data (spikes) in a Time view time range so that they all start at the same horizontal position on the screen and all share the same y axis. Use this so you can see different spike classes and locate outliers. There are special mouse clicks you can use in this mode to locate individual spikes or to give a selection of spikes a defined code.
- Overdraw data copied to an XY view for various graphical effects. Use this to generate user-defined images and waterfall plots.
- Overdraw multiple triggered frames of data in a time view based on a list of trigger times with optional 3D display. Use this to see how the response to a stimulus changes with time.

Interrupting drawing

Despite our best efforts to draw huge data files quickly, drawing can take a long time. You can interrupt it with the Ctrl+Break key combination. The Break key is usually at the top right of the keyboard and is often labelled Pause with Break on the front. If your system has sound enabled and you have selected a sound for "Exclamation", Spike2 plays this sound to confirm that drawing has been abandoned for the current channel. Screen areas that have not been filled with data are filled with a uniform background colour.

Spike2 can detect that a drawing operation is likely to be slow. If it is, the mouse pointer changes to the hourglass cursor to indicate that you have to wait. If a drawing operation takes more than 1 second, the window title changes to remind you that you can use Ctrl+Break to interrupt the drawing.

Spike2 may interrupt drawing during data sampling if the host computer is not keeping up with the data capture. This is unusual, and normally only happens when you run at high sample rates with a slow interface (such as USB 1). It can also happen with a severely fragmented disk system that has become very slow, or if you have very little free memory and a lot of paging to disk is taking place.

If a drawing operation to a time view takes more than 2 seconds, Spike2 will try to break it up into multiple drawing operations as long as the channels that require painting are not in Overdraw WM mode, are not overdrawn and are not in triggered overdraw mode. If this is not possible, after 5 seconds of uninterrupted drawing, the operating system may mark Spike2 as "Not Responding". This can lead to problems, as once the drawing operation ends, the system may decide that the invalid area that was being redraw still needs to be redrawn, which restarts the process.

Recycle Bin

If you do not save a created Time view, the default Spike2 behaviour when Spike2 closes the file is to move it to the Recycle bin. We do this out of an excess of caution; some files may be very valuable and very difficult to replace. However, this can result in the Recycle bin being filled with a large number of files, especially if you often abandon sampling sessions and restart them.

If you accumulate a very large number of files in the Recycle Bin, this can be time-consuming to empty. We suggest that you check your Recycle Bin periodically and Empty it from time to time.

From version [10.20] there is a new Edit menu Preferences option in the Sampling tab that sets the minimum duration of a created file that will be Recycled if not saved.

If you control sampling from a script, the `FileClose()` command has an option to not move a closed and unsaved file to the recycle bin.

Text view topics

Text view features
Text view keyboard shortcuts

Text view features

The Log view, script views, the output sequencer and text windows created by a script all count as Text views and they have common features. This section deals with editing features:

Drag and drop
Virtual space
Multiple selections
Select rectangular text area
Read only text

Drag and drop

The editor supports drag and drop of text both within Spike2 and between Spike2 and other applications that support it (for example the Spike2 Help system). Spike2 also supports drag and drop for rectangular text areas.

Operation	Method
Move block	Select the text to move. Move the mouse pointer over the selected text and hold down the left mouse button and drag. The mouse pointer will indicate that you can now drag the text and the text caret will show the insertion point. Drag the text to the desired insertion point and release.
Copy block	Select the text to copy. Hold down the `Ctrl` key and move the mouse pointer over the selected text, click and drag. A small + symbol indicates the copy operation and the text caret will show the insertion point for the duplicate. Drag the text to the target position and release the mouse button to duplicate the text. The `Ctrl` key must be down when you release the mouse button or the operation will move the text.

Virtual space

If virtual space is enabled you can position the caret beyond the end of the text in a line by clicking in a blank area with the mouse, or using the cursor right key. You cannot position the text caret below the last line of text. When the caret is beyond the end of the line, it is said to be in *virtual* space. If you type with the caret in virtual space, space characters will be added to fill in the virtual space up to the text you type.

There are two main uses for virtual space: to add comments without having to space along to the required column, to make rectangular selections without having strange visual effects due to short lines.

If you use the script language to manipulate the text caret and make selections, virtual space is ignored; a caret in virtual space will be treated as if it is at the end of the line. There are no script commands that will move the caret into virtual space. If there is a requirement for the script to report or use virtual space, we will extend the script in a compatible way to incorporate it.

Multiple selections

You can make multiple selections in a text view. To do this, hold down the `Ctrl` key and click and drag. Each time you make a new selection in this way it becomes the current selection; all previous selections are shown with a different selection colour. When you have a multiple selection, each selected area has a flashing text caret at the insertion point. If you type characters, these will appear at all insertion points, replacing all the selected text. If you use the delete or backspace key, all selected text will vanish. Note that pasting into a multiple selection will clear all the selected text, then insert the pasted text at the current (last made) selection.

Multiple selection can be useful when you want to move several non-consecutive script functions to make them consecutive. Select all the sections you want to move, use `Ctrl+X` to cut them all, release `Ctrl`, click at the insertion point, then use `Ctrl+V` to paste the result.

Select rectangular text area

You can select, cut, paste and drag rectangular selections within Spike2. To select a rectangular area hold down the `Alt` key then select text with the mouse. The point where you hold down the mouse button will be one corner of the selection, the point where you release the mouse will be the other corner. You can use this feature to change the alignment of comments in a script, or to convert a single column of numbers into multiple columns. You can also paste such text into other applications as plain text.

A rectangular selection will paste within Spike2 as a rectangular selection. Beware that `Ctrl+X` (cut) on a rectangular selection followed by `Ctrl+V` (paste) will not leave the text unchanged (unless you select the text from bottom to top). The cut operation will leave a vertical flashing line (assuming a fixed pitch font) with the insertion point marked by a more visually obvious caret. The paste operation will paste the cut text as a rectangular block at the insertion point. A rectangular selection is a multiple selection.

Read only text

If you open an output sequence or script file that is held in a read only file, the text window will also be marked read only and you will not be allowed to change it. This is to allow users to protect sequences and script from inadvertent change. If you want to edit such a text file you have two options:

1) Close the file, remove the read only status on disk, then open it
2) Save the file to a new file with a different name

You can use the `Modified()` script command to get and change the read only status of a text file. However, if the original disk file is marked read only, using `Modified()` to allow you to edit the text will not change the status of the disk file, so you will not be able to save it to the same file name.

Text view keyboard shortcuts

Text views have more keyboard short cuts than any other area of Spike2. We have grouped them by function to make the huge list more digestible.

Find Replace and Bookmarks

The Find and Replace commands can be accessed from the Edit menu, from the Edit Toolbar and by keyboard short cuts:

Key	Operation
`Ctrl+F`	Open the Edit menu Find dialog. In addition to searching for text you can also use this dialog to bookmark all matching text.
`Ctrl+G, F3`	Find next.
`Ctrl+Shift+G`	Find previous.

Ctrl+H	This short-cut key opens the Edit menu Replace dialog.
F2	Move the text caret to the next bookmark. You can use the edit toolbar to move to the next or previous bookmark.
Shift+F2	Move the text caret to the previous bookmark.
Ctrl+F2	Toggle bookmark on the current line. You can use the edit toolbar to set or clear a bookmark and to clear all bookmarks.
Alt+F2	Clear all bookmarks

Bookmarks tag a line for future reference. They are displayed as a blue mark to the left of the text. Bookmarks are kept as long as the current file is open; they are lost when you close the file. The easiest way to use a bookmark is from the Edit Toolbar. You can show and hide this from the Edit menu (when a text-based window is active), or by clicking the right mouse button on any toolbar or on the Spike2 application title bar and using the pop-up context menu that appears.

Text caret control

The text caret is a flashing vertical bar that indicates the current position. Do not confuse this with the I-beam mouse pointer which does not flash and which indicates the mouse position. Each time you click and release the left mouse button (we assume you haven't swapped the mouse buttons), the caret moves to the nearest character position to the click point. To select text with the mouse, click at one end of the text you want to select and drag (move the mouse with the button held down) to the other end of the text. You can also use the keyboard to move the caret and select text:

Key	Operation (+Shift to extend a text selection, +Alt+Shift for rectangular)
Left arrow	Move the caret one character left. At the start of a line it wraps to the previous line end.
Right arrow	Move the text caret one character right. You can move it into uncharted territory beyond the end of the line. It does not wrap to the next line.
Up arrow	Move up one line.
Down arrow	Move down one line.
Ctrl+Left Ctrl+Right	Move one word to the left/right. This means move the caret to the next boundary between a word character and a non-word character. Word characters are defined to be useful and vary depending on the view type. In a script view, they are A-Z, a-Z, 0-9, _, $ and %. In an output sequence, they are A-Z, a-z and 0-9. All other text views use A-Z, a-z, 0-9 and _ as word characters.
Ctrl+Up/Down	Scroll the text window up and down, leaving the selection unchanged.
Alt+Up/Down	Move the lines containing the selection up or down by one line.
End	Move the caret to the right of the last character on the line.
Home	Move the text caret to the first non-white space character in the line. If already at that point, move to the start of the line.
Alt+Home	Move the text caret to the start of the current line.
Ctrl+End	Move the text caret to the right of the last character in the file.
Ctrl+Home	Move the text caret to the left of the first character in the file.
Ctrl+] ([)	Start of next (previous) paragraph (after empty line)
Ctrl+\ (/)	Word part right (left).
PgUp	Move upwards by one window of lines.
PgDn	Move downwards by one page of lines.
Insert	Swap between insert mode caret | and over-type caret _

Cut Copy Paste Delete Undo and Redo

Some of these operations are also available from the Edit menu and the main toolbar.

Key	Operation
Ctrl+A	Select all the text in the document.

Ctrl+C Ctrl+Insert	Copy selected text to the clipboard. If no text is selected, nothing is copied. Some keyboards have Ins in place of Insert.
Ctrl+Shift+T	Copy the current line to the clipboard.
Ctrl+V Shift+Insert	Paste the contents of the clipboard into the text at the caret. If there is a selection, the selection is replaced.
Ctrl+X	Cut the selected text and copy it to the clipboard.
Shift+Del	Cut the selected text and copy it to the clipboard.
BackSpace	Delete the selection or the character to the left of the text caret.
Del	Delete the selection or the character to the right of the text caret.
Ctrl+Del	Delete word right. Add Shift to delete to the end of the line.
Ctrl+D	Duplicate the selection. If it is empty, duplicate the line containing the text caret.
Ctrl+Shift+L	Delete the current line.
Alt+Up/Down	Move all the lines in the current selection up or down by one line. Note that this does not work with the numeric keypad Up and Down keys. This is because holding down Alt, typing a number on the numeric keypad and releasing Alt enters the character with the typed in code.
Ctrl+Z Alt+Backspace	Undo the last interactive text operation. The editor supports more or less unlimited levels of Undo.
Shift+Ctrl+Z	Redo the immediately previous Undo operation.

Miscellaneous

These commands do not fit into any other category!

Key	Operation
Ctrl+U	Convert the selection to upper case. Add Shift for lower case
Ctrl+Add, Sub	Change font size (Add and Sub are numeric keypad + and –). You can also change the displayed text size by holding down the Ctrl key and using the scroll wheel on your mouse (if you have one). The View menu Standard Display command will remove any zooming. The script equivalent is ViewZoom().

Indent and Outdent

The structure of Spike2 scripts is often made clearer by indenting program structures. To make this easier, you can indent and outdent selected blocks of text to the next or previous tab stops. The tab size is set in the Edit menu Preferences option.

Key	Operation
Tab	If there is a multi-line selection, all lines included in the selection are indented so that the first non-white space character is at the next tab stop. If there is no selection, a tab character is inserted (or spaces to the next tab stop depending on theEdit menu Preferences settings).
Shift+Tab	If there is a multi-line selection, all selected lines are out-dented so that the first non-white space character on the line is at the previous tab stop. If there is no selection, the text caret moves to the previous tab stop unless it is already at one.

The Log view

The log window is a useful scratch text area that is always available. It behaves exactly like any other text window except that deleting the Log window only hides it; use the Window menu Show command if it is hidden or type Alt+W then 1. You can clear the contents interactively with the Edit menu Clear command.

You can type text into the window, or text may be added by a script or when an error occurs or to help us to diagnose problems. You can also save the contents of the log view as a text file. You can limit the number of text lines in the Log view from the Edit Menu Preferences General tab.

The log window is the destination for output from the `PrintLog(), DebugList()` and `DebugOpts()` script commands, for any output from the Evaluate toolbar, for interactive commands that need to report detailed information, to save information from the Cursor Regions and Cursor Values dialogs and is used to dump information if a script hits an error. You can also write the current sampling configuration as text to the Log window from the Sampling Configuration Channels tab and write useful information to it from the Spike2 About box and from Talker Info windows.

When Spike2 starts up, it writes where is loaded the sampling configuration from to the Log view and if there are start up problems, these are saved in the Log view.

Script uses can get the handle of this window with the `LogHandle()` script command. Once you have the handle you can manipulate the view and its contents.

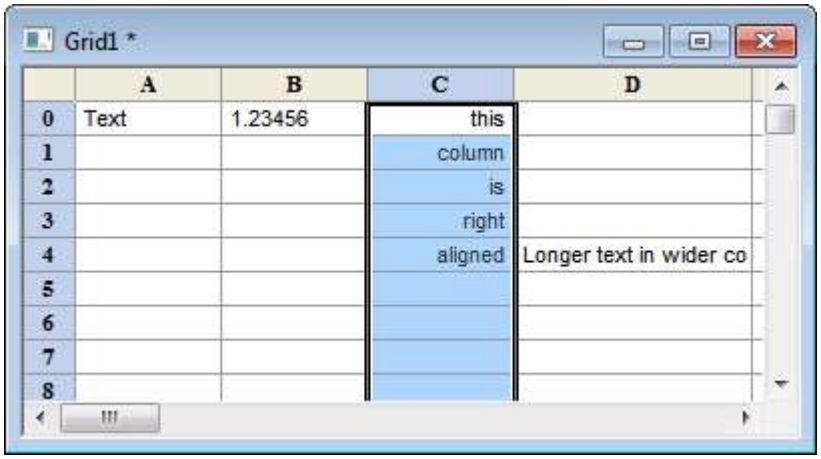
Grid View topics

You can create a Grid view interactively with the File menu New command or from a script with the `FileNew(17, mode%, rows%, cols%);` command. The grid can be controlled from menu commands and from a script.

You can save and restore grid views with `.s2gx` files (these are XML files internally). Grids support cut and paste of text using a Tab character to separate columns and an end of line character `"\n"` to separate rows. There is also the `GrdXxx...()` family of script commands to support the grid.

What the Grid is and is not

Our implementation of a Grid is not attempting to be a spreadsheet program. It is implemented as a utility to make it relatively easy to generate tabulated data. The initial implementation stores data as text, but we may extend this to allow data to be stored as integers, or real numbers with some user-control of formatting. There is no capability in the grid to refer to other cells or to allow expressions to be evaluated.



Grid properties

The underlying grid implementation is capable of giving each cell its own font, alignment, type, frame and colouring. In the current implementation we have restricted the grid as follows:

Size

The grid is currently restricted to 1000000 rows and 1000 columns (version 8 was limited to 10000 rows and 1000 columns). We may change this to a limit based on the product of rows and columns (so you can have more or one at the expense of the other). Rows and columns are numbered from 0,0 at the top left. There is a default size for grids created interactively (with an option to set the size and to resize an existing grid); from a script you can create a grid with a defined size with `FileNew(17, ...)` and you can resize it with `GrdSize()`. The grid is held in memory; you can exhaust memory (and make things run very slowly) with a huge number of grid cells. In particular, saving a huge grid to a file and reading it back can take a long time, especially if you modify properties of individual cells.

Headers
The grid has one header row at the top and one header column on the left. You can choose to hide or show these headers. By default the header column displays the 0-based row number and the column header displays columns as letters: `A-Z` for columns 0 to 25, `AA-AZ` for columns 26 to 51, `AAA-AAZ` for columns 702-727 and so on. The status bar at the bottom of the window (if enabled), displays the co-ordinates of the current grid selection as numbers. Double-click on a header cell to edit it. You are not allowed to set the column headers to empty or to white space. From a script you can set column headers with the `GrdSet(text$, col%, -1)` command.

Font
All cells in the body of the grid use the same font. The headers use a bold version of the same font. You can set the font with the View menu Font... command and from a script with `FontSet()`.

Row and column spacing
For simplicity, all rows are the same height, which is based on the font. Columns can be sized individually, or you can set a standard size, or you can optimise the columns widths to match the data displayed in them.

Alignment
You can apply Left, Centred or Right alignment on a column by column basis with the View menu Align column command or from a script with `GrdAlign()`.

Colouring
From [10.05] the you can change the background and text colouring of the entire grid, and from the script set it by column and by cell.

Grid cell properties
Cell properties are stored at three levels: a grid default (only used when creating a new column to set the column default), a column default (used for all cells in a column that do not set their own state), and cell properties attached to each cell. You can set the grid default colouring with the View menu Change colours... command and the script `GrdColourSet()` command. From the script you can set default column width, alignment and colouring. You can also set individual cell colouring and contents.

Grid view history

Spike2 version	Feature
8.05	Grids added to Spike2
8.06	`GrdSize()` command added.
8.10	Print Screen knows about grid views, `GrdAlign(n%, -2)` sets alignment of all columns.
8.18	Grid views support `Selection$()`
9.10	`FileList()` supports grid views
10.05	`GrdColourSet()` and `GrdColourGet()` added, extensions to `ViewColourSet()` and `ViewColourGet()`.
10.16	`GrdColWidth()` extended to get width of row titles, available column space and vertical scroll bar.

Information window topics

The information window is a specialised type of view which can display times and special information using a very large font for extra visibility. Information windows are attached to a time, result or XY view and when saved to disk these views will also save the information needed to re-create any associated information windows when the file is re-opened.

You can create an information window interactively with the View menu New Info window command or from a script with the `InfoOpen()` function, other script functions can be used to manipulate information windows. An information window display is set by a string which includes special fields that are replaced by other information (for example, the last data value on a channel or a timer).

Information windows can also display an image file instead of the standard background and either the area containing the buttons or the window title area (or both) can be hidden if desired. These and other options are available via a menu generated when you right-click on the information window.

In addition to displaying information and images, an Info window can be set to Speak the text contents. This can be very useful when you cannot look at the screen during an operation, for example, when using a microscope to make adjustments and you require audio feedback of the result of an adjustment.

From version [10.15], all features of the Information window can be set interactively from the Settings dialog; previously hiding all the Info window buttons and title and the ability to lock the font size were available from the `InfoSettings()` script command only.

When information windows are used from a script, they require idle time to allow them to update.

Copy file path, edit title

When working with file views (Time, Result, XY, Grid or text-based) you can copy the full path to the file (if the view has an associated file) by right clicking on the view title bar and then selecting the Copy path item that appears. This is especially useful if you have set the operating system to not display full path names.

The script language equivalent of this is `EditCopy(FileName$());` to copy the path to the file associated with the current view to the clipboard.

The context menu that pops up also has an option to set the Window Title. The script equivalent of this is the `WindowTitle$()` command.

Other file types

In addition to the document file types (data, result, XY, Grid, text, script and output sequencer), Spike2 also uses:

Result view files

These hold result views (file extension `.srf)`. Result views are created as the result of an analysis, or from the script language, not with the File menu New command. Version 11 reads result view files written by previous versions of Spike2. We write result view files in the oldest format compatible with the data for backwards compatibility. Files holding raster data with event times that exceed the 32-bit timing capability of older versions of Spike2 are only readable by Spike2 version 8 and later.

Resource files

Spike2 creates and reads resource files with the extension `.s2rx`. These files hold additional information about data files, such as screen positions, channel order, virtual channels and Process information. Resource files are associated with data files of the same name, but with the extensions `.smrx`, `.srf` and `.sxy`. They hold configuration information so that Spike2 can restore the display. These files are not essential to Spike2 and if deleted, the associated data file is not damaged in any way.

Old format resource files

Originally, Spike2 wrote resource files with a binary format and the extension `.s2r`. This format was compact and fast, but inflexible. We changed to the current XML-based format at Spike2 version 7.11 and versions 7 and 8 can read old format files and write the modern format. Spike2 versions 9 and later do not read old-format resource files. If you have old-format resource files you can use Spike2 version 7 or 8 to convert them. Open and close the associated data file to write the new format. The Spike2 installation media includes an older version of Spike2 for this purpose.

Configuration files

Spike2 stores sampling configuration information in files with the file name extension is `.s2cx`. Spike2 versions 9 onwards do not real old format files with the extension `.s2c`. To convert them, load the configuration into Spike2 version 7 or 8, sample with them, then save the configuration.

Filter bank files

These files hold descriptions of digital filters and have the extension `.cfbx`. They are used by the Analysis menu Digital filters command. Spike2 does not read the old format `.cfb` files. To convert them open them in Spike2 version 7 or 8 and save them.

Multimedia files

These files have the extension `.mp4` and `.avi`. They are created by the separate `s2video` program and are opened by the View menu Multimedia Files command.

Compatibility with Spike2 version 7

The `.s2rx`, `.s2cx` and `.cfbx` files hold their contents in XML format. Spike2 version 7.11 onwards can read these files; earlier versions of Spike2 cannot. The compatibility of `.srf` files is described above.

The Spike2 command line

When Spike2 starts, it checks the command line for option switches and files to load. If there is no command line, Spike2 looks in the current directory for `startup.s2s` and runs it if it exists. If `startup.s2s` is used or the command line loads a file, start up messages that need a user response are suppressed.

The command line holds options and file names separated by white space characters (space and tab). If a file name contains spaces, you must surround the file name with quotation marks. Options start with / or – followed by a character to identify the option.

`/M` When Spike2 starts, it checks if it is already running and quits if it is. `/M` removes the check, allowing multiple copies to run. You need a Spike2 licence for each copy except when using multiple synchronised 1401s to capture related data on one computer under the control of a single operator, when one licence is sufficient. Set separate file names for each 1401 in the Automation tab of the Sampling Configuration.

`/Un` `n` is 1-8 to select a 1401 when you have more than 1. The default is `/U0`, which uses the lowest-numbered unused 1401. You set a device number in the CED 1401 device settings in the Device Manager (My Computer->Properties->Hardware).

`/Q` Quiet start. Suppress all message boxes, the Spike2 "splash screen" and the *new Spike2 version detected* help page. It does NOT suppresses the check for a sampled file that was not closed properly as this is considered to be too important to suppress.

`/Tn` Where `n` is 0 to 9 and sets the pipe number to use when communicating with Talkers. This is for use when sampling with multiple copies of Spike2 running and you need to associate a particular copy of Spike2 with a particular Talker. With this option set, Spike2 will only detect Talkers that have the same `/T` option set.

The remaining items in the command line are assumed to be file names. Spike2 attempts to load the files in command line order (from left to right). The files must have extensions so that the file type is known. If a script file is included in the command line, Spike2 runs it before continuing with the remainder of the command line.

As an example, suppose we want to launch Spike2 so that it automatically opens a data file called `lots of data.smrx` and runs `doit.s2s` to process it. Follow these steps:

1. Create a short cut to `sonview.exe` (this is the Spike2 program). To do this right-click on the Spike2 program icon and choose Create shortcut.
2. Right-click on the new short cut and select Properties and open the Shortcut tab.
3. Add `"lots of data.smrx" doit.s2s` to the end of the Target field. The quotes are needed because the file name includes spaces.
4. Set the Start in field to the folder that contains your files.
5. Click OK.

This example assumes that both files are in the same folder. You could also have included the full path to each file in the command line.

Unrecognised command line switches and the System$ command

Spike2 ignores command line switches that it does not recognise. If you have a command line that runs a script, it can be useful to pass other information to Spike2 using the command line. As anything that does not start with / or - is treated as a file to be loaded, if you want to pass additional information, you should make sure it has a

switch on the front. For example, if you want to pass a file to be processed by a script, you could have a command line:

```
pathtosonview/sonview.exe -XMyFileName.dat runme.s2s
```

In this case, `System$(0)` would return `"/XMyFileName.dat"` and `System$(1)` would return `"runme.s2s"`. To simplify parsing, the `System$` command converts options started with a - character into options started with a `/`. To pass in a string that holds spaces you must wrap the string in quotation marks:

```
pathtosonview/sonview.exe "-XMy File Name.dat" runme.s2s
```

and `System$(0)` would return `"/XMy File Name.dat"`.

Shell extensions

Previous Spike2 versions added shell extensions that displayed additional information in Windows Explorer when the mouse pointer hovered over Spike2 data and script files. In Windows NT2000 and XP you could also display file comments. These shell extensions were held in `SonCols.dll` and `SonInfo.dll` in the Spike2 installation folder. Regretfully, we removed these shell extensions at version 8, because:

1. From Vista onwards, Microsoft has removed the underlying operating system support used by `SonCols` (which displayed file comments in Explorer if you enabled the Comments column).
2. There are problems with library versions that make installing `SonInfo` problematic. These can prevent libraries being updated correctly when new versions of Spike2 are installed, leading to obscure crashes.

We may reinstate them at a future date if we can do it without causing problems.

Removing the shell extensions

To remove shell extensions installed by a previous version of Spike2 without uninstalling the old copy of Spike2, open a command prompt and navigate to the old Spike2 installation folder and type:

```
C:\>cd \Spike7                    Change to the old Spike2 folder
C:\Spike7>regsvr32 /u SonInfo     For all Windows versions
C:\Spike7>regsvr32 /u SonCols     Only for NT2000 or XP
```

You may need to run as administrator to do this (particularly for `SonCols`). To do it:

1. Click on the start menu at the bottom left of the screen.
2. In the box with "Search programs and files" type `CMD`
3. A list of matching files should appear including `cmd.exe`.
4. Right-click on `cmd.exe` and select "Run as administrator".
5. In the command prompt you can then issue the commands listed above to remove the shell extensions.

64-bit operating systems

Before Spike2 version 8, all versions of Spike2 were 32-bit programs, even if the operating system was 64-bit. From Spike2 version 8 onwards, by default, we will install a 64-bit version of the program on 64-operating systems and a 32-bit version on 32-bit operating systems. The advantages of 64-bit code on a 64-bit operating system are:

1. The 64-bit version of the CPU has more registers and instructions; "typical code" runs maybe 10% faster.
2. Calls into the operating system do not transition from 32-bit Spike2 code to 64-bit system code and back.
3. A 32-bit program is limited to a 4 GB memory space, and of this, quite a bit (typically 1GB) is reserved for the 32-bit operating system. With a 64-bit program you can use a lot more memory.
4. The new son library uses 64-bit integers to hold times; these are faster to manipulate with 64-bit registers than with 32-bit registers.

There are some disadvantages, too:

1. 64-bit code is typically larger (for example the `sonview.exe` file is 20% larger as 64-bit code).
2. Pointers in memory are double the size, so memory usage increases.

Unless your system has limited installed memory or you need access to external features (such as importers or signal conditioners that only exist as 32-bit code), we recommend that you use the 64-bit version of Spike2 on a 64-bit operating system.

Since the advent of Windows 10, It is extremely unusual for users to install Spike2 onto anything other than a 64-bit operating system. It is likely that at some point Spike2 will become 64-bit only; this will only happen when there is no need for 32-bit versions due to compatibility requirements with non-CED software.

Mouse buttons

Throughout the Help we assume that you have not swapped your mouse buttons over. That is, the left button is the standard click and the right button is used for context menus. If you have swapped your mouse buttons, then you must interpret the descriptions accordingly.

The left-hand mouse button selects items. Most items that are selectable will indicate their state by a change in appearance. For instance the title bar of a selected window or a channel number in a time or result view.

The right-hand mouse button generally opens a 'context' menu. This is a list of items that we think are likely to be useful in the current situation for the item you clicked on.

Folders (or where are my files?)

This topic applies to Spike2 versions 7 onwards, so we refer to installation directories as `SpikeN` where `N` stands for `7, 8, 9` or `10`, as appropriate. It should help users who are migrating from the old-style arrangement of running Spike2 from a top level folder such as `C:\SpikeN` to the Microsoft-preferred installation within `C:\Program Files`. If you are interested in the special Spike2 folders created by the installer, see here.

Note: The example file paths in this section relate to English language versions of Windows. `Program Files` can have different names depending on the language set for the system. Similarly, `My Documents` can be different in different languages.

Previously, we installed Spike2 in a top level folder such as `C:\SpikeN`. While this practice was satisfactory when used with earlier versions of Windows, it has become deprecated, mainly for security reasons. We have altered Spike2 so that it can be installed in the protected `C:\Program Files` folder as expected by Microsoft. Some users prefer to keep their existing arrangements so we still allow installation into `C:\SpikeN` or similar; we do not recommend or support this and the ability to install in this way will be removed at some future point.

The main change caused by installing inside `C:\Program Files` is that this location is expected to be read-only as seen by users. While there is no problem in storing data files, sampling configurations, scripts and other files in `C:\SpikeN`, you are not allowed to modify anything inside `C:\Program Files`. Therefore as part of the Spike2 installation new folders are created for the current user's data, for data shared between all users and for data generated by the application.

New folders created by the installer

User data folders

User data folders are used to hold data that is directly saved or loaded by the user, for example data files, sampling configurations and scripts. Two such folder are created by the Spike2 installer:

- Current user data: called `SpikeN` in the `My Documents` folder for the current user. When installing Spike2 inside `C:\Program Files`, the installer puts all data intended for the user's direct use here. The folder contains the `Data, Scripts, Sequence, ExtraDoc` and `TrainDay` folders and an empty `Include` folder for your own script include files. If there are other Spike2 users on the system, they should create a `SpikeN` folder inside their own `My Documents` folder. When searching for a user data folder, Spike2 seeks the current user data folder first and if it is not found, looks for the all-users data folder:
- All-users data: called `SpikeNShared` in the `Documents` folder provided for all users of the system. This is typically `C:\Users\Public\Documents\SpikeNShared\` The folder contains a copy of the `Data, Scripts` and `Include` folders. These folders are available for multiple system users to store and shared data and files.

Application data folders
Application data folders are used to store data that is generated by the application without any direct user involvement, for example the default filter bank settings and the last-used and default sampling configuration. As for the user data folders, two application data folders are created by the Spike2 installer:

- Current-user specific. This folder location has varied with the operating system version but in Windows 7 onwards the folder is `C:\Users\User\AppData\Local\CED\SpikeN` where *User* is the user name. From [10.02], if this folder does not exist, Spike2 attempts to create it during program start up. If the folder is not found and a different user is logged-on to the system they must either create it manually or rely upon the shared all-users application data directory. This is the location used for saving importer configuration files, filter bank settings, default sampling configurations and the text version of the graphical sequence. When searching for an application data directory, Spike2 generally looks for the current user's application data directory first and if this is not found looks for the all-users application data directory:
- All-users application data. This folder location varies with the operating system version but in Windows 7 onwards the folder is `C:\ProgramData\CED\SpikeN`. If there are multiple Spike2 users on the system this directory ensures that there is always a suitable folder available for application data. However, there can be Access Rights problems with this folder for users without Administrator privilege. This folder is created when Spike2 is installed.

Usage tip
These folders can be tricky to find, especially if you are under pressure. You can make this easier by adding them to your Favourites folder. The Favourites folder holds short-cuts to other folders; this means that you can give the folders descriptive names in the Favourites (otherwise three of the four folders end up called `SpikeN`, which is not so helpful).

The Help menu About Spike2 box Copy button includes these folders as part of the information copied to the clipboard.

Moving your files to a new-style location
After installing inside `C:\Program Files`, the original `C:\SpikeN` folder (or whatever was used) will be unchanged, holding both your own files and the previous installation of Spike2. You can continue to load sampling configurations from and save data files into `C:\SpikeN` just as before and some users may prefer to do this, perhaps after cleaning up (see below). Alternatively you can choose to keep your data and other files in the new location. To do this, copy (or move) all `.smr, .sxy, .s2rx, .s2cx, .s2s` and `.pls` files, plus any other files you might want to keep and any folders of your own into the `SpikeN` directory inside your `My Documents` folder. Once you do that you will be able to use your data files, scripts, sampling configurations and other files from the new location.

If the old location held `.s2r` or `.s2c` files, these are not read by this version of Spike2; you can use Spike2 version 8 to update them to the modern formats.

There can be issues in moving files. If you copy (rather than move) files to the new location and if they are referred to in scripts or sampling configurations by their full path name, the original versions of the files will be used, which can cause confusion. After moving your data to a new location you should check the following:

Application-level issues:

- The Sampling page of the Preferences dialog sets the folder for the temporary `smr` data file that is created by sampling. By default in older versions of Spike2 this was set to be the application data folder, so you may find that it is set to `C:\SpikeN`. You can set this as you prefer, or clear the path and click OK and Spike2 will set it to an application data folder. Do not set a folder in `C:\Program Files` as this may cause problems with write permissions.
- Global resources - some options for the global resource file location use the Spike2 installation folder. While this location will still be used, Spike2 now uses a list of folders which are searched in turn for the required file. The list starts with the current user's data folder inside `My Documents` - if you are using global resources you should make sure the resource files are in a suitable location; the `SpikeN` folder inside `My Documents` is probably the best place.
- The Sample and Script bars hold lists of sampling configuration and script file names. These lists hold the complete path to the file so if you use either of these you should update the file names so that they point at the new file locations.

Data and XY file issues: There should be no problems with the data files themselves as they do not contain any information relating to folder locations. However if a background image file has been set up for a data or XY file the background image settings will contain a complete path to the image file which should be updated.

Sampling configurations: Sampling configuration files can contain a number of folder paths which may need to be adjusted. Note that (as documented here) Spike2 now only writes sampling configuration files using the new XML file format with a new `.s2cx` file extension so your adjusted sampling configuration files will have to be converted to the new format by using them for sampling before they can be saved as XML. Therefore the changed files will all use the new file name extension and your old configuration files will remain unchanged. You should take care to only use the new-style configuration files once they have been adjusted and checked.

- Sequence file. If the sampling configuration Sequencer page specifies a sequence file, you may need to edit the file path to point to the new file location.
- Automatic file-naming. If the sampling configuration Automation page sets a folder for automatically-named files this path might be unsuitable for the new arrangement and should be adjusted.

Sequence files: If the file uses `#include` commands you may need to check that these still work.

Script files: If the file uses `#include` commands you may need to check that these still work. If the script uses file path string constants, these may need adjusting. Any use of `FilePath$(2)` to get the Spike2 installation folder is likely to give problems as you should not create or write to files inside Program Files. The parameters to `FilePath$()` have been adjusted to allow you to find the `My Documents` folder where Spike2 data files are installed - we suggest that you change to using this location wherever possible.

Cleaning up the old installation location without moving your own files

If you change from `C:\SpikeN` to `C:\Program Files` and you want to leave your own files in `C:\SpikeN`, you may want to clean out redundant files. If the new installation is a different Spike2 major version, for example moving from version 8 to version 11, the easiest way to remove the old installation without deleting files that you created is to un-install the old version using the system control panel. However, if you are moving within the same major version this would delete the current version; you can do the following instead:

- Delete the `1401`, `BaseDemo`, `ExtraDoc`, `Export` and `import` folders from the old install location. If you not put any of your own files into them, you can also remove the `data`, `scripts`, `sequence`, `include` and `trainday` folders.
- Inside the old Spike2 installation folder itself, delete all `.exe`, `.dll`, `.chm`, `.s2l`, `.t14` and `.tip` files. These are files specific to the Spike2 installation and will have been replaced by the newly installed files.

See Also:

Special Spike folders, View types and files, Other file types, The Spike2 command line, Shell extensions, 64-bit operating systems, Mouse buttons, Unicode, Resource limitations, File recovery, Drag and Drop

Special Spike2 folders

The standard Spike2 installation process creates several file system folders that Spike2 expects to find and use. Some of these data folder are created based on the current user name. These folders are:

Public shared data

This folder is typically: `C:\Users\Public\Documents\Spike11Shared\` and is used to store:

- Example data file in the `Data` folder
- Standard script include files in the `Include` folder
- Standard scripts in the `Scripts` folder
- Talker licence information (from [10.07] onwards

This folder and the contents are created and set by the Spike installer (also see below). Most users can survive without this folder, but various Spike2 examples and some of the scripts on our web site rely on the standard `Include` folder being found. It is the recommended place to save items that should be accessible to all users of Spike2. From a script, you can get the path to this folder as `FilePath$(-5);`

User data

This folder is typically: `C:\Users\<UserName>\Documents\Spike11\`, it is:

- Where we suggest you store user-specific data, scripts, configurations, sequences and the like
- Part of the search tree used to locate `#include` files
- Part of the search tree when looking for global resources
- Fallback location for information about used Talkers.

This folder is created for the current user by the Spike installer (also see below). Most users can survive without it, but it is useful for saving user-specific items. From a script, you can get the path to this folder as `FilePath$(-4);`

Local application data

This folder is typically: `C:\Users\<UserName>\AppData\Local\CED\Spike11\` and is:

- Used to store `S2PSEQ$.PLS` - the compiled version of the graphical sequence file for sampling.
- The default New file path if no path is set in the Registry for the Edit Preferences or a problem with the supplied path.
- The location to store the `last.cfgx` sampling configuration and the default location for `default.cfgx`.
- Part of the search path for loading default configurations and `#include` files.
- The default location for saving IIR and FIR filter banks (`filtbank.cfbx`).
- From [10.07], information about used Talkers in the file `sp2talks.datx`.

This folder is created by the Spike installer (also see below). The lack of this folder can cause problems on system that have separate logins for different users. This is because if this folder is missing, Spike2 will try to use the `ProgramData` folder in its place. If the user has insufficient privilege to access that folder, various aspects of Spike2 may cease to function. From a script, you can get the path to this folder as `FilePath$(-3);`

ProgramData

This folder is typically: `C:\ProgramData\CED\Spike11` and is used to store:

- Crash Dumps in the `CrashDumps` folder.
- Prior to [10.07] this was used for talker licence information

This folder is created by the Spike installer (also see below). It is also one of the places searched to find the last used and default configurations and the FIR and IIR filter banks. If this folder is missing it is likely that your copy of Spike2 was not installed properly (perhaps it was copied from another location), or that you do not have sufficient privilege to access the location. From a script, you can get the path to this folder as `FilePath$(-6);`

Actions on start up

Spike2 version [10.02] onwards tests for the presence of 4 folders during start up. If any folders are missing, a message is written to the log view:

```
The following data folders are missing:
Public, User, Local application, ProgramData
Either reinstall Spike2 for the current user or create them manually
Lookup 'Missing folders' in the Spike2 Help Index
```

The list of missing folders will hold one or more of the 4 possibilities. The start up process attempts to create these folders if they are missing, so it is unusual to see this message. If you do, it likely means that you do not have sufficient privilege to create the missing folder(s) and you will need to consult your system administrator.

Unicode

Before Spike2 version [8.03], all text in Spike2 was represented by 8-bit characters with codes in the range 0 to 255 (hexadecimal 0x00 to 0xff). In the 8-bit character world, characters with codes 0 through 127 are pretty much universal, for example character 32 (0x20) is a space anywhere on the planet. These are sometimes referred to as the ASCII character set. However, character codes 128 to 255 (0x80 to 0xff) had meanings that depended on which code page your computer was set to. As some cultures have many more characters than can fit in the 255 available, in some cases pairs of 8-bit characters were used to represent characters. The result of this was that you could use characters with codes above 127 to represent national characters in text views, but if you sent a data file or text file that used such characters to someone with a computer set to a different code page, the result was unpredictable (and often gibberish).

The solution to this problem is to use a character encoding that does not limit us to 8-bit characters. The Unicode Consortium has defined a standard set of characters (1,112,064 valid code points) that can encompass all the cultures on the planet. The characters are grouped into 17 'planes' of 65536 characters, of which the most important is the first, known as the Basic Multilingual Plane (BMP). Character codes 0-127 of the BMP have the same meanings as the ASCII codes 0-127, which is very convenient for backwards compatibility. The BMP holds the vast majority of the characters used in everyday communication by most cultures. The remaining planes hold more infrequently-used and specialist characters, such a music clefs, historic scripts (hieroglyphs), playing card symbols and the like.

There are a variety of ways of storing Unicode characters in memory and in data files. Only two need concern us: UTF-8 and UTF-16LE.

UTF-8

This is a method of storing Unicode characters using a sequence of 8-bit characters. It has the property that if your text only uses character codes 0 to 127 (0x00-0x7f), then it is identical to ASCII text. So any Spike2 script that you have that does not use any characters with codes above 127 is already coded in UTF-8. Character codes in the range 128 to 2047 (0x80-0x7ff) are coded in two bytes, character codes in the range 2048-65535 (0x800-0xffff) need three bytes and character codes greater than that take 4 bytes. If your text is mainly ASCII, or uses only European characters, this is the most compact coding. The encoding is arranged so that you can always tell if a particular byte is the start of a character or the continuation of a character.

We use UTF-8 as the encoding for all external files (scripts, output sequences, XML resources, text in Spike2 data files). The text editor within Spike2 uses UTF-8 internally (though this is of no consequence to most users). UTF-8 is probably the most common format used for storing Unicode in files; in July 2024 it accounted for 98.3% of all web pages with a known encoding. All other encodings share the remaining 1.7% and this share is falling.

UTF-16LE

This uses 16-bit Little Endian values to store the characters. This is the format that Windows uses internally for all text. Unicode builds of Spike2 uses this format internally everywhere except where we have to use 8-bit characters (1401 communications, serial line input and output, external text files, resource files, the script editor, data import, MATLAB interface). In UTF-16LE, characters in the BMP (Basic Multilingual Plane) are all represented by one 16-bit value. Characters in all other planes require a pair of 16-bit values.

Surrogate pairs

Character codes in the range `0xd800` to `0xdbff` are reserved for the first (lead) of a two word pair and characters in the range `0xdc00` to `0xdfff` are reserved for the second (trail) part of a two word pair. These are known as surrogate pairs; they encode all characters that are not in the BMP.

Updating to a Unicode version of Spike2 [8.03]

Spike2 version 8.03 and onwards uses Unicode characters internally and stores output text in UTF-8 compatible files. When you update from an ASCII (8-bit character) to a Unicode version of Spike2 our aim is that you should see no difference. We achieve this by assuming that any characters that we read above code 127 that are not valid UTF-8 sequences are local code page characters, and we convert text appropriately:

- When we open a script or sequencer or text file in a text editor, we attempt to guess the format. If it starts with a BOM (Byte Order Mark: a special code that indicates the file format) for UTF-8 or UTF-16LE, we assume that is the format. Otherwise, if it holds no characters above code 127 or if it holds correctly formed UTF-8 (this is easy to detect), we assume it is UFT-8. If the text is an even number of bytes long and holds 0

bytes in odd positions, we assume it is UTF-16LE. If it is none of these, it must hold 8-bit characters above code 127 and we assume these are code page characters and we translate the text using the local code page set on the computer. It is possible that we might be fooled into interpreting code-page text as UTF-8, but this is unlikely for text of any length.

- When we read text from a resource or from a Spike2 data file we test to see if it holds correctly formed UTF-8. If it does not, we assume it holds code-page characters.

However, for performance reasons, we do not convert text from `#included` script files. If you have included a script file that use characters codes above 127 you must convert these by opening it in the script editor and saving it. Note that we do not count changing the format of a text, script or sequencer file from code-page based to Unicode as a change (the result should appear identical). To force a converted file to write you will need to use the File Save As command or edit the file.

Backwards compatibility

If you update to Unicode by saving a script or a modified data file or a resource and you used character codes above 127, the modified files will no longer read correctly into non-Unicode versions of Spike2 (before version 8.03). This will not stop you opening the files or resources, but any character codes above 127 will not appear correctly. If you write all your text as ASCII characters this will not be a problem for you; you will see no change.

The `.smr` (32-bit) file format has limited size fields for channel titles, units and comments. The channel units field can be a problem as this is limited to 5 8-bit characters. If you use extended European characters, these typically code as 2 UTF-8 bytes each, so you are limited to 2 of these, or 1 of these and three ASCII characters. If you use a Far East typography, such as Chinese or Japanese, each character uses 3 UTF-8 bytes, so you only have space for one such character and 2 ASCII codes.

If you require 100% backwards compatibility with old versions of Spike2 you must stick with the `.smr` 32-bit file format and use only ASCII character codes for channel titles, units comments and file comments.

Character set used in scripts and sequences

The characters that are accepted as numbers, punctuation, keywords and variable and constant names in the script language and in the output sequencer are restricted to the ASCII set. There are additional sets of numbers, punctuation and a-z and A-Z in Unicode. For example Japanese defines wide versions of numbers and A-Z; these cannot be used. You have a free choice of Unicode characters for comments and for string literals in the script language.

Resource limitations

There are some limitations that are imposed on us by the operating system and the computer environment. Some of these are obvious, some less so:

Memory

Your computer will have a fixed amount of memory (typically several GB). This is shared out between all the competing programs as they require it. If you use a lot of memory, for example by using Memory buffer channels and copying vast quantities of data into them, Windows will try to cope with this (especially if you run under a 64-bit operating system), but at some point the operating system will start swapping allocated memory to disk. This makes things much slower. If you keep allocating more memory, at some point Spike2 will start reporting that it is out of memory. The more memory your system has, the more you can use.

If you run under particularly restrictive rights you may find that you get error -544 when Spike2 tries to allocate the memory Working Set.

Disk space

This is a pretty obvious limitation. Once you have filled your disk, you cannot use more space. If you use a Hard Disk Drive (HDD) rather than a Solid State Drive (SSD) you may well find that once a disk becomes significantly full (this may be as low as 50% full), disk operations start to get slow. It is usually well worth periodically cleaning up your disk system by deleting unwanted files. It may also be worth de-fragmenting your disk (though this seems less important in modern versions of Windows).

Solid State Drives (SSDs)

If you are fortunate enough to use a SSD as your drive, fragmentation is not a problem. However, be aware that SSDs (in 2020) have limited rewrite capabilities. You SSD manufacturer will have specified a TBW figure (Total Bytes Written) that they 'guarantee' you can write. If you divide the total drive size in bytes by this figure, you get the number of times you can rewrite every block of memory on the drive. This figure is typically in the range 300-1000 in 2020. This does not mean that the drive will fail at this point; it is likely that you can write much more than this, but is does mean that SSDs are more suitable for write infrequently, read often data than for streaming data at high speed all day long.

If you have a 1 TB drive that you use only for data capture, sampling continuously at an aggregate waveform rate of 1 MHz, you will use around 5 GB/hour. So one total disk write will take 200 hours, and disk endurance is not much of a problem. However, if you have a 256 GB SSD with maybe 100 GB already used for the operating system and other programs (which will also be writing data), this may start to be a consideration and you might consider a separate drive for data.

GDI and User Handles

Windows tracks the use of GDI objects (things like fonts, bitmaps, brushes, pens and drawing surfaces) and User objects (things like desktops, windows, menus, cursors, icons and menu short cuts). For reasons tied up in the history of Windows and for backwards compatibility, these have 16-bit identifiers, which meant that there can only be 65535 GDI and 65535 User handles. Each application is limited to 10,000 GDI objects and 10,000 User objects. Most programs do not use a huge number of either (typically a few hundred of each), so this is not usually a problem.

It starts to be a problem when an application creates lots of windows as each window will use quite a few handles. Beware that data channels and axes all live in their own windows). You can track the number of GDI and User objects with the `App()` command. Each new view (visible or not) uses some 20 to 100 of each of these objects. This means that you can run out of handles by opening 300-400 data views (or even fewer if they all have a lot of channels). We refuse to allow you to create new views if you have used more than 9900 of either object. If this happens, you are probably running a script that is not deleting windows that it has created.

Symptoms of resource exhaustion

Windows does not manage running out of resources very gracefully. The symptoms include:

- Messages from Spike2 warning that there was not enough memory or resources to complete an action
- Missing areas or channels in window or screen repainting problems
- Text appearing in degraded fonts
- Poor system performance (to the point of appearing to stop)
- Crashes

If you start to suffer from these problems while using Spike2, do the following:

- Save any volatile data in case things get so slow that you cannot continue.
- Check for a lot of hidden windows in the Windows menu Show list or in the Windows... command. Close down (and save, if required) unwanted, hidden windows.
- If you are running a script, check that you are closing all windows that you have opened. Each `FileNew()` or `FileOpen()` should have a matching `FileClose()`.
- If you are running a script, have you used a huge array space, and could you reduce it? Script users can use `DebugHeap()` and `App(-4)` commands to get information on used memory resources.

If none of the above fixes it and you suspect that some operation in Spike2 is 'leaking' resources, please let us know and give us the information to reproduce the effect.

File recovery

To protect against power failure and program crashes, Spike2 makes backup copies of modified but not saved text-based, result and grid views once every 5 minutes. These file types are held in memory and changes would otherwise be lost in the case of an uncontrolled program exit. You can recover these files the next time you start Spike2.

Data files (`.smrx` and `.smr`) are not saved in this manner as they could take a very long time to save and they are not held in memory. If power is lost or the program crashes during data capture, there are other arrangements for data recovery.

Drag and Drop

You can drag Spike2 document and configuration files from other applications and drop them on Spike2 unless Spike2 is sampling or is running a script and has forbidden use of the File menu or is busy. This works for applications that use standard formats to do this. The files are opened in whatever location they are dragged from.

Microsoft Outlook

Microsoft Outlook uses its own format for Drag and Drop. From Spike2 [10.20] we have enabled use with Outlook. To do this, we have to store the dropped files (as they may not exist in a physical form). We generate a folder called `Outlook` (or `Outlook(n)` if this folder already exists) in the current user `Downloads` folder. This is typically located at:

`C:\Users\<username><.domain>\Downloads`

Large files can take a detectable time to save.

4: Sampling data

Sampling data

If you worked through the *Getting started with Spike2* section you already have most of the skills to sample data. Sampling a new document is the same as working with an old one, except that the length increases with time.

Hardware requirements to sample data

To sample data you need a CED 1401 interface: Power1401, Power1401 mk II, Power1401-3, Micro1401 mk II, Micro1401-3 or Micro1401-4. If you have a 1401*plus* or the original micro1401 you can capture date with Spike2 version 7 (contact CED if you need to install this).

You can extend your data capture, or capture data without a 1401 using Talker interfaces.

Sampling configuration

Before you start to sample data you must set a sampling configuration. This defines the channels to capture, the file type and timing resolution, how the sampling is to start, time limits for sampling and outputs to generate during sampling. You can load a previously saved configuration with the File menu Load Configuration command, manipulate the configuration from a script or you can set the configuration interactively with the Sample menu Sampling Configuration dialog. This dialog holds several pages activated by the tabs at the top, each page controlling a different aspect of data capture.

The title bar holds the name of the configuration file from which the configuration was read. There is a * at the end of the name if the configuration has been changed. You can extract the current configuration file name and the last user-defined name by right-clicking on the title bar. Options are:

Copy Path	Copy the path to the current configuration file to the clipboard.
Copy Last	New at [10.07]. This is present if the current path ends with `LAST.s2cx` or `DEFAULT.s2cx` and Spike2 knows the file that the current configuration was derived from. It copies the last known configuration source file name to the clipboard.
Close	Close the dialog

You can copy the full path to the configuration file by right-clicking in the title bar and selecting the copy option from the drop down menu. This is equivalent to the `SampleConfig$(0)` script command. If the configuration is modified by sampling, `<S>` is added to the name in the title bar.

The OK button accepts the current state, Cancel rejects any changes you have made since you opened the dialog and the Run now button opens a new data document, ready to sample. You can also open this dialog from the toolbar.

The sampling configuration also holds windows positions, channel arrangements and settings for on-line data processing into result and XY views or into time view data channels. These properties are set by opening a file for sampling, setting the window, channel and processing as you need it, then saving the configuration. To see the script language equivalent of a configuration, load or set the desired configuration, turn recording on, open a new file ready to record, then turn recording off.

You can open this dialog when you are sampling. Most items are disabled as most features are fixed during data acquisition. However, you can adjust and apply changes to the graphical sequence editor.

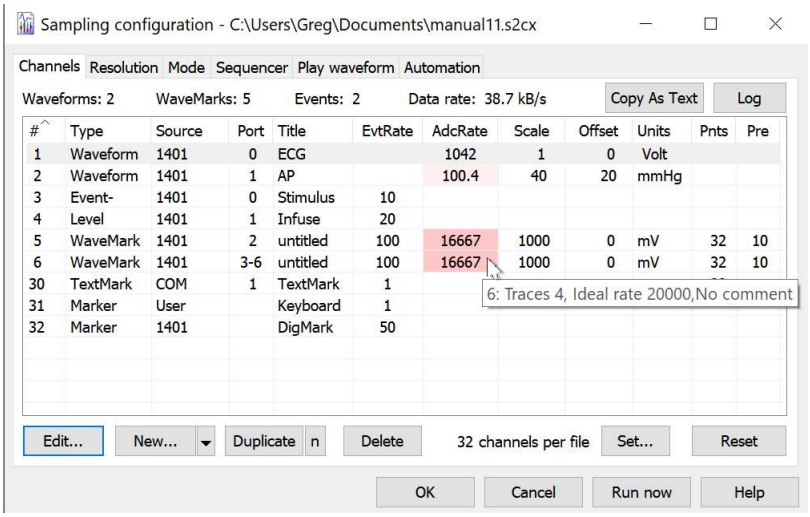
Channels

The Channels tab lists the channels to sample. Channels are taken from a 1401 interface, from the keyboard (TextMark data can also be read from a serial port), or from a Talker, or are Derived from one of these. For most channels there is a free choice of channel number, but there are three Special channels (keyboard marker, Text Marker and Digital Marker) that have forced numbers.

The top line of the Channels tab lists the number of Waveform, WaveMark and Event channels and an estimate of the overall data rate written to the data file. As we cannot know the rates of event channels in advance, we use the estimated maximum sustained rate set for each event channel. Note that the length of the

available data buffering (used in triggered sampling mode to save old data) is approximately 8 MB divided by this rate. There are also buttons to copy the configuration as text to the clipboard or to the Log window.

The main part of the dialog holds the list of channels that are configured for sampling. If the list length exceeds the available space, you can scroll the list. Alternatively, you can resize the dialog by dragging the top or bottom edges vertically. You can click the header of the channel list to sort the channels by most of the fields; the ^ symbol on the # column (below) indicates that the list is sorted on the # column in ascending order. Click the channel header a second time to invert the sort order.



If you hover the mouse pointer over a channel in the list, a tool tip appears holding additional channel information (ideal waveform sampling rate, debounce time for event and marker channels, WaveMark trace count, channel comment, explanation of problems with Talkers). This dialog is resizeable; drag the bottom edge to make more space to display long channel lists.

The # column holds the channel number as it will appear in the sampled data file.

The Type column displays the channel type as it is sampled. This is followed by an asterisk (*) if the channel has an attached process. A channel with a process may be stored in the file as a different type. For example, a Waveform channel with a process will be stored as a RealWave channel.

The Source column indicates the device from which the data is obtained: 1401 for a channel sampled by a 1401 interface, COM for a marker taken from a COM port (serial line), User for keyboard input, or the name of a Talker. If this is a Talker that is not available, the entire line is drawn in grey text. If the Talker is available, but there is a problem with the channel, the text is drawn in red.

The Port column indicates the physical connection on the source device that is used for the channel.

The EvtRate column is the expected (estimated) event rate for the data.

The AdcRate column displays the sample rate that waveform-based channels will use to collect data. If this is too different from the Ideal rate set for the channel, the background of this field set to a shade of red (there are controls in the Resolution tab to improve the match). The red is a darker shade for a larger difference. If a Waveform channel has a process that divides the rate by n displays the division ratio after the sample rate as /n. A derived channel displays only the division ratio, so a derived channel with no divide down set shows /1 here.

The Scale, Offset and Units columns are used when scaling between user units and 16-bit integer Waveform values.

The Pnts and Pre columns are used by WaveMark channels.

You edit the channel settings by double-clicking on a channel, or by selecting a channel with the mouse or keyboard and clicking the Edit button. The Reset button deletes all editable channels and sets a standard sampling state. The buttons in the dialog are:

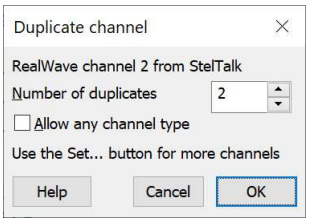
Edit...
This opens the appropriate dialog for the channel selected in the Channel list. You can also edit a channel by double-clicking a channel in the list.

New...
This opens the New channel dialog to add a channel sampled by a 1401. If you know the channel type, or you want to add a Talker channel or a Derived channel, click the down arrow to the right of the button and select the channel type. If you want to create several channels without opening a channel configuration dialog, hold down Ctrl when selecting the option.

Duplicate
The Duplicate button is enabled if it is possible to duplicate the selected channel. In all cases, there must be spare channels in the file. You can duplicate a 1401-based channel if there is a spare port for that channel type.

You can duplicate a Talker-based channel if another higher-numbered channel of the same type is available from the Talker. If you hold down the `Ctrl` key, the requirement that the channel is the same type is removed.

Duplicate n
The small n button next to the Duplicate button lets you duplicate a channel a set number of times or until the duplicate operation fails. If you need more channel space in the file you can increase the allowed channel count with the Set channels command.



If the current channel is from a Talker, the Allow any channel type check box appears. If you check this, the duplicate operation will start by looking for higher-numbered channels of any type that are free. Once these are exhausted, the search will start at the beginning of the list of talker channels. To see the full list of channels available in a Talker, create a single channel, open this dialog, set the number of duplicates to a large number, check the box and click OK.

Delete
Remove the selected channel from the list of channels. If there are derived channels that depend on this channel, they are also deleted. You cannot delete the Keyboard channel.

Copy As Text, Log
These two buttons render the sampling configuration as text in a format suitable for a laboratory notebook. Copy As Text places the text on the clipboard. Log copies the text to the Log window (so you can print it). The `SampleConfig$()` script command gets this text with additional formatting opportunities. From version [10.07], if you hold down the `Ctrl` key and click Copy As Text, the copied text uses `Tab` characters to separate the columns.

Set channels
The Set… button opens a dialog in which you can set the maximum number of channels that can be stored in a data file created by the File menu New command. You can also set where the special channels should be placed. The script equivalent is the `SampleChannels()` command.

Reset
The Reset button clears all the sampling information in the Sampling Configuration dialog to a default state. As this is destructive, you are asked if you really want to do it. Even if you do reset your configuration, changes can be forgotten with Cancel. Once you use OK or Run now the changes are confirmed. The Reset button does not clear information on duplicate windows and channels; see the Sample menu->Clear configuration command for this.

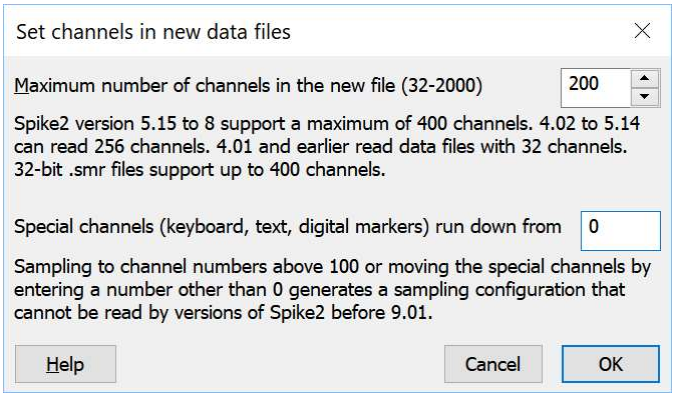
Special channels
Most channels have a free choice of channel number, but there are three that for reasons buried in history have forced numbers. These channels are the Keyboard Marker channel, the Text Marker channel and the Digital Marker channel. Originally, these were at fixed channels number of 31, 30 and 32, but from Spike2 [9.01] onwards, you can move these channels to a consecutive list of three channels in the sampling configuration by setting a special channel number by clicking the Set... button in this dialog. The script equivalent is the `SampleChannels()` command.

Special channel number	Keyboard channel	TextMark channel	Digital Marker channel

0 = Original scheme	31	30	32
special >= 32 [9.01]	*special*	*special*-1	*special*-2

Moving the special channels away from their original position makes the sampling configuration incompatible with versions of Spike2 before 9.01.

Set maximum channels



The Set... button in the Sampling configuration dialog opens the Set channels in new data files dialog. In this dialog you can set the number of available channels in the new data file and you can also choose if the Keyboard marker, TextMark and Digital Marker channels (the Special channels) are positioned at channels 31, 30 and 32 or are positioned elsewhere in the channel map. The script equivalent of this dialog is `SampleChannels()`.

Maximum number of channels in the new file

The minimum number of channels in a data file is 32 and the maximum is currently limited to 2000 (64-bit files have a theoretical limit of 65535 channels, but this is not yet supported by Spike2). If you reduce the number of data channels and there were channels defined in the Sampling Configuration dialog with numbers above the new limit, these higher numbered channels are deleted from the configuration.

Currently a CED1401 interface can sample a maximum of 100 channels (you will need a 1401 interface with expansion top boxes to use all of these channels). Before Spike2 9.01, all channels sampled by a 1401 or a Talker had to have channel numbers in the range 1 to 100. From 9.01 onwards you can use any channel number you wish (except a channel number reserved for a special channel).

Extra channels are often required in a data file for subsequent analysis operations, or to save additional channels generated during sampling.

Setting a larger number of channels increase the size of the file header, which holds the channel title and any units and sample rate information. It does not allocate data space in the file, so apart from an increase in the channel header size per channel and in memory usage per file, there is no other penalty. The 32-bit filing system does not allow you to change the number of channels once the file is created. The 64-bit filing system theoretically can add more channels after the file has been created and used, but there is no support for this in Spike2 yet. It is a good idea to allocate enough channels for your foreseeable data analysis needs. You can, of course, export a file to create a new file with the same data and more channels.

If you select the 32-bit .smr file format in the Resolution tab but select more channels than can be stored in a 32-bit file, Spike2 will open a 64-bit `.smrx` file.

If you will share your files with people using old versions of Spike2 you should be aware that before Spike2 version 4.02, only (32-bit) data files with 32 channels were readable. Version 5 reads 32-bit files with up to 256 channels (5.15 onwards can read 400 channels). Version 8 can only read 64-bit files with up to 400 data channels.

Special channels (keyboard, text, digital markers) run down from

Up to Spike2 version 9.01, the Keyboard marker, TextMark and Digital Marker channels have been treated specially, and have always been at fixed channel numbers 31, 30 and 32 in sampled data files. It can be

somewhat inconvenient to have these fixed channels numbers, especially in systems with many waveform channels where it would be convenient to have then in a continuous range.

You have the option to move these channels elsewhere in the channel map by setting this field to a value from 32 to the maximum number of channels in the file. If you set a value less greater than the maximum number of channels in the file it is limited to the maximum number of channel in the file. If you set a value less than 32, it is set to 0 (meaning use the old scheme).

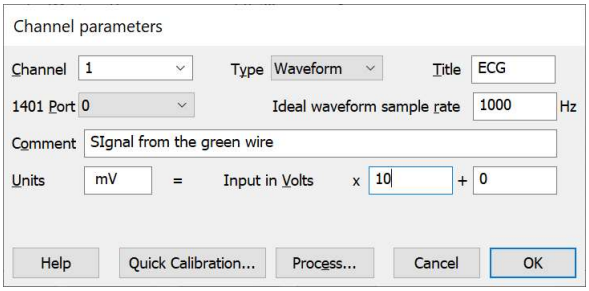
If you set a value of n, the keyboard channel is set to `n`, any TextMark channel is set to `n`-1 and any Digital Marker channel is set to `n`-2. If there were channels already set at these channel numbers, the channels are deleted.

Backwards compatibility

If you set a sampling configuration with a sampled channel at a channel number above 100 or with more than 400 channels in the file or with the special channels moved, this is not compatible with Spike2 version 8 or 7. Spike2 version 7.20 and 8.13 onwards will reject such configurations, but older releases will not and may fail if you attempt to load such a configuration.

Create a new channel

The New… button creates a new channel at the lowest free channel number and opens the Channel parameters dialog where you can edit the new channel to the desired state and set the channel number. To make several similar 1401-based channels, set up the first channel, then click Duplicate to insert a new channel at the next free channel and physical 1401 port with the same settings. To remove an unwanted channel, select it and click Delete.



You can also use the small button to the right of New... to select the type of the new channel from a drop-down list that also includes the names of any Talker devices that can be used as data sources and allows you to create Derived channels from 1401 data. Selecting one of these channel types opens a Channel parameters dialog with the channel type set.

You can also open the Channel parameters dialog from the Sampling Configuration by a click on the Edit… button or a double-click on a channel.

Channel parameters dialog

There are five fields that apply to all data types.

Channel — The channel number identifies the channel. Channels 30 to 32 are reserved for Text, Digital and keyboard markers (unless you have moved them). The remaining channels are for waveform, WaveMark and event data and data from Talkers. You can change a channel number, but not to a channel that is already in use.

Type — The type of the data to sample on the channel. You can set any of the following:

- **Off** — The channel is unused, equivalent to deleting the channel.
- **Waveform** — The channel holds waveform data, stored as 16-bit integers that are scaled to user units.
- **RealWave** — The channel holds waveform data, stored as 32-bit floating points values (useful with signal conditioners).
- **Event-** — Event channel timed on the falling edge of the data.
- **Event+** — Event channel timed on the rising edge of the data.
- **Level** — Event channel with the times of both data edges saved.
- **TextMark** — The channel holds (short) text messages and their times.
- **Marker** — The data is an event with either a keyboard character or a digital value attached
- **WaveMark** — The data is a small waveform fragment, usually a spike shape

If you change the type of a waveform or RealWave channel to a non-wave type, any dependent Derived channels are deleted.

Title — Text to identify the channel. This was limited to 9 characters before [10.07]. You can now use longer titles. However, if you sample to a 32-bit `.smr` file the title will be truncated. We suggest that you keep short titles, where possible; long titles can cause a cluttered screen. Title can include place holders (for example `%c` for the channel number) that are replaced when creating the data file, see below.

1401 port — For a waveform or WaveMark channel, this is the 1401 ADC input number. For an event channel, this is the digital input port number. Marker channels do not use this field. You will not be allowed to sample if you request a physical port number that does not exist in your 1401.

Comment — Text to describe the channel. There is a limit of 71 characters of comment to encourage you to keep it relatively short. The comment can include place holders (for example `%c` for the channel number) that are replaced when creating the data file, see below.

Title and Comment place holders (new at [10.09])

It is a common requirement to include channel and port numbers in title and comment strings. To make channel duplication simpler, we allow some channel properties to be included symbolically with `%x` markers (where `x` determines the replacement). The current set of replacements are:

`%c` The channel number.

`%d` The source device name. This is `1401` for a 1401 channels, the talker name for a Talker channel or `Chan n` for a channel derived from channel `n`.

`%p` The ADC or Event port number for a 1401 device, the serial port number (0 if unused) for a TextMark channel, the 0-based source channel index for a Talker device or the source channel for a Derived channel.

`%u` The channel units (before any changes made by processing).

`%%` Replaced by a single `%` character.

For example, if you want to generate 16 Waveform channels you could create one on ADC port 0 and give it a channel title `"ADC %p"`, the use the Duplicate n command to make 15 copies that will also get the same title. When used to generate a data file, the titles will be: `ADC 0`, `ADC 1`, ..., `ADC 15`.

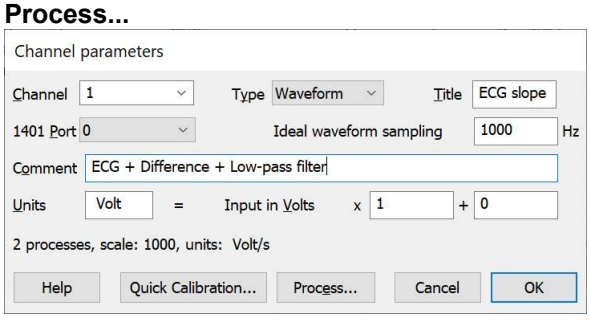
Quick Calibration...

The Quick Calibration... button is present for Waveform and WaveMark channels if there is no signal conditioner support installed for the channel and opens the Quick calibration dialog for the current 1401 port. You can find the documentation for using the Scale and Offset fields in the documentation for creating a new Waveform channel.

Conditioner...

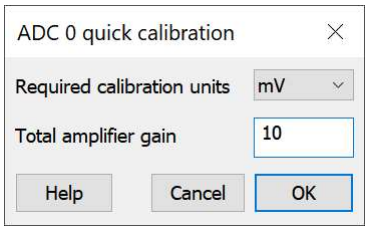
The Conditioner… button is present for Waveform, RealWave and WaveMark data types if a signal conditioner is connected for the 1401 port. It opens the Signal conditioner control panel.

Process...



The Process... button is present for Waveform (and RealWave from [10.08]) channels sampled from a 1401. It allows you to add real time data processing to the waveform channel that is applied before the data is written to disk. This is a new feature in Spike2 version [10.01]. If you apply any process, the channel is saved as RealWave data, even if sampled as Waveform. When you add processes, a text line appears in the dialog listing the number of processes and any changes these make to the scale, offset, units and sample rate of the channel.

Quick Calibration dialog



This dialog is available for Waveform and WaveMark channels that do not have signal conditioner support. It if for use in the common case where:

- The channel units are kV, Volts, millivolts, microvolts or nanovolts
- An input of 0 at the 1401 ADC input is to be recorded as 0
- You know the total gain between the measured signal and the 1401 ADC input
- The total amplifier gain is in the range 0.0005 to 10^9

If these conditions are satisfied, you can type in the gain, click OK and the correct gain and unit values for Spike2 to use will be set in the Channel parameters dialog. If the channel units are already set to a scaled Volt value the dialog will pick up the current settings. We accept most forms of the units (microvolt, microV, uV...) and ignore case. The channel units are set to one of: kV, V, mV, µV or nV.

There is no special script language support for this dialog. If recording is turned on, the effect of this is recorded when you open a new file for sampling. When you turn recording off you will see the following in the recorded file for a 1401 with a ±5 Volt input range:

```
SampleCalibrate(1,"mV",100, 0); 'scale and offset
```

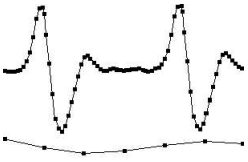
If the front end amplifier has a gain of 10, the full range of the input represents ±500 mV, rather than ±5 Volts (for unity gain), so we need a scale factor of 100.

Maximum sampling rates

The ultimate limit on the sampling rate is how fast data can be transferred from your 1401 to the host. If you use a USB 2 interface, this can manage rates between 10 and 40 MB/second (depending on the type of 1401), so this is not usually a limit. However, if you have an early Power1401 with a USB 1 interface or if you plug your USB 2 capable 1401 into a USB 1 port, you will be limited to around 1 MB/second. The PCI interface has a similar limit to USB 1. If your 1401 has a choice of interface, USB 2 is by far the quicker.

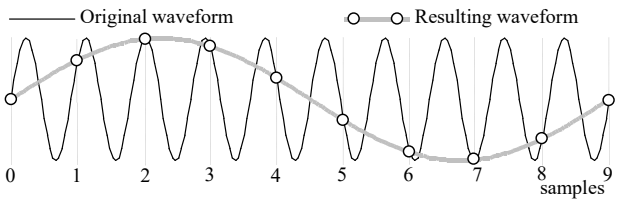
Another limit on throughput is the maximum ADC sampling rate that your 1401 can achieve. You should consult the Owners Guide that came with your 1401 to get the details. The sampling configuration allows you to specify the target 1401 type. You are limited to achievable rates for the selected target.

Waveform channels



The waveforms you record are continuously changing voltages. Spike2 stores waveforms as a list of 16-bit numbers in the range -32768 to 32767 that represent the waveform amplitude at equally spaced time intervals. The process of converting a waveform into a number at a particular time is called *sampling*. The time between two samples is the *sample interval* and the reciprocal of this is the *sample rate*, which is the number of samples per second. The dots in the diagram represent samples; the lines show the original waveform. The 1401 interface samples data as 16-bit values, so it is natural, and space efficient to store the data as 16-bit values in a Waveform channel. From Spike2 version [9.01] onwards you can also choose to sample data as 32-bit floating point values in a RealWave channel.

Minimum sample rate



The waveform sample rate must be high enough to represent the data. The sample rate must be at least double the highest frequency contained in the data. If you do not sample fast enough, high frequency signals are aliased to lower frequencies (as shown as the resulting waveform in the diagram). On the other hand, you want to sample at the lowest frequency possible, otherwise your disk will soon fill. Unlike many data capture systems, Spike2 lets you capture different waveform channels at different rates to minimise data file sizes.

Use of filters

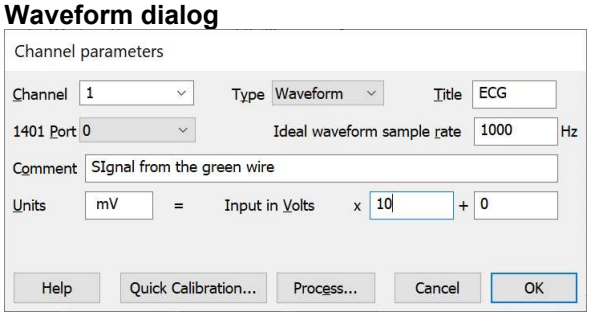
Many users pass waveform data through amplifiers or signal conditioners with filter options such as the CED 1902 to limit the frequency range. Some transducers have a limited frequency response and require no filtering.

Input connections

Channel	8	9	10	11	12	13	14	15	Gnd
Power (37 pin)	28	29	30	31	32	33	34	35	1-19
Power (44 pin)	33	34	35	36	37	38	39	40	3-10

Connect your waveform channels to the 1401 ADC inputs. Channels 0-7 (0-3 for a Micro1401) are the labelled BNC connectors. Channels 8-15 are on the Power1401 rear panel 37-way Cannon connector except for late model mk I and all Mk II, which have a 44-way high density connector and each signal has its own ground. Pin numbers are given in the table. If you have an ADC expansion fitted you will have more channels; see the accompanying documentation for the connections. Power1401 top boxes can reassign the rear panel channels. The standard input Voltage range is ±5 Volts. If you have a ±10 Volt system check that the Voltage range is set correctly in the Edit menu Preferences.

Waveform dialog



The Waveform channel dialog has the standard fields, plus:

Ideal rate Set the Ideal waveform sampling rate field to the desired sampling rate for this channel. The actual sample rate will be as close to this ideal rate as possible. The AdcRate column in the Sampling Configuration dialog displays the actual rate (in varying shades of red if it differs to much from the ideal rate). If the rates differ too much, adjust the optimisation and clock settings in the Resolution tab. If you sample with a rate that is outside the range 10% less than ideal to 30% more, a warning is added to the Sampling Notes.

This determines the rate that the 1401 uses to capture data. If you add a real time process to the channel that down-samples the data, this will change the rate that is written to the disk file and the rate that is visible within Spike2. For example, you might capture a channel at 1 KHz, then use real time processes to low-pass filter it to 40 Hz and down sample it to 100 Hz.

Units This field holds the waveform units. Units are limited to 10 characters in 64-bit smrx files and to 5 in .smr files. Shorter strings are preferred. The following fields are the *scale* and *offset* to convert the 16-bit input into user units. If your target units are some variety of Volts (for example mV or uV) you may be able to use the Quick Calibration option. Note that if a channel has Processes, from version [10.09] if the process causes a change to the scaling, offset or units, these changes are applied to the values set here. Prior to this, no changes were made to allow fro processes.

input in user units = (16-bit input value/6553.6) * *scale* + *offset*

This is arranged so that with a ±5 Volt input range, the *scale* is the number of user units for every one Volt increase in input; *offset* is the value represented by 0 Volts at the input. In this case:

input in user units = Volts at the input * *scale* + *offset*

However, some systems have ±10 Volt inputs. In this case:

input in user units = 0.5 * (Volts at the input) * *scale* + *offset*

When used with a RealWave channel, the *scale* and *offset* are used to convert the floating point values into 16-bit values for situations where the channel is required in integer form.

Scale — See the Units description.

Offset — See the Units description. If your offset is 0, see the Quick Calibration option.

As an example, consider a situation where a waveform represents a position. 1 Volt is equivalent to 10 mm and 3 Volts is equivalent to 50 mm and the 1401 has ±5 Volt input range. In this case you would set:

scale = (50 - 10) / (3 - 1) = 20.0 mm V^{-1}
offset = 10 - (1 Volt) * *scale* = -10.0 mm
Units = `mm`

To display in metres in place of mm, set *scale* to `0.02`, *offset* to `-0.01` and *units* to `m`. You can calibration a channel after sampling from known input data with the Analysis menu Calibrate command.

A common requirement is to display the sampled data in millivolts. In this case, just set the Units to mV, the scale to 1000 and the offset to 0.

For the scaling to work as expected, the Edit menu Preferences option for Voltage range (5 Volt range or 10 Volt range) must be set correctly for your 1401; this should happen automatically for a modern 1401 where we can read back the voltage range. Since version 6.12, we store the voltage range in the sampling configuration and adjust matters as appropriate. If you open an old sampling configuration, we will assume that the voltage range is whatever is currently set.

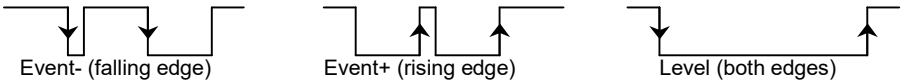
Note that the scale value associated with a channel of sampled data and accessed by the `ChanScale()` script command is the same as the scale value defined here for 5 Volt systems, but will be double the value here for 10 Volt systems. The scale and offset values defined here are also set by the `ChanCalibrate()` script command.

Process...

The Process... button is present for Waveform channels sampled from a 1401. It allows you to add real time data processing to the waveform channel that is applied before the data is written to disk. This is a new feature in Spike2 version [10.01].

Event data

Spike2 stores time stamps efficiently as integer multiples of the time resolution set in the Resolution tab of the sampling configuration. Simple time stamps with no other attached data are called *events*. Each event uses 8 bytes of storage. The 1401 recognises events as changes of state of TTL compatible signals connected to the 1401 digital input bits 15-8. We refer to these inputs as *event ports* 7-0. There are three types of event:

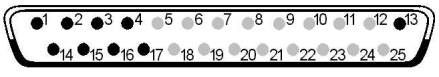


Event- — Spike2 saves the time of the falling edge of the input signal. The minimum input pulse width is 1 µs; wider is better.

Event+ — The same as Event-, but Spike2 saves the time of the rising edge.

Level — Spike2 saves the time of both edges. Pulses should be a minimum of 20 µs wide or the time resolution set for sampling, whichever is the larger. Do not use this type unless you need the times of both edges. From Spike2 version 8.00, Level events are actually a form of Marker data and the state of the event (low going or high-going) is stored in the marker code.

Micro1401 and Power1401 event port connections



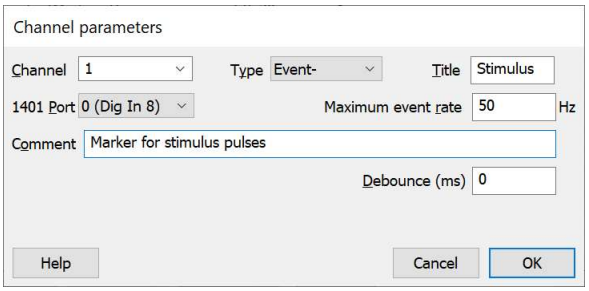
Event ports 0 and 1 are BNC sockets on the front panel. If you have the optional Spike2 top box, event ports 7-2 are also on BNC sockets, otherwise you must use digital input bits 15-10. The digital input connector is on the rear panel. If you want to use the rear panel digital input connector for event ports 0 and 1 there is an option in the Edit menu Preferences… to use the rear panel connector for all events.

Input connections (digital input)

Digital input bit	15	14	13	12	11	10	9	8	Gnd
Digital input pin	1	14	2	15	3	16	4	17	13
Event port	7	6	5	4	3	2	1	0	

The pin connections for the digital input connector are the same for all 1401s:

Event dialog



The event channel dialog is similar to the waveform dialog. There is no Units field and there are new fields for the Debounce period and the Maximum event rate. If two events are closer together than the debounce period, the second event is ignored. You can also set a negative value (-1 recommended) to convert simultaneous events into events at consecutive clock ticks. Set this field to 0 to disable this feature; setting 0 saves processing time.

The Maximum event rate should be set to your estimate of the maximum mean event rate sustained over a few seconds. When used with Level data, the rate is the rate of edges. This is not the maximum instantaneous rate, which may be much higher. Spike2 uses this information to optimise buffer space allocation. The buffer allocation also determines how far backwards in time triggered sampling can extend so setting a realistic value can be important.

As an example, an event channel might have a mean rate of 30 events per second, but it could have an instantaneous maximum rate of 1 kHz if two events fell within 1 millisecond of each other. In this case, the rate you should enter is 30 Hz, not 1000 Hz.

Debounce

This field appears for Event-, Event+ and Digital Marker channels. It sets the minimum acceptable interval between consecutive events in milliseconds. Events closer that this to the previous event are not saved to disk. This is typically used when events are logged from a mechanical switch. Mechanical switches commonly have bouncy contacts, such that when you activate the switch you can get multiple switch actions in a short period of time. You can use this setting to keep the first of a series of pulses and reject all the extra ones within the debounce period of the first. There is a time penalty for setting this value non-zero, so you should only use this option if you need it.

In terms of system performance it is always better to fix the bouncy signal (for example by using a better switch or an electronic switch debouncer) as all the extra events use up 1401 resources to capture them and transfer them to the host computer where additional resources are used up in eliminating them.

There is a Channel Process option to debounce event channels off-line.

Level event channels

There is no on-line debounce option for Level event channels. If you sample to a 32-bit smr file we save all the data, as sampled; duplicated event times may cause some problems with subsequent analysis. You can use SonFix to repair such files.

The 64-bit smrx file system implements a data buffering scheme that allows us to delete pairs of duplicated times. This means that any even number of level events at the same time are deleted and an odd number are recorded a single event. There is still the possibility of a problem with this scheme if a commit happens that writes a level event, then the next event is at the same time. As a future extension, it may be possible to keep pairs of level events at the same time by setting the second event one tick later; this would at least preserve the event if not the timing of it. This would probably need to be an option; most multiple events are not intentional.

There is a Channel Process option to debounce Level event channels off-line.

Keep all events

You can also set the debounce period to -1 to ask the system to attempt to keep simultaneous events by giving simultaneous events consecutive clock ticks.

When to use this option

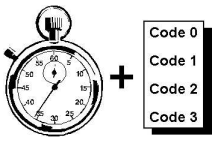
If you get a message from sampling stating that there were multiple events at the same time and suggesting that you use this option and it is not practical to fix the event source, then enable this option. This warning only occurs if the repeated events are so quick that two or more get the same time stamp. It is more common that you have events that look correct, but the number of events counted between cursors is more than you expect. Zooming in on the data will reveal that what looked like single events were, in fact, multiple ones. A simple way to find these is to display the event channel as an instantaneous frequency.

TTL compatible signals

TTL stands for Transistor-Transistor Logic, a method for passing logical information between devices using voltage levels. Levels above 3.0 Volts are in the High state, levels below 0.8 Volts are in the Low state. Levels in between 0.8 and 3.0 Volts are undefined.

Do not subject 1401 TTL inputs to voltages above 5.0 Volts or less than 0.0 Volts. CED hardware has special circuits on TTL compatible inputs to provide some protection, however determined abuse will damage them. The 1401 TTL compatible inputs are pulled up by a resistor to 5 Volts. They require a current of no more than 0.8 mA to pull them into the Low TTL state. Alternatively, you can connect them to ground to pull them low (this can be useful for the Trigger input). See the *Owners handbook* of your interface for full details of all input ports.

Marker data



Spike2 samples keyboard markers and digital markers. The keyboard marker channel is always sampled, the digital marker channel is optional. A Marker is a 64 bit time plus 4 bytes of marker information (codes). The first of these 4 bytes is the ASCII code of the keyboard character pressed by the user or an 8 bit digital code read by the 1401. The remaining three bytes are set to zero. When marker data comes from other sources (such as Talkers or the script), all 4 codes may be used. There is also an additional 4 bytes that are reserved for future use, either as 4 more marker codes, or as an addition 32-bit value.

Spike2 treats all marker types identically once the data has been captured; they differ only in their source. You can also treat data types that are derived from markers (such as WaveMark, TextMark and RealMark) as if they were markers.

The keyboard and digital markers are sampled on particular channels numbers, see Special channels for details.

Keyboard markers

The timing of Keyboard markers is not precise; it depends on the load on the computer. Use the event inputs for exact timing. Any keyboard character that is not trapped for a special purpose (for example `Ctrl+L` opens the Evaluate window) is recorded, but *only when the sampling document window is the current window*. In the special case where keyboard markers trigger writing data to disk, the exact time at which the trigger took effect is stored. The keyboard marker channel is always enabled.

From [10.10], the Keyboard marker channel can be the target of an online Measurement to data channel process.

Keyboard markers can cause the output sequencer to jump to a particular instruction and the arbitrary waveform output to play a particular waveform. In both cases this happens when the key pressed matches a key set in the sequencer or waveform output.

Keyboard markers are not only used to record key presses. During data sampling, keyboard markers with specific codes are added on the following events:

Timed mode sampling start and stop
Each time sampling starts due to timed sampling mode code 00 is added. Each time sampling stops, code 01 is added.

Channel gain change and no TextMark channel
If a signal conditioner changes the state of a channel, the system attempts to add code 02 to the TextMark channel. If there is no TextMark channel, it adds code 02 to the Keyboard marker channel.

Special keyboard codes
Codes 00, 01 and 02 are reserved and mark sampling changes. Do NOT use them, or codes above 0x7F, to link the keyboard markers as they will not trigger other actions (PlayWave, Output sequencer).

Talker drift diagnostics
From Spike2 version [10.16], when a Talker is asked to provide drift information it adds diagnostic codes `Dx` (`x` is a hexadecimal digit) to

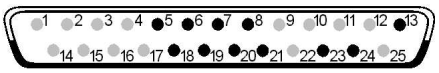
Data overrun
If a data overrun occurs that causes data to be lost, but that does not cause the integrity of the data sampling to be compromised, special keyboard markers with a first marker code of 0xFF are written to the keyboard marker channel. From Spike2 8.04a onwards, the second marker code is a sum of codes for the source that caused the problem. The source codes are:

2 Event channel overflow. There are so many changes happening on the event inputs that we have had to ignore some of them. This is usually caused by noisy input signals. Setting a debounce period will not fix this as debounce only applies after the data has been captured and stops excessive events from being added to the data file. You should fix this problem as the work being done in the 1401 to deal with these inputs may block other 1401 activities.

4 There are more WaveMark channel spikes being captured that can be transferred and we have had to ignore some. This has never been reported in normal use. It is likely that sampling would be stopped for other reasons before this error could be generated.

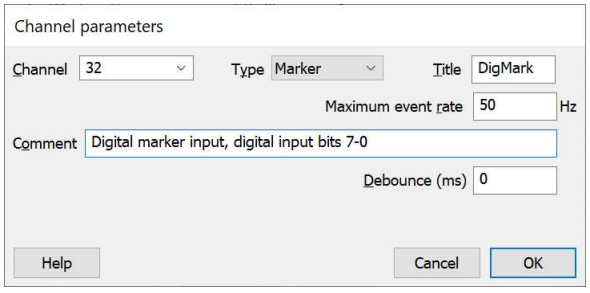
64 The output sequencer was unable to keep up with the sequencer interval you set. In many cases this will not be a problem, but if you were relying on the sequencer timing being exact you may need to check that this did not occur in a timing critical part of the sequencer code.

Digital markers



The digital markers are timed as accurately as the event data. They record 8 separate channels of on/off information, or one channel of 8 bit numbers, or any combination in between. Digital marker data is sampled when a low going TTL compatible pulse is detected as described below. The data is read from bits 7-0 of the 1401 digital input. Recording of the digital marker can also be triggered from the output sequencer.

Digital marker dialog



The Maximum event rate field is used to allocate the system resources for data capture on this channel. Set a reasonable estimate of the maximum sustained data rate over several seconds. Do not set the peak rate or you will waste resources.

The Debounce (ms) field sets the minimum period between digital marker events with the same marker code that you expect and is used to reject spurious markers caused by bouncy mechanical switches. Markers with different codes can be spaced closer than this time as long as the time difference is not 0. You can also set a negative value (-1 recommended) to assign markers at the same time to consecutive clock ticks. Set this to 0 to disable this feature (which saves time on every Marker).

Digital marker connections

Marker data bit	7	6	5	4	3	2	1	0	Gnd

Digital input pin	5	18	6	19	7	20	8	21	13

The digital marker is read from data bits 7-0 of the digital input connector. The Micro1401 and Power1401 require a pulse on digital input pin 23 to flag a digital marker. In addition to the data lines, there is an optional handshake (h/s) signal.

Signal	Micro & Power
Event flag	pin 23
h/s	pin 24
h/s active	TTL low

To flag an event, apply a low going TTL pulse at least 1 µs wide to the Event flag input. When the 1401 detects a falling edge at the Event flag input, it sets the h/s line active (within a few microseconds). The falling edge of the Event flag input latches the input data in the Micro1401 and the Power1401. The h/s returns to a non-active state after the 1401 reads the input (meaning external equipment can send another value without danger of it being lost). There is no requirement to make use of the h/s signal.

Digital marker output sequencer link

There are links between the digital marker channel and the output sequencer. If the `REPORT` instruction is used in an output sequence, this simulates a digital marker input pulse and causes the digital input to be read and the time to be recorded. As this is an internal activity, the handshaking described here is not available.

You can also use the `MARK` output sequence instruction to record a digital marker without reading the digital inputs (the instruction sets the 8-bit marker code). This instruction is often used to record output sequencer actions as part of the data file.

You can mix externally and internally generated digital markers, but this is not recommended unless care is taken to differentiate between the two sources of markers during analysis. This could be done by connecting the external marker handshake line to one of the digital marker data bits so that all external markers were flagged.

Warning: The `DIBEQ`, `DIBNE`, `DIGIN` and `WAIT` sequencer instructions use the same inputs as the digital marker and can cause digital marker events to be missed with old 1401 hardware and firmware. This is only an issue with Power1 and Micro2 units with very old FPGA images. Spike2 will warn you if this is an issue with your hardware and you can download up-to-date firmware from our web site to fix this.

Marker codes

When Spike2 displays markers, or data derived from markers, such as WaveMark or TextMark data, by default it shows the code of the first of the four markers. Marker codes occur in several other guises, for example to set trigger codes, in the TextMark dialogs, as arbitrary waveform output codes and in the spike shape module.

Marker codes have values from 0 to 255. The first half of this range (0-127) is the same range of numbers that the ASCII character set uses, and it is often convenient to treat the codes as ASCII character codes (for instance when dealing with keyboard markers). At other times it is more convenient to deal with the codes as numbers.

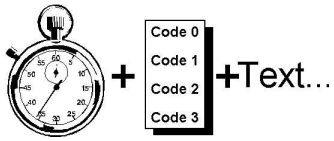
Whenever Spike2 displays a marker code that is the same as the ASCII code of a printing character, it shows the printing character, otherwise it displays the character as a two digit hexadecimal code. Hexadecimal (base 16) numbers use the standard digits `0` to `9`, but also use `a` to `f` or `A` to `F` (for decimal 10 to 15). Thus `00` to `09` hexadecimal is equivalent to 0 to 9 decimal. `0a` to `0f` is equivalent to 10 to 15 decimal. `10` to `1f` hexadecimal is 16 to 31 decimal, `20` to `2f` is 32 to 47 decimal and so on. This behaviour can be changed so that only hexadecimal codes are displayed on a channel by channel basis using the Draw Mode dialog or with the `MarkShow()` script command.

+	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

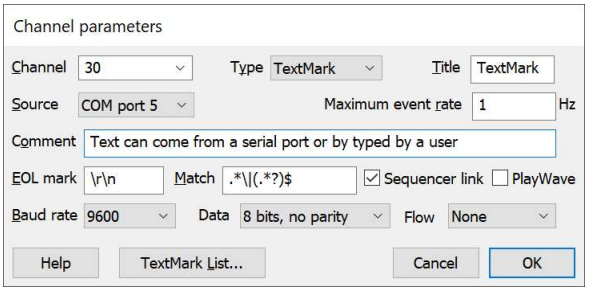
The printing characters are `20` to `7e` hexadecimal, 32 to 126 decimal, as in the table. Character `7f` may or may not print, depending on the character set. To find the hexadecimal code of a printing character, add the number above it to the number to the left of it. For example, the hexadecimal code for A is 41. To convert a code to a character, look up the first digit in the left column and the second in the top row. For example, `3f` codes to `?`, the intersection of the row for `30` and the column for `f`.

When typing marker codes (for example in the Gate Settings dialog, or when assigning codes in the spike shape module), type two hexadecimal digits for a code or type a single character to stand for itself. Most places that accept marker codes will not let you type in an illegal combination.

TextMark data



This type is a combination of a marker and a text string. It is stored as a time and 4 bytes of marker information followed by a text string that can be up to 80 characters long. This type allows you to insert timed comments into a data file. In the special case where the TextMark channel triggers data sampling, the precise trigger time is stored. From a script you can set longer or shorter strings when you create the sampling configuration with the `SampleTextMark()` command. Most files have at most one TextMark channel, but you can create more with memory channels or by importing data with multiple TextMark channels. The Sampling Configuration dialog allows you to create one TextMark channel.



This is one of the Special channels and the channel number is fixed to either 30 (for compatibility with old sampling configurations), or is the special channel number -1. The Special channel number is set by the Set channels in new data file dialog, or by the `SampleChannels()` script command. If you select the channel number reserved for the TextMark channel, the data type is forced to TextMark. The Maximum event rate sets the expected maximum TextMark rate over several seconds and is used to allocate sampling buffers.

COM port (serial line) input

The Source field can be set to Manual (data is added by user interaction) or a choice of COM ports. COM ports, also called by names such as Serial line, RS-232 or RS-423 ports or UART ports, are a relatively slow communication protocol that has largely been overtaken by USB. However, many devices still use these ports, or simulate them with USB. RS-232 ports have been deprecated in modern computers since the 1990s, but serial line devices and USB devices that emulate serial interfaces are commonly in use. If you have a device that generates serial line output it is likely that you will be using a USB to Serial converter to generate the external serial port, so consult the documentation that came with the converter for pin connections.

From [10.09] we identify ports that currently exist on your computer as COM port n (from [11.00] the drop down list includes the port *friendly name*), and ports that you can select, but that do not currently exist on this machine as (COM n) with n in the range 1 to 19.

Spike2 is set to expect to read 'lines' of ASCII text that may contain one or more marker codes. The time associated with each line of text is the time at which the first character of the text was collected by Spike2. The end of each line is marked by a terminator, which you can define.

If you select a serial port more fields can be set.

Line termination

The EOL mark field (End Of Line mark) sets the input that terminates a text line. Spike2 needs to know the final character(s) of a message from a device, otherwise it cannot tell when the device message is complete. This field allows you to specify the character or characters that terminate a line. These characters can be typed as text. You can also include `\r` for carriage return (ASCII code 13), `\n` for line feed (ASCII code 10) and `\t` for Tab (ASCII code 9). If the field is left empty, the system uses carriage return (`\r` or ASCII code 13) as the line terminator. You can also use `\xNN`, where `NN` stands for two hexadecimal digits. For example if all lines ended with the letter `E`, `N` and `D` followed by the ASCII code 0 you could set the terminator to:

```
END\x00
```

You are allowed up to enter up to 7 characters to define the terminator. Note that the line end characters are not included in the text that is saved. All received characters are included in the captured text except character code 0 (which terminates text strings).

We expect that the received text is either ASCII characters, or UTF-8 characters. If you need to receive binary data from a serial port you can do this from a script using the serial line support routines.

Before Spike2 [10.18], you could supply one termination character only and characters codes less than ASCII 32 were ignored unless they were the terminator. The field was called Terminate and used the same coding as for marker codes, so a single character stood for itself; two characters were interpreted as a hexadecimal code. Common codes were `0D` for carriage return (the default) and `0A` for line feed; `00` could not be used. If you did not set a terminator, or set 00, `0D` was used.

Match the text to use for the code

The serial data can set the marker code associated with each data item. Prior to version [11.00] this was done by ending the text with a vertical bar followed by a decimal or hexadecimal number (starting with `0x`) that encoded the marker codes `c0`, `c1`, `c2` and `c3` as: `c0 + 256*(c1+256*(c2+256*c3)`. Each code is in the range 0 to 255.

The Match field allows you to override this method, which can be useful if you do not have control over the format of the serial line data. If the field is empty, the old method is used. See here for details and examples. The script equivalent is `SampleTextMarkLink()`.

Serial line settings

You can also set the standard serial line parameters for Baud rate, data bits and parity and handshaking (Flow). These must match the data source for reliable operation (so you need to read the manual that came with your data source device). You can set the flow control to None (works in the majority of cases), Hardware (needs additional RS-232 pins to be connected) and XON/XOFF (uses transmitted characters for flow control). Set None unless your data source requires you to use one of the other protocols. If you have a choice of Baud rate, choose the highest one available on your device. If your device does not talk about data bits and parity, choose 8 bits, no parity.

It may be that your device has a USB interface and that when you plug it in, it appears on your computer as a COM port. In this case, it is quite possible that the it does not matter what serial line parameters you select as there is no serial line involved.

When reading the serial line, characters codes below 32 are ignored unless they match the terminator. The marker time is set when the first character arrives except for the special case where the TextMark channel triggers data sampling, when the precise trigger time is stored.

The script language can also access the same COM port as is being used for TextMark input. The script language will usually use this ability to write text to the port; this will often be used to configure or to trigger the device attached to the port. The port is opened for TextMark input when the sampling file is created and is closed when sampling stops. However, if it has been opened in the interim by the script language, it will remain open for script use until closed by the script. Both the script and TextMark input can open the COM port without blocking each other. Beware that on open, they both configure the port and flush the buffers.

Time of serial line input

The time attached to the input is the latest data acquisition time known when the first character of the message was read. Prior to Spike2 version 9.04, it was the time for which all data had been received, which was typically a few (2-5) milliseconds in the past. Data transmitted via a serial line will always have some latency. At 9600 Baud it takes around 1 millisecond to transmit a character, at 115200 Baud it takes around 0.08 ms. Even if no serial line is involved (your device is purely USB and emulates a COM port), there is still time latency through the USB system of order 0.2 ms in USB 2.0. If you need more precise timings, use an event channel.

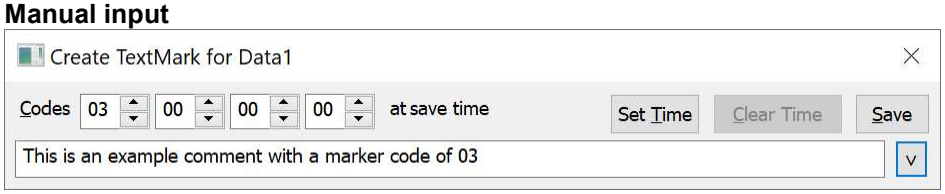
Sequencer and PlayWave link

These check boxes allow codes received from the serial line to trigger output sequencer jumps and arbitrary waveform (PlayWave) output) normally done by the Keyboard Marker channel. The script equivalent is `SampleTextMarkLink()`.

TextMark List

Click this button to open the Set TextMark list dialog in which you can edit the pre-set list of TextMark items that can be applied during sampling. This dialog also has separate controls to link manually applied TextMarks to the output sequencer and arbitrary waveform output.

Manual input



TextMark data is added to your file during sampling from a serial line and with the Sample menu Create a TextMark... command. You can also activate the dialog from the system toolbar and with the `Ctrl+T` key combination as long as the sampling data file is the active window.

TextMark data is drawn as small rectangles. The rectangles are yellow unless the first marker code is non-zero, in which case the same colour coding as for WaveMark data is used. Move the mouse pointer over a marker to see the attached text. Double click to view and edit the text and codes and display a list of the markers in the file.

If you enable this channel, Spike2 logs any programmable signal conditioner changes as TextMark items.

Link to signal conditioner changes

If the TextMark channel is enabled during data capture with a programmable signal conditioner and a signal conditioner change occurs, a TextMark with an explanatory message and code 02 is added to the channel.

High TextMark sample rates

TextMark data was not (originally) designed with high sample rates in mind. However, some users have taken to using this channel type to store large quantities of data at relatively high sample rates. Some steps were taken at version 7.11c to make this work better, and more changes were made in version 8. If you have large numbers of TextMark data items in a file we limit the number of displayed items in an attempt to maintain performance. If you are using an old `.smr` file with huge numbers of TextMark items, this dialog may be noticeably slow to open.

Extracting a code

The Match field holds optional text that can locate the marker code in an input serial text line and that can determine how the located text is converted into marker codes. The format of the line is (items within curly braces are optional):

```
{X^}{regular expression}
```

The first optional part is a single character followed by ^. This controls how to decode the located text into a 32-bit number that encodes the marker codes `c0`, `c1`, `c2` and `c3` as: `c0 + 256*(c1+256*(c2+256*c3))`. Each code is in the range 0 to 255. Currently, the recognised options are:

`c^` The code is expected to be either a single character, and the number is set to the ASCII code of the character, or 2 or more hexadecimal digits that generate a 32-bit number that is used as the 4 codes. This matches the default setting for displaying Marker codes in Spike2.

`h^` The code is expected to be hexadecimal digits (no leading `0x`) that are decoded into a 32-bit number. "27" is interpreted as decimal 39.

`n^` The code is a decimal number, or if it starts `0x` or `0X` it is a hexadecimal number, that is decoded into a 32-bit number.

We use ^ in the second character position to indicate that the first character is a format code because in a regular expression, ^ means the start of a line, so would not occur after the first character unless the input text had multiple lines, which is unlikely in this usage. You can also use the upper case versions of these letters (`C^`, `H^`, `N^`). This causes the entire matched regular expression to be removed from the input string before it is set in the TextMark. If any other letter than `c`, `h` or `n` (or `C`, `H` or `N`) is used, `n^` is assumed. If you have decoding requirements not covered by these options, contact CED and we will see what can be done.

The second optional part is an ECMAScript regular expression that isolates the code in a capture group. If there are multiple capture groups (as can happen with alternatives), the last, non-empty capture group is used. If (unusually) there are no capture groups, the text matched by the entire regular expression is used to locate the code.

Extracting a code when no regular expression provided

If the {regular expression} part of the Match field is empty, the code is located as the text following the last vertical bar in the input line (this was the method used prior to version [11.00]. Originally, the text was expected to be either a decimal number, or a hexadecimal number introduced by 0x or 0X. However, now the number format can be set by the first two characters of the Match field. Here are some acceptable inputs with n^ format set, or a completely empty Match field. In the examples, <term> stands for the line termination:

```
This text has no code, so will be coded 00<term>
This input has decimal code 49, which is ASCII A|49<term>
This sets codes 0-3 as hexadecimal 01, 23, 45, 67|0x67452301<term>
```

If you use this method, the text from the vertical bar onwards is not included in the text stored in the TextMark.

Examples of regular expressions

The following are simple-minded examples of how to locate marker codes. Much fancier expressions are possible, for example, if you know that the code is a decimal number you could replace the .*? used to match anything with \d*? to match only decimal digits.

Match code after a vertical bar

This is equivalent to the original method, and could be used to locate the text after any single character used to mark the code. There are many ways to express this search, but the simplest is:

n^\|(.*?)$

The n^ at the start indicates that the matched group is to be interpreted as a number. The \| means: find a vertical bar. We have escaped (placed a backslash before) the vertical bar because it is a special character that means alternation and adding \ makes it a normal character. The pair of brackets means capture the text within the brackets to be interpreted as a code and $ means the end of the text.

A dot means match any character. The * means match the previous item as many times as possible, but following this with ? means a non-greedy match, which matches the shortest possible sequence consistent with what went before (the vertical bar) and what follows (end of text). An input line this would work with might be:

```
Treatment type 3 started|3<term>
```

This would store a TextMark data item holding "Treatment type 3 started|3". To eliminate the |3 from the end of the string you could use:

N^\|(.*?)$

This flags we want numeric interpretation of the code and we should delete the entire matched text (not just the text captured for the code). The TextMark data item would get the text "Treatment type 3 started".

If you wanted to find the text after the last colon on the line you could use:

N^:(.*?)$

Colon is not a special character, so does not need escaping.

Match code surrounded by markers

If we had input lines of the type:

```
Treatment type <3> started<term>
```

We could match this with:

<(.*?)>

We have omitted any control of how the captured text is interpreted, so it will default to n^. In this case, the captured text would be "Treatment type<3> started".

If more that one type of enclosing marker is possible, for example <3> or [3] we could use:

<(*.?)>|[(.*?]

This generates two alternate capture groups, one of which will be empty.

Match code at a particular position in a line

Some equipment generates output in a fixed format:

```
Temperature:  23.6 State: 04 Time: 12:23:06<term>
```

If we wanted to convert the state to a code we could do it with:

```
State: (.*?) Time:
```

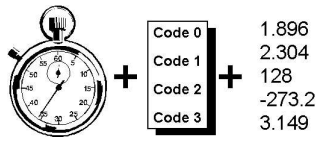
but if the surrounding text is not fixed, but the field widths are fixed, we can match based on position. In this case we want the 27th and 28th characters on the line. We can express this as:

```
^.{26}(..)
```

This translates as: beginning of text, any character 26 times, then capture the next two characters.

RealMark data

RealMark stores a 64-bit time, 4 bytes of marker information, then a user-defined number of single precision floating point numbers. You can create and manipulate this data type via a Talker, from the script language or by using active cursors to process measurements to a channel or through memory channels.



If you import waveform data into a RealMark memory channel, the first of the floating point numbers is set to the peak, trough or level value that was used to detect the event.

Titles and Units

From Spike2 version [9.02] onward, you can assign separate titles and units to each attached data value for use when the channel is displayed as a waveform from the Channel Information dialog or with the `ChanUnits$()` and `ChanTitle$()` script commands. You can select which index is displayed from the channel draw mode dialog or with the `ChanIndex()` script command. The format of multiple titles and units is described in the `ChanUnits$()` and `ChanTitle$()` script command documentation. Talkers and File import filters can use this to generate data with titles and units set for each item.

From version [10.20], Talkers that generate RealMark data have an extra 64 characters of space to set separate titles and units for each item. This is usually sufficient for RealMark data with up to 10 items.

Duplicate channel

From Spike2 version [10.18] you can duplicate a RealMark channel to generate one channel per attached item and display it as a waveform.

RealWave data

This is identical to waveform data except that the data is stored as 32-bit IEEE floating-point data, not as 16-bit integers. The channel has a scale and offset. These are used to convert between waveform data and RealWave data:

RealWave data = integer data * scale / 6553.6 + offset

From Spike2 version [9.01] onwards you can sample RealWave data and from [10.01] Derived channels are stored as RealWave data. You can also create RealWave data as a memory or virtual channel and with scripts. RealWave data is used when importing data from sources that provide floating point data channels or channels with more than 16-bits of integer resolution. Spike2 versions before [4.03] cannot open data files holding this data type.

Sampling 1401 data as RealWave uses twice the disk space of waveform data. However, if you are using a programmable signal conditioner, using a RealWave channel allows you to change the conditioner gain and offset without affecting all the previously sampled data (as is the case with a waveform channel).

Using RealWave data as integers

There are several places in Spike2 where you can specify RealWave data for use where integer values are required. For example, if you specify a RealWave channel as a source for data written to a DAC for playing offline or for an arbitrary waveform. In these cases, the channel scale and offset are used to convert the data from user units into integers:

integer data = (RealWave data - offset)*6553.6/scale

If the result would exceed the range -32768 to 32767 it is limited to these values.

Not a Number and Infinity values

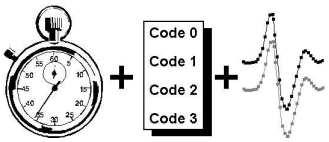
When RealWave data is imported or sampled from a Talker it is possible to get values that are not representable on the y axis. For example, dividing a value by zero can generate positive or negative infinity. Some Talker devices can use special *Not a Number* (NaN) values to denote corrupted or missing data sampled over a radio or other link.

Spike2 copes with infinities by drawing them as very big positive or negative values. Spike2 copes with NaN values by treating them as 0.0 for drawing purposes. You can make Spike2 skip NaN values (treating them as gaps in the data) with the Skip NaN channel process. You can fill gaps automatically with the Fill Gaps channel process. Short gaps can be filled by Linear Prediction. It is possible to replace NaN values with estimated values from a script using the `ChanWriteWave()` command, but the only way to find then is to convert them to gaps and then use the fact that `ChanData()` reads end at a gap.

Beware that NaN values are contagious. A single NaN value makes the result of all calculations it takes part in a NaN, so operations such as filtering and cubic splining that generate results from a range of data points will spread a single NaN value.

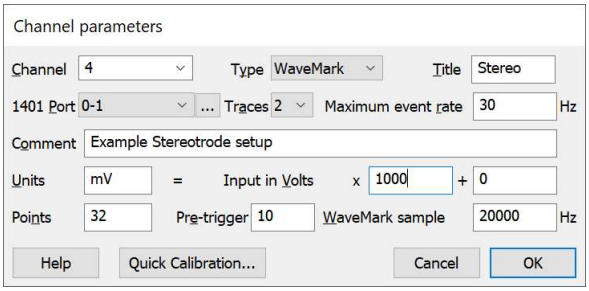
WaveMark data

This type combines waveform and marker data. It is stored as a 64-bit time and 4 marker bytes, followed by up to 126 waveform points on 1, 2 or 4 traces. The traces hold a spike shape stored as 16-bit integers. The first marker byte holds the spike classification code or 0 if it is unclassified. Script users can create WaveMark data for use off-line with up to 1000 data points.



Use WaveMark data where a high waveform sampling rate is needed to characterise very short events, for example nerve spikes. When the incoming waveform crosses a trigger level, the signal is tracked to the next peak (or trough), and data around the peak is saved.

WaveMark dialog



The Maximum event rate is the expected maximum spike rate averaged over several seconds. Spike2 uses this value to calculate how best to share out the buffering space used during data capture; an exact figure is not required.

The WaveMark sample rate field sets the desired waveform rate for all WaveMark channels. Spike2 will sample at a rate as close to this as it can achieve given the constraints of the sample rates set for the other sampled Waveform, RealWave and WaveMark channels.

The Units field is the same as for a waveform channel. You can use the Analysis menu Calibrate command to calibrate known values. There are fields to set the number of data and pre-trigger points per event; these can be adjusted during template formation.

Points — The waveform points per trace to store for each WaveMark on this channel in the range 6 to 126. You should set this to the smallest value you can (the larger the value, the more space is used on disk, and the slower it is to process). The lower limit is set by the requirements of template matching, the upper by the requirement to write efficient code within the 1401.

Traces — A Micro1401 mk II or -3 or a Power1401 can sample multiple traces for stereotrode and tetrode data. By default, traces use consecutive ports; the 1401 Port field sets the port for the first trace.

For example, if the 1401 Port field is set to 2 and you have 4 traces, data will be sampled on 1401 ports (ADC inputs) 2, 3, 4 and 5.

 However, with multiple traces you can click the ... button to open a dialog where you can set non-sequential ports. The first port set for each WaveMark channel must be different, the following ports are a free choice.

Pre-trigger — The number of data points to keep before the first peak or trough to exceed the trigger level for this channel.

WaveMark sample rate — This field sets the ideal sample rate for all WaveMark channels. Spike2 will adjust the sampling parameters to get as close to this rate as it can. See the Resolution Tab description for more information about rates. If the rates differ too much, adjust the optimisation and clock settings in the Resolution tab. If you sample with a rate that is outside the range 10% less than the ideal rate to 30% more, a warning is added to the Sampling Notes.

The total number of WaveMark traces you can sample depends on the 1401 type: 32 for Power1401 and 16 for Micro1401 mk II. For example, a Power1401 could sample 4 WaveMark channels with 4 traces plus 4 with 2 traces plus 8 with one trace.

The Conditioner… button is enabled if a programmable signal conditioner is present.

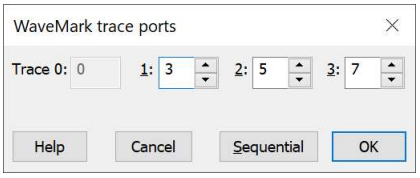
Spike sorting

Spike2 can match waveforms to a set of templates. This is normally used to extract single spike units from multi-unit recordings, but other uses are possible (for example extracting R waves from ECG data). If you record WaveMark data, a template set up window appears when you open a file for sampling.

Set WaveMark codes

When you display WaveMark data as a waveform in a time view you can change the marker codes of items that you select with the mouse. Hold down `Alt+Ctrl` and click and drag a line over the events you want to identify. On mouse up, a dialog opens in which you can set codes for the intersected events.

Non-sequential ports for traces



From Spike2 version 8 onwards, the ports used for tetrode and stereotrode data need not be sequential. If you click the ... button in the WaveMark set up dialog when 2 or 4 traces are active you can use this dialog to set the ports.

The port for trace 0 is fixed (it is set by the WaveMark dialog). The remaining ports are a free choice; there is currently no check that these are different from each other. The Sequential button sets the ports into sequential order.

Talkers

You can sample additional data channels from plug-in devices, called Talkers. Each Talker can provide one or more channels of data; these channels can be of any data type that Spike2 supports. A talker can also provide additional keyboard marker channel input.

For example, consider a blood pressure monitor that automatically inflates a cuff every 10 minutes and measures systolic, diastolic and mean blood pressure. If this device has a computer interface, the output could be added automatically to a Spike2 data file by writing a Talker interface. It could either provide three channels of RealMark for the three outputs, or it could provide a single channel of RealMark data with three attached data items.

Talkers include mechanisms to compensate for timing rate differences between the Talker data and Spike2, we call this drift compensation.

If you want to experiment with talkers, you can use the example talkers that are installed with Spike2 (unless you excluded them in a Custom installation). The web site also has information about talkers that can be used to

connect to real devices. Note that while some Talkers are free issue, others are licensed and require a fee for use.

Steps to allow Spike2 to be aware of a talker

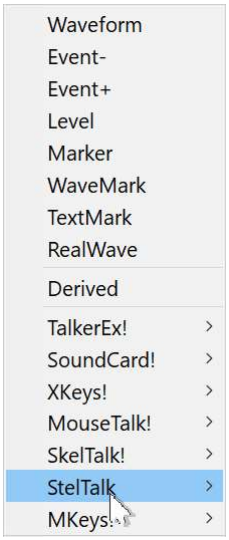
Spike2 maintains a list of Talkers that it knows about. When you first run Spike2, it will not list any talkers. Spike2 adds Talkers to this list when they connect to Spike2. For example, to connect the TalkerEx (one of the example talkers provided with Spike2), you should start Spike2 then run the TalkerEx.exe program. Open the Talker File menu and select the Auto-connect option or Connect. The talker should display a list of available channels. Once a connection has been established, Spike will be able to refer to this Talker. Each time you add a new talker you should repeat this procedure (though the connection details may vary).

From Spike2 version [10.16] Talkers can tell Spike2 where they are located, allowing Spike2 to start them automatically if they are not already running. There is also a Run command in the Sample->Talkers->*TalkerName* menu to launch a Talker if it is not running and Spike2 knows where to find it.

From Spike2 version [10.16] Talkers remember the drift state at the end of a sampling session and apply it at the start of the next. As most Talkers have very similar drift characteristics each time they are used, this can improve the initial drift compensation. Spike2 assumes that you have run with the same conditions as the last time you used the Talker. Before you use a Talker for the first time, it is a good idea to sample with it for 15 minutes or so with the sampling configuration you will be using (you can turn of writing to disk to save space, if you wish), so that Spike2 can make an initial assessment of the timing characteristics. If you do not do this, the new Talker will be assumed to keep perfect time until sufficient data has been acquired to calculate a drift factor.

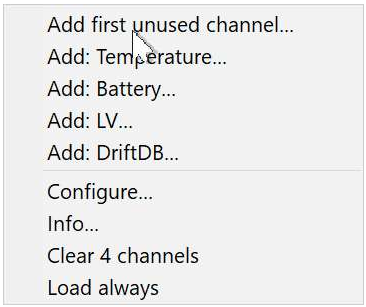
Adding Talker-based channels

Channels from Talkers are added to Spike2 from the Channels tab of the Sampling Configuration dialog by clicking on the small button to the right of the New... button. A drop-down list of possible channel types appears, with any Talkers known to the system at the bottom and the Derived option (to create a channel derived from an existing one) in the middle. If you select one of the normal channel types at the top of the menu, this is equivalent to clicking the New... button and selecting a channel type. You can choose to skip the channel configuration dialog by holding down the `Ctrl` key when you select the item. This will create a channel of the desired type with a standard configuration.



The lower section of the menu displays a list of Talkers registered with Spike2. If a Talker is not currently connected to Spike2, the talker name is followed by an exclamation mark (!) to warn you that you cannot sample with it until it is connected. In the example, taken from my test machine, there are all the example talkers (not loaded) and one loaded talker (StelTalk). You will likely have at most one or two. If you have set any Talker to Load always, there is a final option to clear all Load always settings.

When you use this menu to add a channel, it appears in the channel list as the current (selected) item. Whenever you add a Talker channel, the Source field in the list of channels displays the Talker name. If the Talker becomes unavailable to Spike2, the lines describing the channel in the sampling configuration become gray. If the talker is running, but has a problem, the Source field displays in red.



If you select a Talker name, in this example StelTalk, this opens a new pop-up menu. The menu is in two sections.

The first section of this menu allows you to add a channel from the selected Talker. If you select one of these options, the new Talker channel dialog opens to allow you to adjust the channel and giving you the opportunity to add a channel process.. You can create the channel without opening this dialog by holding down the `Ctrl` key before selecting the option.

The second part of the menu has options that relate to the Talker itself, rather than to individual channels. Only options that are relevant in the current situation display.

The full list of possible menu options are:

Add first unused channel...
This option is active if the selected Talker has channels that have not already been selected for sampling. If all talker channels are in use, this option is present, but is disabled (gray). Select this option to open the Talker channel dialog. If you select this option with the `Ctrl` key held down, the channel is added to the configuration, but the dialog does not open.

Add <Channel Title>...
This option appears for each channel that is available in the Talker. Some talkers have a lot of channels, so we attempt to group channels together. To do this, we scan all the available channel titles and replace all consecutive digits (0-9) with the # symbol, then amalgamate all the titles that are the same. For example, a device with channels called `EMG_00`, `EMG_01`, `... EMG_63` would list all these channels as EMG_# (x64); the number in brackets indicates the number of channels that match this title. Once the total number of remaining channels becomes manageable we list all the channels without compressing the names. There is a limit (currently 12) on the maximum items we will list. When you select this option, the first available channel that matches the selected title is added to the sampling configuration and the the Talker channel dialog opens unless you also hold down the `Ctrl` key.

Configure...
This option appears if the Talker has a configuration option and is running, or if Spike2 knows the Talker location, so can run it. If you select this option, Spike2 will attempt to run the Talker (if not already running) and if this succeeds, it will open the Talker Configuration dialog. You can also access this option from the Talker channel dialog. Configuration often affects the entire Talker and in extreme cases can cause all previously set channels for the current Talker to become invalid.

Run...
This option appears if the selected Talker is not currently running and Spike2 knows its location, and this location is on the current computer. If you select this option, Spike2 will attempt to run the Talker. Some Talkers can take a while to start up, so it can be useful to get them loaded early. This is the same option as in the Sample menu Talkers command.

Info...
This displays information about the current Talker. This is the same option as in the Sample menu Talkers command.

Clear <n> channels
If you have already set one of more channels of this particular Talker, you can use this command to remove them all from the sampling configuration. The `<n>` is replaced by the number of existing channels. This can be useful if you use the Configure... command and several channels become invalid.

Load always
This command toggles the Load always status of this Talker (shown by a check mark next to the option). From version [10.20], the sampling configuration holds a list of Talkers that are always loaded when the sampling configuration is used to sample regardless of whether any data channels are captured from them. This allows you to cause a Talker that is used to extend the Keyboard to log keyboard markers to load; the XKeys example talker does this. We can also imagine future talkers that use the Talker communications facilities to manage data capture, but that do not capture data themselves. For example, we could control programmable stimulators via the Talker interface. If any talker has Load always enabled, the Clear Load always (n) option will appear in the parent menu.

Using Talkers without a 1401 interface

Before Spike2 version 9, a 1401 was always required to sample data and to provide a time base to synchronise Talker input. From version 9 you can sample data using Talkers without a 1401. To do this the sampling configuration must not use any of the following 1401 features:

- The output sequencer
- Arbitrary waveform output
- Data channels sampled through the 1401 interface

When Spike2 detects a sampling request in this state it will not open a 1401 (even if one is available). Instead, it uses the high resolution timer of the host computer for all timing purposes. You still use the sampling configuration to set the time resolution and select Talker channels and you can also use the keyboard marker

and TextMark input channels to annotate your data. Note that you can still set a triggered start, but this will be ignored and the Trigger check box in the sampling control bar is disabled.

As the timing drift for a talker is measured against the computer high-precision clock when no 1401 is used and against the 1401 internal clock when a 1401 is used, the drift rate in these two situations will differ, so be careful if you switch and consider running a 'calibration' run before using the Talker for important data. We are researching the best way to deal with this for a future Spike2 release.

Talker timing synchronisation and timing drift

The talker system allows data from a wide range of devices to be piped into a Spike2 data file. The timing systems in most devices are based on crystal oscillators, which are typically accurate to a few seconds a day, so absolute timings between devices will tend to drift apart. The Talker system has built-in features that allow us to detect timing drift and compensate for it. We assume that devices have a stable clock (or that if it does change, it does so slowly). If a 1401 device is used to control sampling, it acts as the timing master. Otherwise, the high-resolution timer of the host computer is the timing master. Timing corrections for event-based data is straightforward; corrections for waveform data require interpolation (implemented with cubic splines).

If a talker can connect to different hardware devices for each sampling session (typically an animal implant), it can store different drift information for each device using a *Device qualifier*. This removes the need to manually clear the saved drift information for the last device and potentially improves the quality of data capture.

Relative timings between channels sampled by a 1401 are very accurate (microsecond accuracy). Timings between a Talker and a 1401 or a different Talker are as accurate as we can make them, and should not have systematic timing drift. If you depend on timings between different Talkers or a Talker and a 1401, you should record the same signal into both devices and measure the timing accuracy. Do NOT assume the microsecond accuracy of the 1401 family.

First use of a talker

When you use a Talker for the first time, or after changing between sampling with a 1401 and sampling without one, it is a good idea to run a dummy sampling session (you can turn writing to disk off to save disk space if you wish) for 15 minutes or so to get a reasonable estimate of the drift rate. If you do this, make sure that equipment used has been on long enough for temperatures to stabilise (temperature seems to be one of the strongest causes of timing changes).

Running multiple copies of Spike2 and Talkers

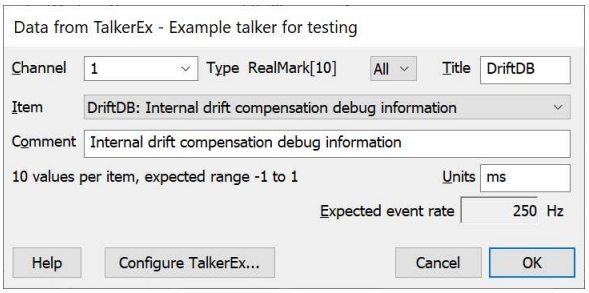
If you have multiple copies of Spike2 running and you want more than one of them to use a Talker you must use the `/Tn` command line option (n is 0 to 9, with 0 being the default setting) for both Spike2 and the Talker so that they can find each other. If you do not do this, the first copy of Spike2 that runs will grab the default Talker communication pipe and will be in control of all talkers.

If you want to use multiple instances of a talker you must use the `/Nn` command line option (`n` is 0-9 with 0 being the default setting) when the talker is run to tell it which Talker instance it is. If the talker number is other than 0, the number is appended to the talker name. For example, if you wanted to run two instances of `TalkerEx` (an example talker that simulates data) you would run the second copy with the command line `/N1` and it would show up with the name `TalkerEx1`. The talker name is important as we keep a registry of known talkers on a machine and the registry is indexed by the talker name.

Talker Log

Most talkers have a Help menu with the item Enable Log file output. If you enable this, each Talker session writes a log file to the folder Documents\CED Talker logs with the title: <TalkerName> log <Date>.txt, for example: TalkerEx log Wed Sep 14 12-03-29 2022.txt. The contents of this file are inscrutable and likely only of interest to a CED engineer.

Talker dialog



For the purposes of this example we will assume that you have selected a Talker called TalkerEx that we provide with Spike2 as an example. This example Talker can generate data items of all possible types.

To run TalkerEx

Locate Spike11 in the Windows Start menu and within it select Talker example emulate data types. Alternatively, you will find this Talker as `TalkerEx.exe` in the Talker folder in the Spike2 installation folder. Double-click it to set it running. Once running, use the TalkerEx File menu to select the Auto Connect command. Navigate back to Spike2 and select TalkerEx from the drop down list next to the New... button in the Channels tab of the Sampling Configuration dialog.

For other Talkers the process is similar, but the name of the device is different. See the documentation that comes with the talker for details.

If you run a Talker on the same machine as Spike2 and connect to it, Spike2 saves the Talker location and the command line used to run it. Subsequently you can run it again with the Sample menu Talkers->TalkerName->Run command. Spike2 will run the Talker automatically if you open a file for sampling that requires the Talker and Spike2 knows the talker location.

Talker dialog fields

When you select a Talker in the Sampling Configuration dialog by clicking on an enabled talker name, a new dialog opens. The editable fields are:

Channel — This field sets the channel in the Spike2 data file to use to store the Talker data. You can change the channel number, but not to a channel that is already in use.

Type — The type of data in the channel (determined by whichever Talker Item you select). If this is a RealMark channel, the type is followed by [n] where n is the number of data items attached to every RealMark. For RealMark data there is also a drop-down list that allows you to set the Title and Units of each RealMark data item. This is set to All to set the default title and units and 1-n to select a particular item.

Title — The channel Title. This is usually set by the Talker, but you can edit it. In the case of RealMark items with multiple data values you can choose to set individual titles for each item. Any item with a blank title is given the default title. If you set a long title and sample to a 32-bit `.smr` file, the title will be truncated.

Item — Every Talker that can generate this dialog provides one or more data channels to write to the data file. This field selects the Talker channel to add to the sampling configuration. When you add an Item, it is removed from the list of available items for this Talker. Once all available items have been added for a Talker, the Talker name is disabled so you cannot add further channels.

If an item is marked as currently unavailable, this may be because it requires configuration. In the case of the example talker use the Configure TalkerEx... button and enable the channels to make them available.

Comment — Text to give more information about this channel, stored in the data file.

Units — If this field is present, it sets the units for the channel. If the channel holds RealMark data, you an set units for each data item, as for the Title field.

The remaining fields are not editable and give information about the channel. The values displayed here are supplied by the Talker.

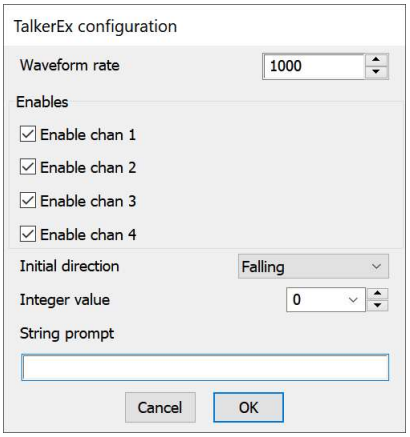
Configure TalkerName...

This button is enabled if the Talker supports configuration and is connected. It leads to a Talker Configuration dialog, which can be local (part of Spike2) or remote (part of the Talker application).

Configure Talker dialog

You open this dialog from either the Configure <TalkerName> button in the Talker Channel dialog or from the Configure... pop up menu option in the New... drop down menu in the Sampling Configuration dialog.

This command opens either a simple, generic dialog controlled by Spike2, or if the Talker needs more complex setup, we tell the Talker driver to display a configuration dialog and wait for this Talker set up process to end.



In the case of the TalkerEx example, it opens the local, simple, generic dialog allowing you to modify simple Talker parameters. This dialog is constructed based on information exchanged with the Talker, and is a list of items that can be integer or real values, check boxes, choices from a list or strings. The Talker can also request that consecutive items (the channel enables, in this case) are grouped. For the TalkerEx example, the Enables must be checked to allow you to use the corresponding talker channel. The Initial direction field lets you set the edge of the example Level event channel. The remaining fields are just examples of what is possible and have no effect.

Important
You must read the documentation included with your Talker for a description of the configurations options available to you.

Changes made in this dialog can have a radical effect on the capabilities of the Talker and may make the current (or all) Talker channel(s) unavailable. If this is the case, you may need to clear all existing channels for the Talker before configuring the system.

Consider a Talker for a device that can handle several different remote items that are selected by this dialog. If you change the remote item, the list of available channels can also change. Depending on the device, Spike2 may be able to match existing channels in the configuration to the new device, or it may not.

Caution
Complex talkers that provide their own set up dialog system run it as a separate entity from Spike2. If a talker is hosted on a remote computer, the configuration dialog will run on the remote system.

Apart from the problem of physically finding the machine running the dialog, remote dialogs can take a while to open. In some cases, where we must interface with third-party software, even when the dialog is on the same computer you may need to locate the dialog yourself; in at least one case, the dialog opens but does not move itself to the front of the display, so can be hidden under other open windows.

Talker file location and documentation

Standard installations of Spike2 install example talkers in the `Talker` folder and the optional talker documentation in the `ExtraDoc` folder inside the Spike2 installation folder. You can exclude the example talker support and documentation by choosing a Custom install. The Talker documentation is in the file `Talker.pdf` and is technical in nature; you do not need to read this to use the example talkers.

As Talkers are independent of Spike2; apart from the example talkers described here, you will not find documentation for specific Talkers in the Spike2 documentation. A Talker that is written by CED to support specific hardware will have its own documentation, and we would hope that any Talkers provided by third parties to support their hardware will be suitably documented. If you have a problem with a non-CED originated Talker, it is unlikely that we will be able to provide you with support; you will need to take this up with the authors.

Communications between the Talker and Spike2 are done using *named pipes*. This allows the Talker to run on the same computer as Spike2, or on a separate computer linked by a network. There is documentation and example code available to allow anyone (with the requisite programming skills) who wishes to write their own Talker to do so..

If you run multiple copies of Spike2 on one computer, the first one run will create the Talker communication service using the default pipe (equivalent to running with command line option `/T0`) and will connect with all talkers on the default pipe. Most talkers support the `/Tn` command line arguments that lets you set the

communication pipe number they will use. Each copy of Spike2 will only see Talkers that communicate on the same pipe. Once a copy of Spike2 has grabbed a particular pipe, no other copy can use it.

You can download the Talker development kit, including the source code of the example talkers, from our web site. This download contains sufficient information for a programmer to create their own talker.

List of talkers

Spike2 keeps a list of all Talkers that it has been exposed to in a repository called `sp2talks.datx` in the all users application data directory (for example, `C:\ProgramData\CED\Spike11`). This allows Spike2 to configure sampling for talkers that are not currently connected. If you delete this file, all previous knowledge of Talkers will be lost, but Spike2 will still run and will start accumulating information again.

Example Talkers

The standard Spike2 installation copies example talkers and the talker technical documentation to the `Talker` folder in the Spike2 current user data folder. Custom installations can choose to exclude the example talkers so if you cannot find the example talkers, run the Spike2 installation again, choose Custom and check the box to include the example talkers.

How Talkers communicate with Spike2

When Spike2 runs, it creates a master named pipe called `Spike2Talkers` that can be opened by Talkers running on the same machine or on the same network. If Spike2 is running on a machine called SID0000 (this name is displayed as the Computer name field of the Spike2 About box), the master pipe is referred to on the network as `\\SID0000\pipe\Spike2Talkers` and within the computer where Spike2 is running as `\\.\pipe\Spike2Talkers`. Talkers open this master named pipe, write a packet of information announcing themselves, wait for a response, then close the master pipe. The Spike2 end sets up a process to handle the Talker and creates a new named pipe for use by the Talker. The Talker opens the new pipe and the Spike2 process and the talker communicate through it. The details of the communication are described in the Talker documentation; you do not need to know the details to use a Talker.

Almost all problems with Talkers are in establishing the link between the Talker and Spike2 across a network. It is important that the talker machine can connect to the Spike2 machine without needing to input a user name and password. You can check this in Windows on the Talker machine by opening the File explorer Network folder and checking that you can see the name of the Spike2 machine. Double-click the Spike2 machine name. If all is well, you will see a display of any shared objects (for example folders and printers) on the Spike2 machine. However, if you get a dialog box asking for a user name and password, you must log into the Spike2 machine using the user name and password for an account with access to the Spike2 machine; once this is done, the Talker will connect.

It is usually much more convenient, and timing is more certain, when a Talker runs on the same computer as Spike2.

Domains

It is our experience that if Spike2 is running on a machine that is logged into a Domain, to connect with a talker over the network, the Talker must be logged into the same Domain.

Using a Talker

For Spike2 to connect with a Talker, it must be running, either on the same computer as Spike2 or an a remote computer that is accessible over a network. There are several ways to run a Talker locally:

1. By locating the Talker application and running it. You must do this the first time you use a Talker.
2. You can run the example talkers provided with Spike2 by locating Spike11 in the Windows Start menu; there are links to the example talkers within the Spike11 folder.
3. With the menu command Sample->Talkers->*TalkerName*->Run command. This works from Spike2 version [10.16] onwards for recent versions of Talkers that have been previously connected to Spike2.

Running a Talker on the same computer as Spike2

There are several ways to run a Talker on your local machine by hand:

1. If Spike2 already 'knows' about the talker you can use the Sample menu Talkers->TalkerName->Run command.
2. Use the Windows File Explorer to navigate to the Talker .exe file and double-click it.
3. Open a Command window and type in the path to the .exe file (for example `C:\Program Files\CED\Spike11\Talker\TalkerEx.exe`).
4. Create a short cut to the Talker .exe file on your desktop and double-click it. You can use the short cut to set talker command line options.

Running a Talker on a different computer

If you have a local area network connecting your computers, you can run them on a different computer by using a command line parameter to identify the remote machine that is running Spike2. The following assumes you are on the remote machine and have opened a command prompt:

```
C:\PathToTalker\TalkerName -sComputerName -n1
```

Where `C:\PathToTalker\` represents the location of the talker and `TalkerName` is the name of the talker. The `-s` option sets the name of the computer that is running Spike2. If you omit the `-s` option, the host application (Spike2) is assumed to be running on the same computer as the Talker. Talkers are identified by name and if you have multiple talkers, they must have different names. The `-n` option allows you to run multiple identical talkers; the number immediately after the `n` is appended to the Talker name, so `TalkerEx -n1` generates a talker called `TalkerEx1`. If the computer running Spike2 is called `Samuel` and your talker is in the folder `C:\Program Files\CED\Spike11\Talker`, you can connect with:

```
C:\Program Files\CED\Spike11\Talker\TalkerEx -sSamuel
```

Example Talkers provided with Spike2

Unless you chose a custom Spike2 installation and excluded the example Talkers, they are copied to the `Talker` folder inside the Spike2 installation folder. The example talkers we provide can all run locally or remotely:

`TalkerEx`	Provides dummy channels of every type and generates dummy data. It also logs key presses and sends them to the keyboard marker channel. This is purely an example; it has no practical use beyond getting you comfortable with Talkers. Note you must use the Configure button and enable some of the channels to use them. This is useful as a learning and diagnostic tool.
`MKeys`	Provides an additional marker channel that logs key presses to a Marker channel (*not* the keyboard marker channel). This acts as a remote keyboard, so only makes sense if run on a different computer. You could use this to log responses from multiple subjects.
`XKeys`	Provides additional input to the standard keyboard marker channel. This acts like a remote keyboard that works in parallel with the keyboard on your computer, so only makes sense if run on a different computer. You could use this to log responses from a single subject.
`MouseTalk`	Uses the mouse position to provide a RealMark channel with two values, being the x and y co-ordinates. This probably makes more sense when run on a different computer, but does work on the same computer as is running Spike2. We have seen this used with an XY window to get a subject to track a position with the mouse.
`SoundCard`	Uses the Windows audio input (if present) as a 1 (mono) or 2 (stereo) channel waveform source. This can be useful to record audio or other waveform signals. You can combine this with the Activity detect online process to generate an audio commentary channel that only records when you speak.

Talker startup displays

When you start a Talker, you will see a message confirming connection or giving you a reason why the connection could not be made. Talkers that cannot connect will keep trying to connect until you close them; it is OK to start a Talker before you start Spike2.

Once the Talker and Spike2 are connected, you can open the Spike2 Sampling configuration, go to the Channels Tab, and drop down the list next to the `New...` button. In addition to the standard list of channel types, you should also see the name(s) of the connected talker(s). Spike2 remembers Talkers it has been in contact with; if it knows about talkers that are not currently connected, these talker names appear in the list followed by an exclamation mark (!) to warn that they are not currently available.

Select the name of the Talker in the drop down list to open the Talker dialog to add a talker channel to your sampling configuration.

Standard Talker command line options

The following command line options are the same for all talkers. Talkers may have additional options; these will be described in the Talker-specific documentation.

Option	Param	Purpose
`S` or `s`	`ServerName`	Talkers detect a server by searching for a named pipe. The default search is on the same computer as the Talker runs on. However, you can search different machines on the network by setting the name of the machine to search. If the target machine name contains a space you must enclose the name in double quotes: `/S"Lab machine"`
`T` or `t`	`1-9`	Should match the command line `/T` option that the host program (Spike2) used or omit it if the server omitted it. Talkers search for a pipe on the host computer with a known name to initiate communications. If there are multiple hosts running on a machine the second (and third) host must use `/T` options followed by one of `1` through `9` to change the pipe name by appending `1` through `9` to it. You can use 0 as a parameter, but this is equivalent to omitting the option and does not append 0 to the pipe name. Example: `/T1`
`N` or `n`	`1-9`	Once a Talker has found a host program (Spike2), the host creates a named pipe to transmit data between the Talker and the host. The pipe name includes the Talker name and the Host number (`/T` option). The `/N` option appends the parameter (1 to 9) to the Talker name so that if you run a second copy of the same talker, the server can tell them apart. The server also uses this name to save information about the Talker. Omit this option unless you want to run multiple copies of the Talker with the same server. You can use 0 as a parameter, but this is equivalent to omitting the option and does not append 0 to the pipe name. Example: `-n1`
`H` or `h`	`HardwareName`	Optional. Some Talkers can connect to their hardware across a network; see the detailed Talker documentation. If this is supported by a Talker, `HardwareName` is the network name of the machine. If the name contains a space you must enclose the name in double quotes: `-h"Data source"`

When a Talker runs it parses the command line to find these options (which it expects to find separated from each other by spaces) and sets itself up accordingly. You can also set these options interactively within the Talker from its Connection settings dialog. Note that if you have two talkers of the same type connected to a computer, they *must* use the `/N` option to make the pipe names different. This applies even if they are connecting to different hosts (for example you can run multiple copies of Spike2 or a mix of Spike2 and Signal). Likewise, if you have more than one host running on the same computer, you must use the `/T` option when starting each server to make the pipe used for setup unique.

Automatic Talker connection

From Spike2 version [10.16] onwards, Spike2 will attempt to start all local (not remote) Talkers used by the current sampling configuration that are not already running when it creates a new data file for sampling. The Talkers will start with the same command line as the previous time they ran. This works for the latest generation of Talkers (which includes all the example Talkers).

Talker source code and documentation

You can get the example Talkers source code and Talker documentation from our web site by going to the Downloads page and then selecting the *Talker software development kit*. The installation program places the example Talker applications and the documentation for writing talkers (`Talker.pdf`) in the installation folder. It also copies folders holding the Talker source code, set up to be built with Microsoft Visual Studio.

Talker errors when loading sampling configurations

When you load a sampling configuration that contains Talker channels it is possible to be warned about Talker-related problems.

Unknown or incompatible talker(s)

This message can occur if you load a sampling configuration that references a Talker that you have not connected to previously. The solution is to connect the Talker to Spike2. Having done this, you should load the sampling configuration again to ensure that any Talker-specific configuration information in the configuration is loaded.

You can also get this message if you load a sampling configuration that requires a version of a Talker that is not compatible with Talker connected to your computer. Make sure the named Talker is connected and is up to date (you can download the latest version from our web site, subject to licences). If the sampling configuration is using an old talker version that is not compatible with the latest version you will have to delete the affected channels from the sampling configuration and reconfigure them with the updated Talker.

Error reading/setting talker Name configuration, code n

If a Talker saves configuration information in the sampling configuration and that information has become corrupt or is incompatible, you will get this message. Follow the same steps as for an unknown or incompatible Talker. If all else fails, you will have to delete the Talker channels and reconfigure them.

SoundCard.exe talker

The SoundCard talker allows you to record input from your Windows sound card as part of your data file.

To use this talker, start Spike2, then run the SoundCard.exe program in the Spike2 program folder (usually `C:\Program Files\CED\Spike11`). The first time you run this you will need to configure the program to select a sound source and the channels and sampling rate. You should use the lowest sampling rate that is suitable for your purposes. Do not use multiple channels unless you really need them to optimise disk space use. If you have a choice of data widths, choose 16-bit (Waveform data is stored as 16-bit data, so choosing 8-bit just reduces the quality). If you choose higher resolution than 16-bit, the data is saved as RealWave, which uses double the disk space of Waveform data.

You can use this together with the Triggered sampling mode and the Derived channel activity detect feature to make a voice-triggered memo taker that only uses disk space when you speak. To set this up, start by recoding some speech to get an idea of a suitable trigger level. Then use the Activity Detect process in Activity mode to generate markers when you start to speak and a different code when 'activity' (in this case, speech) falls to a low level. Then sample in Triggered mode and set a trigger group to control recording of the sound channel and set it to trigger on the derived channel with enough pre-trigger time to capture the start of each memo with a duration long enough to cover your longest memo. Set the off code to the one set for end of activity.

Time drift compensation

One very difficult problem when importing data from non-synchronised devices is how to cope with the inevitable time base differences between time as seen by a 1401 interface and time as seen by a Talker. There are generally two components to this problem:

1. Where is time zero.

2. How fast is time passing.

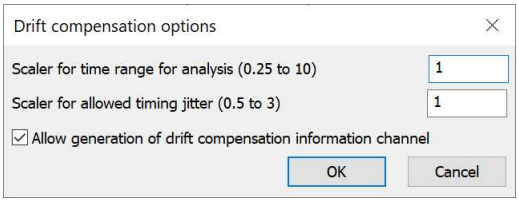
You can solve the first part of the problem if the Talker supports a triggered start, in which case you can use the same trigger for both the 1401 and for the Talker. However, many talkers do not support this. The second way to resolve this is to record the same signal on both a 1401 channel and a Talker channel (or superimpose it on channels). after recording you can apply a time shift to the Talker (or 1401) data to align the data if the time offset is significant.

The second part of the problem is more difficult to fix. Most modern electronic equipment runs at a rate determined by crystal oscillators. Unless these are specially selected to be high-accuracy and high-stability, they will typically have an accuracy of 50 ppm (parts per million), a few seconds per day. Recent 1401s use an oscillator with an accuracy of around 5 ppm over a reasonable temperature range (though we only claim 50 ppm in the handbook).

We would expect that most modern electronics would be accurate to around 50 ppm (though we have seen equipment with a time base much worse than this). This means that if you sampled a waveform with a 1401 and with a Talker, without any form of correction, the two signals would slowly drift out of alignment, maybe by 200 ms per hour.

To compensate for this, the Talker system accepts that this is the case and attempts to compensate for it by warping waveforms and adjusting event times to hold signals to be within a millisecond or so. It makes the assumption that the time base is not grossly wrong and that any change in the rate of drift is small.

If you want to visualise the time base drift correction, all Talkers have an option to generate Drift information as a channel of RealMark data. To enable this, use the Talker File menu Drift compensation options... command:



Scaler for time range for analysis (0.25 to 10)
This value is used to adjust the time range over which the analysis of the data timing drift occurs. The standard and recommended value is 1. Values less than 1 will cause Spike2 to use a smaller time range for the drift analysis, values greater than 1 will give a greater time range.

If the drift timing information shows cyclical variations (such as a saw-tooth often caused by the USB bus) or is otherwise a bit messy you can set a value greater than 1 (values from 2 to 3 seem to work well) to cause the drift analysis to occur over a longer time range and thus smooth out any timing variation, but at the expense of a quick analysis and compensation that adapts quickly to changes in the drift.

If your drift timing information is clean and noise-free but the rate of drift varies significantly over time, you could set a value less than 1 to make the timing analysis more responsive to changes in the rate of drift. You can enter values from 0.25 to 10, but setting values less than 1 is not recommended. Most users leave this value set to 1.

Scaler for allowed timing jitter
This value is used to adjust the limits of acceptable timing variations. Spike2 uses these limits to discard occasional extreme timing values rather than allowing them to interfere with the drift analysis. The standard and recommended value is 1. Values less than 1 will cause Spike2 to discard more drift timing information, values greater than 1 will cause Spike2 to make use of more information. This is most commonly used to set a value greater than 1 if it appears that the drift compensation system is not operating correctly because there is too much variation in the drift timings. You can enter values from 0.5 to 3. Most users leave this value set to 1.

Allow generation of drift compensation channel
Check this box to allow the Talker to generate an extra channel of RealMark data with 10 values that show how the drift analysis and compensation is working. Normally you will leave this box unchecked; if you are having trouble with a Talker, a CED engineer may ask you to enable it. Once enabled, a new channel becomes available for this talker in Spike2. To try this out, you can enable this option in the TalkerEx example talker and add a TalkerEx channel and select the DriftDBL: Internal drift compensation debug information item. You can run the following script (make a data file holding sampled data the current view and adjust the script `dc%` constant to hold the channel number you set for the drift compensation values:

```
'$DriftShow|Display annotated Drift information
const dc% := 1; 'Channel holding drift info - you must edit this
if ViewKind() then Message("Select time view holding drift data");halt endif;
if (ChanKind(dc%) <> 7) then Message("Channel %d is not a RealMark", dc%); halt endif;
var nItems% := ChanIndex(dc%, -2); 'Number of attached values
if (nItems% < 10) then Message("Channel %d has %d items, needs 10", dc%, nItems%); halt endif;
var i%,c%;
ViewStandard();
```

```
Draw(0, MaxTime());
ChanTitle$(dc%, "Drift info|Raw delta|Av. delta|StDev delta|Fit value|# fit|fit slope|fix intcpt|Av. s
ChanUnits$(dc%, "ms|||||Count|ms/s||ms/s|||");
for i% := 0 to 9 do
    c% := i% = 0 ? dc% : ChanDuplicate(dc%);    'original or duplicate
    DrawMode(c%, 3);     'Display as waveform
    ChanIndex(c%, i%);   'Set trace to display
    ChanShow(c%);        'ensure channel is visible
next;
Optimise();
```

The drift analysis looks at the difference between the data time seen by the talker and the data time as seen by Spike2 when each data packet arrives from the Talker. There will be some jitter in this due to the uncertainty of how long it takes for data to be sent to Spike2 and timing jitter in the Talker and Spike2 due to system activity. However, when averaged out over time, we can discern the general drift in timing. This is done by fitting a straight line to the data time differences. The slope of this line indicates the rate of time drift and the intercept is the time offset. The 10 items of data returned are:

#	Title	Units	Meaning
0	Raw delta	ms	Difference of last packet of Talker time to Spike2 time (may be rejected if sudden change)
1	Av.delta	ms	Average time difference over analysis time range
2	StDev delta	ms	Standard deviation of time differences from average (used to decide if we reject a value)
3	Used delta	ms	Input values used for fitting (excludes values with a sudden change)
4	Delta pts	Count	The number of data points used to average differences.
5	Slope	ms/s	The most recent slope of the fit. This would be 0.0000 if the data were perfectly timed with no noise.
6	Slope pts	Count	The number of difference points used for fitting.
7	Avg slope	ms/s	The running average of the calculated slopes.
8	Used slope	ms/s	The slope value we are using
9	Drift	ms	The deduced drift, which is the time correction applied to the data at this point.

Keyboard channel markers

The drift calculations go through several phases while Spike2 accumulates drift timing information. When you enable the drift compensation channel, Spike2 adds markers to the keyboard channel to indicate when the phases change. From [10.16] onwards, these markers have codes `Dx` (`x` is a hexadecimal digit). These codes are for CED diagnostic use. The current codes and their uses are:

Code	Meaning
`D1`	Got initial baseline data, correction using saved drift value from previous sessions (0 if no previous value)
`D2`	Able to produce smoothed time differences
`D3`	Able to start generating drift values from the sampled data
`D4`	Current drift values start being used for correction
`D5`	Fully switched-over to using current values values for correction

Initially, we use the last saved drift rate for this talker (on the grounds that most Talker sources run at a pretty constant rate that tends to remain much the same between sessions). If there is no saved rate we start with an assumed drift rate of zero. We start by establishing a baseline of time differences to give us some idea of the distribution of timing errors. Typically we get a fairly tight distribution of errors with occasional large errors caused by other processes running in the computer or by some activity in the Talker. Once we have a baseline and some idea of the distribution of errors we start generating smoothed time differences. Once we have sufficient values we start predicting time drift values. However, we wait a while so that we are confident that the drift value is not changing wildly. We then start adjusting the drift value to get the data to align; this is not

done suddenly, as that would cause a discontinuity in the data. At some point, when we have sufficient data, we rely on the newly sampled data entirely to control the sample drift.

Device qualification

From Spike2 version [10.19] we allow talkers to qualify the talker drift information with a device identifier. This is for use in situations where a talker can detect to a choice of devices, each with different drift rates. For instance, a Talker might link to a wireless base station that can connect to one of a choice of multiple devices, each with their own drift rate. It would make no sense to restore the drift rate for the last connection to this talker as it might be for a different target device and using the previous drift rate could well make the drift situation worse, not better. If you connect to a talker than knows how to qualify the drift information, this happens automatically. There is no action required by you to enable this. The Talker Info... dialog will show the current qualifier, if your Talker supports this feature.

Setting up on a new system

Talkers load the last-known drift rate when they start. This improves the initial drift compensation a great deal. However, the first time you use a Talker, or after changing a system component that might affect timing (of the computer or the Talker), or if you have told the Talker to forget about timing information, we have no saved information. In this state, the Talker data will drift away in time until enough timing information has been gathered to allow a correction.

When the drift rate is unknown, we recommend that you run a dummy sampling session, ideally for 30 minutes or more. Follow this procedure:

1. Switch on the hardware and allow it to warm up (maybe 30 minutes) to reach thermal equilibrium. The rate at which most electronic clocks run is temperature dependant.
2. Run the Talker(s). If you have never run them previously, or they are on a different computer, you will have to locate and run them yourself. Otherwise you can use the Sample menu->Talkers->TalkerName->Run command.
3. Open the Sample menu->Talkers->TalkerName->Info... dialog and make sure that the Lock talker drift rate check box is clear and close the dialog.
4. Configure the channels and Talker(s) as you would for a sampling session.
5. Open a new file, ready to sample. You can disable writing to disk unless you really want to keep the data.
6. Start sampling and sample for 30 minutes or more.
7. Stop sampling and Save the data file (do not Abort sampling - if you do not Save the file, the drift rate is not updated).

From this point onwards, each time you sample with a Talker, Spike2 reads the initial drift rate from the saved Talker information and updates it at the end of each sampling session. If you would prefer to stick with the value you have and not risk changing it with an atypical session, you can use the Lock talker drift rate check box in the Sample menu->Talkers->TalkerName->Info... dialog.

Derived channels and Real Time processes

Version [10.01] added the ability to process the real time stream of data from a 1401 waveform channel to either create a new channel, or to modify the source channel. Version [10.09] extended this to work with Talker waveforms. Version [10.13[further extended this with activity detection for derived channels.

Derived and Processed channels

When we use this to create a new channel, we refer to this as a *Derived* channel. When we apply it to the same source channel, we call this a *Processed* channel. Processed and derived channels are stored in the data file as RealWave (floating point) values regardless of the original type (Waveform or RealWave) unless you use the Activity Detect process in which case the channel is saved as a Marker. You can add up to 5 processes to a channel (5 is an arbitrary limit). Data is piped from the source and through the processes, in order, before being written to the data file.

Processes are applied as part of the real-time data capture thread; there will be limit on how much processing can be applied when running data at high speed; this limit will be determined by the capabilities of your computer. You can get an idea of how hard the sampling thread is working from the Sample Status bar, in particular from the CPU field.

This data processing happens in the PC, not in the 1401 interface. This means that an output sequence run in the 1401 using a `CHAN` command to inspect a 1401 data channel has access to the original 1401 data, not to the processed data.

You can create and manage derived channels from the script language using the `SampleDerived()` and `SampleProcess()` script commands.

If you modify the type of the source channel such that it cannot be used as a source channel or delete it, the derived channel is deleted.

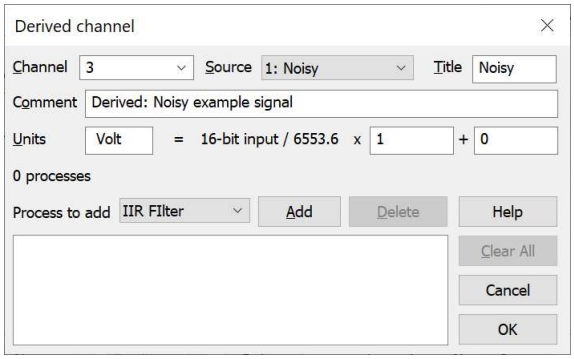
Alternative ways to process data

If you need to process your data continuously during data capture and the provided real time processes can achieve the desired result you should always use them. Your alternative is to have an on-line script that tracks data capture and does what is required. However, scripts run in spare CPU time, so are not real time. They also require knowledge of the script system and are not as fast as a real time process. If you do not have a script running all the time you sample, you can set scripts that run on specific trigger or timed events, or that run periodically during data capture or when capture starts and stops.

One particularly effective method to combine real time processes and a script is to use the Activity detect process to generate a Marker that is used as a trigger for a script that completes the processing task.

Add a Derived channel

You create a derived channel interactively from the Channels Tab of the Sampling configuration dialog by dropping down the list next to the New... button and selecting Derived. This item is enabled when your sampling configuration contains at least one Waveform or RealWave channel. When you select Derived, a new dialog opens (the same dialog opens when you click the Process... button in a Waveform or RealWave channel in the Sampling configuration to add a real time process to a channel):



The script language equivalent of this dialog is the `SampleDerived()` command. The initial dialog settings are obtained by finding the first unused channel in the configuration and by selecting a source as the first suitable existing channel. The dialog fields are:

Channel

This is the channel number of the Derived channel in the sampling configuration, which is the channel number in the output data file.

Source

This is the Waveform or RealWave channel in the sampling configuration that is used as the source of data for this Derived channel. If you change the source channel, the *Units*, *scale* and *offset* fields will also change to match the new source channel. The *Title* and *Comment* fields will also change unless you have edited them from the default values copied from the previous source channel. If the dialog is opened from the Process... button of the Channel dialog, this field cannot be changed.

If this is a Derived channel, any processing applied to the source channel is ignored. For example if the source channel has Rectify and Low pass filter processes applied, the source data used by a Derived channel will be the original data.

Title, Comment
The channel title and comment are copied from the source channel when the derived channel is first created. After this, if you change the source, these fields are preserved if you have modified them to differ from the text copied from the previous source, otherwise they are copied from the new source. These fields can use place holders, for example `%c` for the channel number.

Units, scale, offset
These fields are copied from the source channel each time the source channel changes; it is unusual to edit them. Spike2 uses the scale and offset to convert the channel values back to integers for certain situation, such as when converting a RealWave channel to a Waveform channel for DAC output. The displayed values apply to the input data to the channel. Waveform channels are stored as 16-bit integers (range -32768 to 32767), RealWave channels are stored as 32-bit floating point values.

From version [10.09] onwards, when we create the data file from the sampling configuration, we modify these values to reflect changes made by processes. For example, a *Difference* process (see below) multiplies the scale by the sample rate, sets the offset to 0.0 and adds `"/s"` to the units. Adding a process does not change the displayed values in this dialog. The line below ("0 processes" in the example above) lists any of the units, scale, offset and sample rate that the processing modifies.

Process to add
The area below *Process to add* starts off empty and holds a list of the processes that have been attached to this channel. The *Process to add* field is a drop down list of the processes that can be added:

IIR filter You can add Low pass, High pass, Band pass, Band stop and resonator IIR filters. The IIR processing is relatively time-consuming compared to the other processes and the time taken per point depends on the order of the filter; Band pass/stop filters take double the time of low and high pass filter. When you Add an IIR filter, this opens the Digital filter dialog in IIR mode where you can design the filter to apply. If you add a high-pass or band-pass filter, this sets the channel offset to 0.

Note: You are not allowed to add an unstable IIR filter. However, if you add a filter, then change the channel sampling rate, it is possible for the result to be unstable. This is detected when you prepare to sample and sampling will not start until you adjust the filter.

Rectify Full wave rectification replaces negative data values $-x$ with $+x$ and leaves positive values unchanged. You can also select various half-wave rectification modes. We do not allow you to follow a Rectify processes with another as this is pointless. You set the rectification mode in the Rectification mode dialog.

Difference Each data value is replaced by the sample rate multiplied by the difference of the current value with the previous value. This is equivalent to the instantaneous slope of the signal. This will tend to be noisy, so you will usually follow it with a Low pass filter to smooth the result. This also modifies the channel scale, offset and units.

Down sample This process takes every n^{th} sample from the data stream. You set the `n` in the Down sample dialog that opens when you Add this process or double-click the process in the list of added processes. We do not allow you to add two down sample processes together as this is inefficient; set a larger divide ratio for one process.

This will usually be preceded by a Low pass filter to remove input frequencies that exceed half of the resulting sample rate. For example, if the source channel is noisy, with noise components up to 1 kHz, but we are only interested in signal frequencies of up to 30 Hz, we could sample it at 2.5 kHz, Low pass filter it to 30 Hz, then down sample by a factor of 25 to get an effective sample rate of 100 Hz.

Activity detect Derived channels only. This process takes a waveform as input and converts it to a Marker channel. There are 4 modes available: Activity, Peak detect, Trough detect, Peak and Trough detect. Adding a process or double-clicking an existing Activity process opens the Activity detect dialog where you can configure the process.

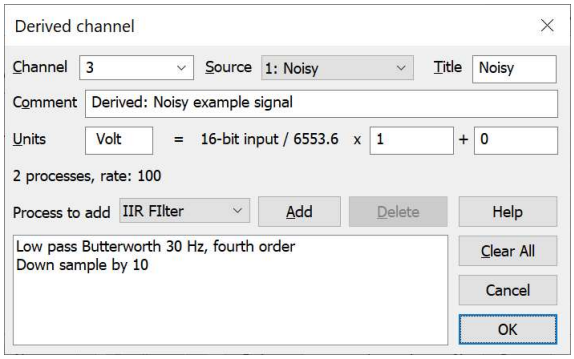
Add
Click this button to add the currently selected process to the end of the list of processes. This button is disabled if the list is full (we allow up to 5 processes to be added), or if the proposed process is Rectify or Down sample and the last added process was the same. The Rectify and Difference processes add immediately. The others open dialogs to configure the added process.

Delete
This is enabled when there is at least one item in the list of added processes. Click the button to delete the selected item from the list.

Clear All
This is enabled when there is at least one item in the list of added processes. Click the button to remove all processes from the list.

Edit a process

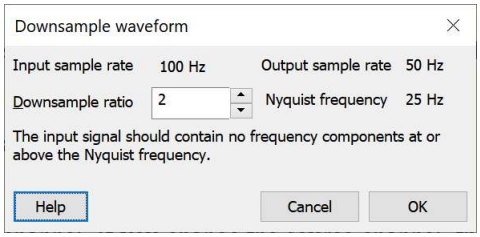
The example, below, shows the dialog with two added processes. The processes are set to low pass filter the input and down-sample the result by a factor of 10. As this changes the channel sample rate, the information line displays the changed rate.



You can edit the IIR filter, Down sample and Activity processes by double-clicking them in the list of processes.

Down sample waveform dialog

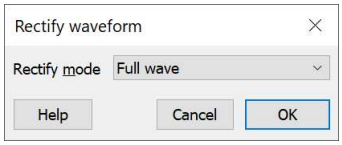
This dialog opens when you Add a down sample process to the channel configuration or double-click the down sample item in the list of real-time processes attached to a channel.



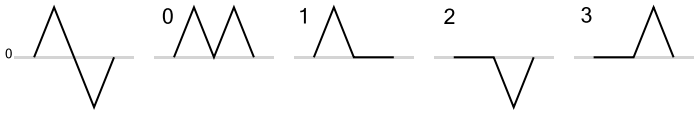
The dialog displays the input and output sampling frequencies and also the Nyquist frequency (highest frequency that can be represented with this sample rate). You can either type in a down sample ratio or use the up and down arrows to change the value.

It is your responsibility to check that there are no higher frequencies in the input than the displayed Nyquist frequency. One convenient way to do this is to include a low pass filter before the down sample process for this channel. You can test the frequencies in the channel by sampling it at a relatively high frequency with no channel processes and running a power spectrum on the channel.

Rectification mode dialog



This dialog sets and edits the rectification mode set for a real-time channel process. There are 4 possible modes:



Mode	Name	Effect
0	Full wave	Replaces negative data with positive data of the same magnitude.
1	Half wave	Replaces negative data with zero values.
2	Negative half wave	Replaces positive data with zero values.
3	Inverted half wave	Inverts the data and replaces negative values with zero values.

You are not allowed to add a rectify process directly after a rectify process as this would make little sense.

The script language equivalent of this dialog is the `SampleProcess(chan%, 1, mode%)` command.

Activity detect dialog

The activity detect dialog configures the methods that convert a sampled waveform to a Marker channel that signals detection of activity and peaks and or troughs. This process changes the basic type of the channel from waveform to a Marker and we allow this only when used with a derived channel. The script language equivalent is the `SampleProcess()` command with a `proc%` argument of 4. This was added at Spike2 [10.13] as an experiment; the details may change based on user experience.

You can use the generated Marker channel to turn data sampling on and off in real time on one or more channels, which can be very useful in a long term recording to reduce the quantity of recorded data.

All methods share the same dialog, which is divided into three sections:

Time constants
All detection methods start with a simple DC removal (high-pass filter) phase with a user-defined time constant (this operates in a similar manner to the DC Remove Channel process). The effect of this is to remove wandering baselines. If you do not want the offset removed, set a time constant of zero (which is treated as a special case). The longer the time constant, the longer the process will take to recover from a sudden change of the input offset.

The second field in this area is either Peak decay (for Activity mode) or Peak width for all Peak/Trough detection. These are described below.

Trigger and off level
All methods use the Trigger level field, which is always set as a positive value, even when used in Trough mode (when it is negated). This is the level that the input signal must cross to transition away from a 'nothing happening' state. The value in this field is in the units of the waveform input to this process.

The Off fraction is the proportion of the trigger level (for Activity mode) or of the signal peak/trough that marks the end of activity or confirms a peak or trough. This must be greater than 0 and less than 1.0.

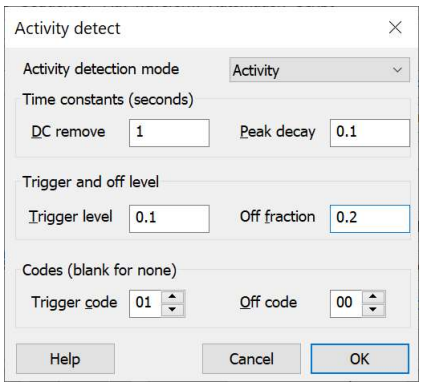
Note that you can use the Channel process DC Remove and Rectify options to visualise the effect of the processing as an aid to setting useful trigger levels.

Codes
The Activity mode can set the Marker codes for the start and end of a detected activity. The peak and trough modes can set Marker codes for peak and trough. Codes are entered as either a single ASCII printing character or as two hexadecimal codes. You can also set the second code (where present) as blank, for no code.

Activity detect mode
This field sets how Spike2 processes the input waveform to generate a Marker channel. You can set it to one of Activity, Peak, Trough, or Peak and Trough.

Activity
This mode detects periods of activity in a signal that is usually quiescent. One application of this is to detect input on a microphone channel to turn recording on and off on the channel to save disk space. Another is to detect activity on a recorded channel, perhaps to trigger a script to run to generate a response. This process can generate two marker codes: the Trigger code when the input signal crosses the Trigger level, and the Off

code when the activity is determined to have ceased. You can set one (but not both) of these codes to an empty field (not to a space character, which is a Marker with the hexadecimal code 20), if you do not need one of the conditions to generate an output.



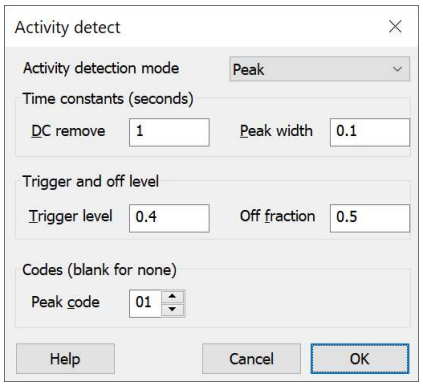
This process works on a rectified version of the input signal after DC removal. It works as a state machine with 3 states:

State	Action
0	Wait for the processed signal to be less than the Trigger level. When it is, set state 1.
1	Wait for the processed signal to cross the Trigger level. When it does, output the Trigger code and record the level as the current Peak, set State 2.
2	Decay the Peak by the Peak decay time constant. If the current processed input exceeds the Peak, set it as the new Peak. If the Peak is less than the Off fraction times the Trigger level, output the Off code and go to state 1, otherwise do nothing.

The Peak decay must be in the range 0.001 to 1000 seconds. This value is related to how long after input activity falls below the Off fraction of the Trigger level the Off code (if set) is transmitted.

Peak

This mode detects signal peaks that are larger than a threshold value and have a width (time from the Peak to the time that the signal crosses a defined fraction of the Peak) that is less than the Peak width time set in the dialog. The single Marker output code will be at the position of the peak, but will not occur (in real time) until the peak is confirmed by the signal crossing the Off fraction of the peak level.



This process works on the input signal (after DC removal). It works as a state machine with 3 states:

State	Action
0	Wait for the signal to be less than the Trigger level. When it is, set state 1.
1	Wait for the signal to cross the Trigger level. When it does, record the level as the current Peak and the time as the Peak time, set State 2.
2	If the current input exceeds the Peak, set it as the new Peak and record the Peak time. If the signal is less than the Off fraction times the Peak level, output the Peak code and go to state 1 or 0 depending on the level. If we are more than the Peak width past the Peak time go to state 1 or 0 depending on the level, otherwise remain in state 2.

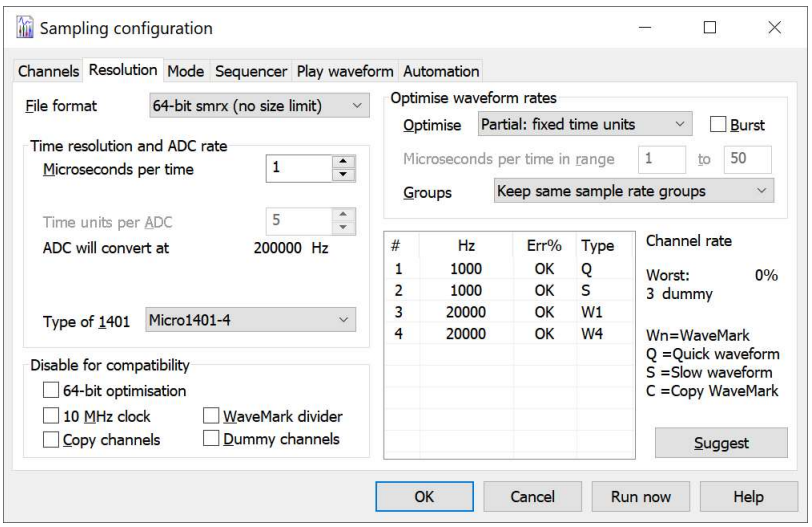
Trough
This is identical to Peak mode except with everything inverted. One oddity is that the trigger level is set as a positive value, but minus this value is used as the level. The Codes field title is Trough code.

Peak and Trough
This behaves as Peak mode if the signal crosses Trigger level or a Trough mode if it crosses -Trigger level. There are two code fields: Peak code and Trough code. If you leave Trough code empty (setting it to the space character is not empty), the trough code used is the Peak code plus 1.

Time resolution

The Resolution page of the sampling configuration dialog sets the time resolution of the document and the ADC sampling rate. To get the best possible results, you should read all this section. However, to get started quickly, follow these "cookbook" instructions:

1. Set the Output file format field (set 64-bit smrx unless your files need to be read by old software).
2. Set the Type of 1401 field to the 1401 type you will use.
3. Set Optimise to Partial: fixed time units.
4. Set Groups to Keep same sample rate groups.
5. Set Microseconds per time unit to the smallest value that gives you the waveform sample rates you need (usually1). If you selected a 32-bit file type in step 1, set Microseconds per time unit so that the Longest run time field is at least as long as the time you want to sample for.
6. Clear all the Disable for compatibility section check boxes (the state of disabled boxes does not matter).
7. If the Burst check box is visible, set it unchecked unless you know that you need burst mode.



This dialog controls how Spike2 optimises the sampling rates. Spike2 minimises the sum of the proportional errors between the desired and the actual rates for the waveform and WaveMark channels. By proportional we mean that an error of 200 Hz in a sampling rate of 10 kHz is the same as an error of 2 Hz in a rate of 100 Hz.

Output file format
Spike2 is optimised to create and work with 64-bit `.smrx` data files. Unless you have a requirement to use 32-bit `.smr` files (for example because you use third-party programs that do not read `.smrx` files to analyse your data or your collaborators are using a version of Spike2 that cannot read `smrx` files) you should set this field to 64-bit. See *Data file types*, below, for full details. The script language equivalent of this field is `SampleBigFile()`.

Type	Name	Comments
64-bit	`*.smrx`	64-bit data files that are limited in size only by the capabilities of the operating system and have no time limitations. This allows both high timing resolution and long run

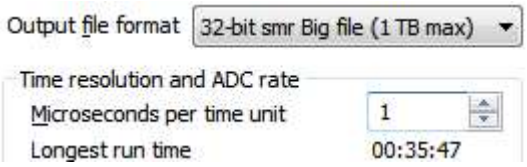
		times. Spike2 version 8 is optimised for this format. Use this unless you need backwards compatibility.
32-bit big	`*.smr`	32-bit data files limited in size to 1TB and in time to 2 billion clock ticks. This is compatible with Spike2 version 7 and can be read by version 6.11 onwards.
32-bit	`*.smr`	32-bit data files limited in size to 2 GB and in time to 2 billion clock ticks. This is the most compatible format for old versions of Spike2.

Microseconds per time unit

This field sets the time units for a new data file in the range 1 to 10000 microseconds. All data of any type stored in the file occurs at a multiple of this time unit. If you have selected a 64-bit file you will want to set this to the lowest value possible (giving the best time resolution) compatible with the desired sampling rates (usually 1). If you have selected a 32-bit file, you will have to balance the desire for timing accuracy against how long you need to sample for, see below. To edit this field, set Optimise to None or Partial. If you select a Power1401 or Micro1401 mk II or -3 in the Type of 1401 field, you can set this field to a resolution of 0.1 instead of 1 microsecond.

Setting the resolution to non-integral microseconds prevents 32-bit files being read by versions of Spike2 prior to 4.02. 64-bit files cannot be read by versions prior to 7.11c.

Longest run time



This field is visible if you selected a 32-bit file in the Output file format field. With a 32-bit file, as in the example to the right, the maximum sample time is 2,147,483,647 ticks of the Microseconds per time unit field and the Longest run time field displays this duration in years, days, hours, minutes and seconds for the current time units. If this period is shorter or longer than you require for your work you can adjust the Microseconds per time unit field to give you the duration you need.

There is a length limit for 64-bit files but it is so large (256 thousand years at 1 microsecond resolution) that it is of no practical concern.

Time units per ADC convert

This sets the ADC (Analogue to Digital Converter) clock interval in the units set by Microseconds per time unit. Each time the ADC is clocked either one sample is taken (non-burst mode) or a group of samples is taken (burst mode). To edit this field set Optimise to None. The ADC will convert at field is the equivalent rate in Hz for normal mode; it changes to ADC burst rate in burst mode.

Type of 1401

Members of the 1401 family have different capabilities. Although you cannot use the 1401*plus* or the original micro1401 to sample, you can still select them in the list in case you need to match previous settings. The choice you make here sets the absolute maximum settings for sampling.

There is no guarantee that the maximum settings are achievable. The maximum rate depends on the entire sampling configuration, the input data load, the speed of the 1401, the speed of the interface connection and on the capabilities of the host computer. For example, if your 1401 is plugged into a USB 1 port (or has a USB 1 interface), the maximum continuous sample rate to the host is 400 kHz (or less), which will limit your aggregate ADC rate to a maximum of 200 kHz.

Regardless of what you set here, when you sample with a 1401, Spike2 will detect the 1401 type and warn you if you have sampling settings that cannot be achieved by the detected 1401.

You can set:

General compatibility

Any 1401 except a 1401*plus* with the old analogue card (an upgraded standard 1401). This is a *lowest common denominator* setting with a maximum waveform sampling rate of 166 kHz and does not assume that your monitor firmware is the most recent. This option is preserved so that you can match old sampling rates with a modern 1401.

1401plus, old ADC (not supported for sampling)
If your 1401*plus* was upgraded from a standard 1401 without an upgrade of the analogue card you must select this option. This option is preserved so that you can match old sampling rates with a modern 1401.

micro1401, 1401plus (not supported for sampling)
This setting is for a micro1401 or a 1401*plus* with an up-to-date monitor. This option is preserved so that you can match old sampling rates with a modern 1401.

Power1401, Power1401 625 kHz
These settings are for the Power1401. You can set Microseconds per time unit in steps of 0.1, and Time units per ADC convert to 1 and sets the maximum ADC convert rate to 400 kHz (625 kHz with the later Power1401 625 kHz). Selecting any Power1401 or the Micro1401 mk II enables additional optimisations when calculating waveform rates (*10 MHz clock*, *WaveMark divider*, *Copy channels* and *Dummy channels*).

Micro1401 mk II, Micro1401-3
Select this option to take advantage of all the capabilities of the Micro1401 mk II and -3. This allows Microseconds per time unit to be set in steps of 0.1, Time units per ADC convert to be set to 1 and sets the maximum ADC convert rate to 500 kHz. All optimizations are available.

Power1401 mk II, Power1401-3, Micro1401-4
This setting allows Microseconds per time unit to be set in steps of 0.1, Time units per ADC convert to be set to 1 and sets the maximum ADC convert rate to 1 MHz. All optimizations are available.

Disable for compatibility
If you replace a 1401 with a more recent model or upgrade Spike2 from before version 6.05 you may get different waveform and WaveMark sampling rates. The new rates will be closer to the requested rates, but if you are half way through a study, compatibility with earlier data may be more important. This section of the dialog disables new features so you can match the original rates. Fields that do not apply to the device selected in the Type of 1401 field are disabled. The settings of disabled fields are ignored. The fields are disabled for the 1401*plus* and the original micro1401 as these devices did not support any of the following features.

64-bit optimisation
Check this box to disable new optimisations added in version 8 that take advantage of 64-bit times. With a 64-bit file, when searching for acceptable Microseconds per time unit values, we take the lowest value that minimises sampling rate errors. With a 32-bit file or with this option selected, we take the highest value.

10 MHz clock
Check this box to force the Microseconds per time unit to be an integral number. Before version 6.05 this control was available as part of the Groups field as *1 MHz, same sample rate groups*.

WaveMark divider
Previously, if you had WaveMark (spike shape) channels, the sampling rate of the WaveMark data set the maximum sample rate for all other waveform channels. Now, we allow waveform channels to be sampled faster than WaveMark channels by taking 1 in every n WaveMark points (dividing the WaveMark rate by n).

Copy channels
If you sample the same channel as both a waveform and as a WaveMark, we sample the data for the WaveMark channel and take a copy for the waveform channel. Check the box to sample each channel separately.

Dummy channels
With older versions of Spike2, it was sometimes possible to get a better match to the desired sampling rates by adding more channels and then ignoring the data in them. For example, if you requested 7 waveforms at 10 kHz each you got 9.99 kHz. Adding another waveform channel produced 10 kHz again. Dummy channels do this for you automatically, and do not waste any time moving data back to the host. Dummy channels are not used (or helpful) in burst mode.

Optimise waveform rates
The Optimise field sets the parameters Spike2 changes to minimise sample-rate error. In full optimise mode, Spike2 chooses the time resolution, the base ADC sample rate, and the best mix of slow, quick, copy and dummy channels. In many situations you will want to specify the time resolution and let Spike2 do the rest.

Sample rate error is defined as (`rate` - `ideal`)/`ideal`, where ideal is the rate you asked for and rate is the rate you got. This is negative if the rate is too slow and positive if it is too high. The optimise command minimises the sum of the absolute value of the errors.

None: use manual settings
You have control over the values in the Microseconds per time unit and Time units per ADC convert fields. In this mode click the Suggest button to change the fields to values that minimise sample rate errors.

Partial: fixed time units
You set the Microseconds per time unit field and Spike2 adjusts the Time units per ADC convert field to minimise the sampling rate errors. If there is more than one solution, Spike2 chooses the one with the slowest ADC convert rate.

Full: time units can change
Spike2 sets the Microseconds per time unit and Time units per ADC convert fields. If all channels have a slow rate, this can take a long time, especially in burst mode. The Microseconds per time range fields set the acceptable range of time units. If there are multiple solutions, Spike2 chooses the one with the largest Microseconds per time unit when writing to a 32-bit `.smr` file (to maximise the run time) and the one with the smallest when writing to a 64-bit `.smrx` file. In a complicated sampling situation with a lot of channels this can take quite a while as there can be a huge number of combinations to check.

Microseconds per time in range
This field is usually set for a range of 1 to 50 microseconds. However, if you are sampling to a 64-bit smrx file you can usually set this to the range 1 to 2 microseconds. This allows a better time resolution and also reduces the time Spike2 spends searching for possible clock settings.

If you upgraded from an old version of Spike2 you may find that the lower figure is set to 2 or more, which, unless this is really what you want, will be a severe constraint.

Groups
This field places additional restrictions on how Spike2 maps the requested sample rates for all the waveform and WaveMark channels into achievable sampling patterns.

Version 3 compatible
This gives the same waveform rates for a given Microseconds per time unit and Time units per ADC convert as version 3. Only use this if you upgrade from version 3 and it is vital that sampling rates match old data files.

Keep same sample rate groups
If you select this option, Spike2 will make sure that all waveforms channels with the same ideal sampling rate have the same actual rate. You will normally use this option.

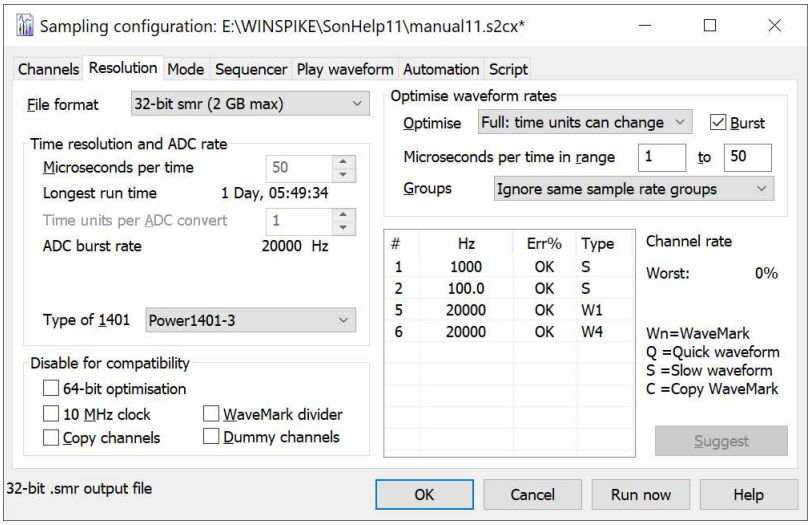
Ignore same sample rate groups
This option gives the smallest sampling rate errors. The price you pay is that channels with the same ideal sampling rates may get actual sampling rates that are different. Some data analyses, such as waveform correlations, multiple channel averages and power spectra demand that channels have identical sampling rates.

1 MHz, same sample rate groups
This setting was present before version 6.05 but no longer exists. You can achieve this now by disabling the 10 MHz clock in the Compatibility section and selecting *Keep same sample rate groups*. If you read an old configuration that used this setting it will be translated to work in the same way.

Burst mode
The Burst check box is visible if you select any type of Power1401 or a Micro1401 mk II, -3 or -4. If you check it, instead of sampling the ADC at equal intervals synchronised to the 1401 clock, it samples a burst of channels at a time, each burst being synchronised to the clock. When you check the Burst mode box, the ADC will convert at field changes to ADC burst rate.

Burst mode sampling can be useful when you sample more than 1 waveform or WaveMark channel. If you have `n` waveform and WaveMark channels, the advantages of burst mode are:

- You may be able to run the Spike2 clock (set by Microseconds per time unit) `n` times slower. This allows longer sample durations if you sample to a 32-bit file; this is not a consideration for 64-bit files (where you should normally set the resolution between 1 and 1.9 microseconds).
- You may be able to achieve sample rates per channel that are `n` times higher for the same Spike2 clock rate or much closer to the requested rate.



In this example, setting burst mode allows us to run for more than 1 day when sampling to a 32-bit file (1 Waveform at 1 kHz, one at 100 Hz, a WaveMark channel with 1 trace at 20 kHz and another with 4 traces). Without burst mode, the time resolution has to be 5 microseconds, which limits the run time to just under 3 hours. Of course, if you target a 64-bit file, you can run at 1 microsecond resolution with no practical time limit.

The disadvantages of burst mode are:

- Waveforms are sampled at times that may fall between Spike2 clock ticks. Channels are sampled at precisely the correct interval, but each channel is shifted sideways by up to half a Spike2 clock tick from when it was sampled. In many applications this will not matter, and the benefits of a better sampling rate may outweigh this. From version 8 onwards, where you can have a very small clock tick, this may not matter at all.
- If you are sampling to a 32-bit file (not recommended), and have used Burst mode to extend the sample time, you will also have reduced the time resolution. In the example, above, the time resolution is 50 Microseconds. You can improve the time resolution at the expense of the run time by adjusting the to field in the Microseconds per time in range. For example, settng it to 10 reduces the maximum run time to under 6 hours.

Burst mode is most effective when you sample to a 32-bit file and have a large number of waveform and WaveMark channels. With a 64-bit file and multiple waveform and WaveMark channels you may be able to get closer to a desired Ideal sampling rate.

If you set the Optimise field to *None*, for full manual control, you will normally set the Time units per ADC convert field to 1 as higher values tend to reduce the benefits of burst mode.

Warning
Checking this box is a suggestion, not an order. In some cases, particularly at low sample rates with small numbers of waveform channels that all are marked as S =Slow waveform, Spike2 will ignore this check box.

Data file types

Spike2 is optimised to use and sample data to the 64-bit SON64 filing system. It can also use and sample data into the original 32-bit SON filing system. We would recommend that you use the new format unless you have pressing reasons for using the old. Not all sampling features are supported by the old filing system and at some point we will remove support for sampling to it. You can read the file format version with the `FileInfo()` script command.

The 32-bit file system (*.smr)

The original Spike2 data format was first released as version 1 around 1988 and the original design has been extended several times since; we are currently at version 9 (the final version). Each revision has added new data types and/or new filing system features. Changes were incremental; older versions of Spike2 would read newer files as long as they did not use new data types or features that the older versions did not know about.

The original design was based on the idea that all items in the file were located in time at an integer multiple of a basic clock period and that there are two fundamental types of data: waveforms (equally spaced in time data samples) and events (items at defined times with other data attached). There is some crossover between these types; waveforms can have gaps and events can have waveforms attached to them. You can download a manual explaining this library plus interface software from our web site.

32-bit original format

There are two limits to the size of a 32-bit Spike2 data file: the number of clock ticks in the file (limited to 2147483647) and the physical size of the file. The physical size before version 7 was limited to 2147483647 bytes, or 2 GB. These limits are related to the maximum size of a signed 32-bit number. For many users these limits are not a problem. However, it is possible to run out of disk space before you run out of clock ticks. For example, if you set the Waveform sample rate to be one clock tick and sample one channel, each sample point uses 2 bytes of data. If you set Burst mode sampling in the Resolution tab with n channels, each taking one sample per clock tick, then each clock tick is using 2n data bytes. This means that your maximum possible run time is limited to 2147483647/2n clock ticks.

Before Spike2 version 7, if you were sampling 32 channels of data at 31.25 kHz each, in non-burst mode you would need to set the clock tick to 1 microsecond, giving a clock tick limit of some 35 minutes of sampling. However, the file size limit would be hit in around 17 minutes. If you changed to burst mode sampling, the clock could be run at 32 microseconds per tick, which extends the clock limit to 19 hours, but the file size limit is still 17 minutes.

32-bit big files (Spike2 version 7 onwards)

If you set the file type to 32-bit big, the maximum `.smr` file size is increased by a factor of 512, and with the example given above (32 channels at 31.25 kHz each in burst mode), you could sample for the full 19 hours allowed by the clock tick period. However, your timing resolution is now 32 microseconds. This would generate a huge file (128 GB) which would be slow to navigate.

32-bit limitations

The 32-bit system served us well, but has limitations due to the 32-bit nature of the format:

1. Times are stored as 32-bit integers, limiting us to a maximum of 2 billion clock ticks per file. At a resolution of 1 μs (microsecond) the maximum duration is 35 minutes and 47 seconds.
2. The file size was originally limited to 2GB. This was later extended to 1TB, but at the cost of making data recovery harder.
3. It can take a long time to locate data in the middle of a file. Finding data was speeded up by including lookup tables, but the methods used were limited by the requirement for backwards compatibility.
4. Waveform data with gaps is not stored efficiently as each gap starts a new disk block, so short waveform fragments use a lot of disk space.
5. There are limits on the number of characters used for channel titles (9), units (5) and comments (71) and file comments (79). Further, these are 8-bit characters, so if you take advantage of character codes outside the ASCII range, these will take two or more UTF-8 characters to store. This can be a problem, particularly for the Units. For example, if you decided to replace `"uVolt"` with `"μVolt"`, and saved this to a 32-bit file, it would be truncated to `"μVol"` as the code for the Unicode character `μ` is U+03BC which is coded in UTF-8 as 0xce, 0xbc.

The 64-bit filing system (*.smrx)

The new 64-bit filing system was designed to be logically compatible with the 32-bit system; by this we mean that it can hold the same types of data as the original without information loss, though some of these types are extended. It is also likely that we will add further data types, as required, in the future. Features include:

1. Times are stored as 64-bit integers. At a time resolution of 1 ns (nanosecond, 10^{-9} seconds), the maximum duration is around 256 years.

2. The file size is limited only by the capabilities of the operating system and by the size of a file that you can manage conveniently for archival. As I write this, disk drives have a maximum size of a few TB.
3. The file format is designed to make recovery of damaged files relatively straightforward.
4. The files have a built-in data lookup system designed to minimise the number of disk reads required to locate data on any channel.
5. The overhead for severely fragmented waveform data has been reduced to a few bytes per fragment.

We have removed many of the limits on things like the number of channels in a file and the length of channels comments and units. Before Spike2 version 8.03 the program enforced the original limits on the length of strings (but treated as Unicode characters). From version 8.03 onwards, a 64-bit `smrx` file in Spike2 can have 20 Unicode characters of channel title, 10 of channel units and comments can be up to 100 characters. The underlying file format does not impose a fixed limit, but practical considerations make it useful to define them.

How older versions of Spike2 cope with 64-bit (.smrx) files

Spike2 version 7.11c onwards can read (but not modify) the new 64-bit file system if the `son64.dll` file is in the Spike2 version 7 folder. Of course, if the file is longer (in clock ticks) than the old file system can read, only the start of the file will be visible. No other older versions of Spike2 can read them.

How older versions of Spike2 cope with 32-bit (.smr) files

Spike2 version 8 can read files with more than 400 channels, but ignores channels 401 upwards. Spike2 5.15 onwards can read files with up to 400 channels. Versions from 5.00 to 5.14 read files with up to 256 channels. Version 4.03 onwards reads files with up to 100 data channels. Versions before 4.03 can read files with up to 32 channels. You can use the File menu Export command to write channels from a data file to a new file with a suitable maximum channel limit so that it can be read by older versions of Spike2. If you check the Big file box you will not be able to open any generated file with versions of Spike2 before 6.11. Spike2 version 6 revisions that can open the file will treat it as read only.

Technical details of 1401 sampling

There is a master clock in the 1401. The Microseconds per time unit field sets the tick period of this clock. All times in a Spike2 data file are multiples of this time unit.

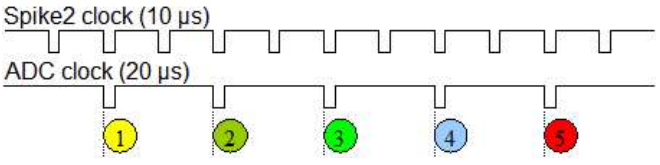
The Analogue to Digital Converter (ADC) samples one input at a time and is shared between all the waveform, RealWave and WaveMark (spike shape) input channels sampled by the 1401. The table below lists the maximum sampling rate in multi-channel mode for the ADCs in each 1401 when used in Spike2 and also if the 1401 supports Burst mode sampling. Items marked * are not supported in Spike2 version 11 but are preserved for reference. Using a slow interface (for example USB 1.0) will reduce the maximum rates.

1401 type	Maximum rate	Can burst
1401plus*	166 kHz	No
micro1401*	166 kHz	No
Power1401	400 kHz	Yes
Micro1401 mk II/-3	500 kHz	Yes
Power1401 625	667 kHz	Yes
Power1401 mk II	1 MHz	Yes
Power1401-3/3A	1 MHz	Yes
Micro1401-4	1 MHz	Yes

To illustrate the difference between non-burst mode and burst mode, we will consider what happens when we set Microseconds per time unit to 10 and Time units per ADC convert to 2 with five waveform channels in both non-burst mode and in burst mode. In the diagrams, the numbered circles represent the ADC sampling each channel and the horizontal position of the circle represents the time at which the sample occurs.

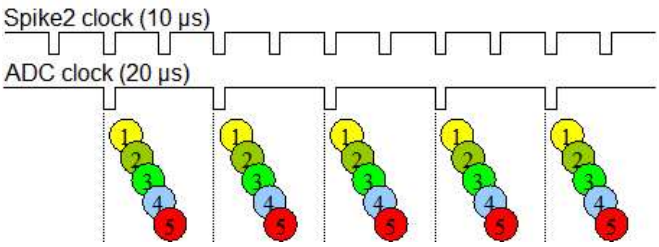
Non-burst mode

In non-burst mode, the Time units per ADC convert field sets how often the ADC samples in units of clock ticks. For example, with 10 microseconds per tick (but see *Improvements at version 8*, below) and the Time units per ADC convert field set to 2 (once every 20



microseconds), the ADC sample rate is 50 kHz. In a simple case with five waveform channels, the fastest each channel could be sampled is 10 kHz. Each sample is synchronised to the Spike2 clock and the pattern repeats every five samples.

Burst mode



In burst mode, the Time units per ADC convert field sets how often a burst of ADC samples is taken. With the same settings of 10 microseconds per tick and the Time units per ADC convert field set to 2, the burst rate is 50 kHz. In a simple case with five waveform channels, five channels would be sampled in a burst, each channel sampled at 50 kHz. The interval between samples in a burst depends on the 1401 hardware and is typically the interval implied by the maximum sampling rate for the 1401 hardware.

As you can see, the ADC samples no longer fall exactly at Spike2 clock times. The times between samples on a channel are exact, but the entire channel may be displayed time shifted by up to half a Spike2 clock as it is timed to the nearest tick. In this case, channels 1,2, 3 and 4 would probably be timed for the tick just before them, and channel 5 for the next tick.

Cycle of channels

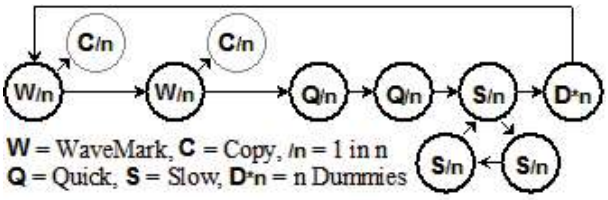
To generate the sample rates, the ADC samples a cycle of channels. In burst mode all the channels in a cycle are sampled in a burst, in non-burst mode they are sampled one at a time. WaveMark channels are sampled once every time around the cycle. Some waveform channels are set as *Quick* and are also sampled every time round the cycle. The other waveform channels are set as *Slow*, and these share one position in the cycle. Quick and Slow channels save every n^{th} data point (n in the divisor in the range 1 to 2147483647). The Version 3 compatibility setting in Groups sets all waveforms as Slow channels and the maximum divisor to 65535.

With a Micro1401 mk II or -3 or any Power1401, there are further features (unless Version 3 compatibility has been set):

WaveMark divider This enables down-sampling (taking 1 point in n) of WaveMark data. This allows you to sample Waveform channels faster than WaveMark channels.

Copy channel If you sample a waveform channel on the same 1401 port as a WaveMark channel, we can use the same data twice, once for the WaveMark channel and once for the waveform. These *Copy* channels behave exactly like a Quick channel, but are more efficient.

Dummy channel Adding additional channels to the sampling loop sometimes gives a more accurate approximation to the desired sample rates. Adding a Quick channel would waste time transferring unused data to the host; a *Dummy* channel just throws the data away. Spike2 adds dummy channels automatically if they improve the channel sampling rates. Dummy channels are not added in burst mode.



The diagram shows a possible cycle for two WaveMark channels and seven waveform channels of which 2 are the same as the WaveMark channels. Each time around the main loop, the 1401 samples all WaveMark, Quick and Dummy channels and one Slow channel. When waveform and WaveMark channels are sampled together, the fastest waveform rate is the same as the WaveMark sample rate.

Spike2 searches all ADC rates allowed by the Optimise setting and the Type of 1401 field for the best combination of Quick, Slow, Copy and Dummy channels and the best value of n for each channel to get as close as possible to the ideal sample rates. With slow waveform rates there can be millions of combinations to search. If the Channels Tab feels very sluggish, set Optimise to None, set the channels, then restore the Optimise value. Impossible combinations display a warning in the lower left corner of the dialog.

If you check boxes in the *Disable for compatibility* section, this stops Spike2 taking advantage of optimisations that are not available for all 1401s. You might want to do this if you upgraded your 1401 and discovered that the sampling rates with your new 1401 were not the same as with the old one. Of course, the new rates would be closer to what you had asked for, but it might be important that they matched the old rates exactly.

The table to the left of the Suggest button lists the channels in order of descending sample rate error, showing the actual sample rate, the error as a percentage of the desired rate or OK if there is no error, and the channel type (WaveMark, Copy, Quick or Slow) and the number of Dummy channels.

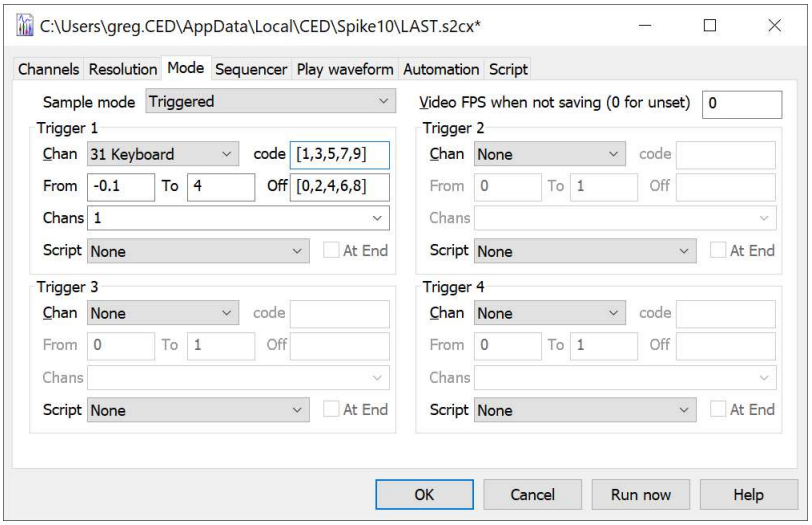
Improvements at version 8

Version 8 of Spike2 allowed you to use 64-bit data files in place of the 32-bit files used in all previous versions. For 32-bit files, the maximum sample time is 2,147,483,647 clock ticks (2×10^9). At 2 microseconds per tick this is 71½ minutes, at 10 this is almost 6 hours, and at 1000 this is nearly 25 days. For 64-bit files, the maximum sample time is 8,070,450,532,247,928,832 clock ticks (8×10^{18}). This is so large that we need not worry about running into the limit. Several of the limitations in sampling in previous versions of Spike2 were caused by the requirement to maximise the Microseconds per time unit field to give long run times; selecting a 64-bit file format removes these limitations.

In the examples above, the Spike2 clock is shown as running at 10 us per tick, which was a commonly-used tick rate with the old 32-bit time file system as it allows a run time of around 6 hours per data file, but at the cost of restricting the ADC sample rate, or forcing the use of burst mode. If you choose a 64-bit data file, you can usually run the Spike2 clock at 1 microsecond per tick, with a much wider choice of sample rates and no limitations on run time.

Sampling mode

The Mode page of the sampling configuration dialog determines when data captured by Spike2 is saved to the disk system. In all modes, time passes at a constant rate, even when nothing is written to disk. When reviewed, areas of a file with no saved data are empty. There are three sampling modes: Continuous, Timed and Triggered. The script language equivalent of this dialog is the `SampleMode()` command.



Whichever sampling mode you select, the Sample control bar Write check box lets you disable data saving to the data document on all channels while sampling. Script users can enable and disable data saving channel by channel with the `SampleWrite()` command. If the Sample control bar or the `SampleWrite()` script command disables writing on a channel, nothing is written on that channel regardless of the sample mode set in this dialog. Once the channel is enabled again, the sample mode then determines when data is saved to disk.

Right-click (Context) menu

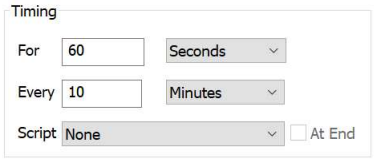
Within this dialog, you can right-click to open a Context menu with options to copy the sample mode settings as text to he clipboard or to the Log view. If a script is defined, right-click within the Trigger or Timer group box for an option to open the script in a Script window.

Continuous mode

The simplest sampling mode is Continuous mode, which records data continuously unless disabled by the Sample control bar or by a script.

Timed mode

In Timed data capture mode, data is saved at intervals. You set the period for which data is saved and how often to save the data. The times at which the blocks were requested are saved in the keyboard marker channel. Marker code 00 is placed at the start of each timed block and marker code 01 is placed at the end of each block. If you use a 32-bit output file you should read the *64-bit and 32-bit file differences* section, below.



From version [10.09] you can also nominate a sampling script to run either at the start or at the end of each sampling period. See the Script tab for sampling script limitations. If a script is selected, right-click to open it in a script window.

Triggered mode

In Triggered capture there are four triggers. Each has a channel list (that can be empty) and an optional script. A trigger is an event or marker that causes data to be marked for writing to disk in a time range relative to the trigger and can run a nominated script (as long as no other script is running). Data Triggers are additive, that is if a new trigger occurs while data is being written due to a previous trigger, the trigger period is extended. Script triggers cause a script run request for every event. There is also an optional Off event that stops the disk writing before then end of the time range.

In triggered mode, any channel that is not associated with a trigger is recorded continuously. Each trigger has the following fields:

Source — This should be set to None to disable the trigger or you can select a channel from the drop down list. You can trigger on any event, Marker or WaveMark channel in the sampling configuration. Before Spike2 version [9.01] you could not trigger on a channel generated by a Talker. You can select only channels that exist in the current sampling configuration.

Code — If the trigger source is a Marker, WaveMark, RealMark or TextMark channel, you can choose to trigger only if the marker code matches the marker mask specification defined by this field. Set the field empty to match all codes. Codes `00`, `01` and `02` in the keyboard marker channel are special and will not cause triggers.

From — This value is the offset from the trigger to the start of the area to record, in seconds. Negative values define pre-trigger start times, positive values start recording after the trigger. If you set too long a pre-trigger time, the data may have been discarded before the trigger is seen. Spike2 normally attempts to use an 8 MB data buffer, so this is not often a problem. To save time, we do not redraw data before the trigger if the From time is negative; we cancel the Not saving to disk colour from the time the trigger was seen, not before.

To — This value is the offset from the trigger to the end of the area to record, in seconds. It must be greater than the time in the From field.

Off — Enabled if a marker mask specification is set in the Code field. If a Trigger channel marker code matches the marker mask specification defined by this field, sampling is disabled from the marker time on the Channel field channels and any script in At End mode will run at the next opportunity. This was added at version [10.13].

Channels — You can either select All Channels from the drop down list, or type in a channel list, for example `1..4,6,8` for channels 1, 2, 3, 4, 6 and 8. These are the channels that will be written to disk each time the trigger event is detected. To set no channels set the field empty or select No channel. No check is made in this field that the channels you type exist; non-existent channels are ignored at sample time.

Script — From version [10.09] you can select a sampling script to run each time the trigger event occurs. See the Script tab for sampling script limitations. The drop down list remembers the last 10 used trigger scripts. If a script is selected, right-click to open it in a script window. There are no guarantees on when a script will run after the trigger event. On a lightly loaded system with no other Spike2 activity (other than sampling), delays of order ten milliseconds or so are usual. However, if the Edit menu Preferences Scheduler sets a long delay between Idle cycles, or if other actions are taking place, delays can be much longer. You must test this in your sampling situation.

At End — This check box is enabled when a script is set and the To time is greater than 0. Checking this box causes the script to run at the To time past the trigger rather than at the trigger time.

Running a script At End means that triggered data from a 1401 will be available to the script. This is not guaranteed to be the case for data from a Talker (Talker data can arrive delayed, the delay depends on the Talker).

Triggered sampling is often used with fast waveform or WaveMark channels to save disk space in situations where only small sections of the data are interesting. However, if you use a 32-bit output file you should read the *64-bit and 32-bit file differences* section, below.

Video FPS when not saving (0 for unset) [10.06]

If you use the s2video application to capture video, from version [10.06] you can set this field non-zero to switch into the Slow frame rate mode when data is not written to disk. Not writing to disk happens automatically in Timed mode and when you pause writing to disk on all channels. We take no action in triggered mode (though we may add this in the future).

The standard state, with this feature disabled, is to set the field to 0. To enable this feature, set the desired slow frame rate. For example, set 1 for 1 frame per seconds, 0.1 for one frame every 10 seconds.

Special keyboard trigger features

If the user changes the state of the Write check box in the Sample control toolbar, a marker code is written to the Keyboard channel to indicate that writing to disk is enabled (code `00`) or disabled (code `01`). If a change is made to a signal conditioner, code `02` and an explanation is written to the TextMark channel, but if this is not present, code `02` is written to the keyboard channel. If the keyboard channel is used as a trigger, writing codes `00`, `01` or `02` does not trigger sampling.

Display during data capture

In all data capture modes, the on-line display shows newly sampled data, even when you are not saving to disk. Recent data is saved in a memory buffer in these modes so the current data is always available. However, if you scroll far enough back into an unsaved time region, the display will become blank as the memory buffer has a limited size.

64-bit and 32-bit file differences

If you use triggered or timed mode with the 64-bit filing system, triggered or timed channels will save only the data implied by the timing or the trigger, as you would expect. Note that Level event channels are always saved in their entirety, regardless of the saving state, to ensure we track the levels correctly. This may be revised in a future revision by adding code to make sure that an even number of edges are dropped.

However, the 32-bit system can only save complete blocks of waveform data and each block is around 16000 data samples. If you select a 32-bit file output file, you will always save at least the data implied by the trigger or timing, but with waveform channels you will usually get more data than asked for. Further, if your triggers or timing periods are closer together than 16000 data points, you will save continuous data.

Sequencer

The Sequencer page of the Sampling configuration dialog sets the output sequence to use during sampling. The sequence can either be a `*.pls` text sequence file stored on disk or it can be a graphical sequence created from this dialog. From version [10.15] it can also be a text sequence stored as part of the sampling configuration.

The main control is the Output selection drop down list. The option available are:

None

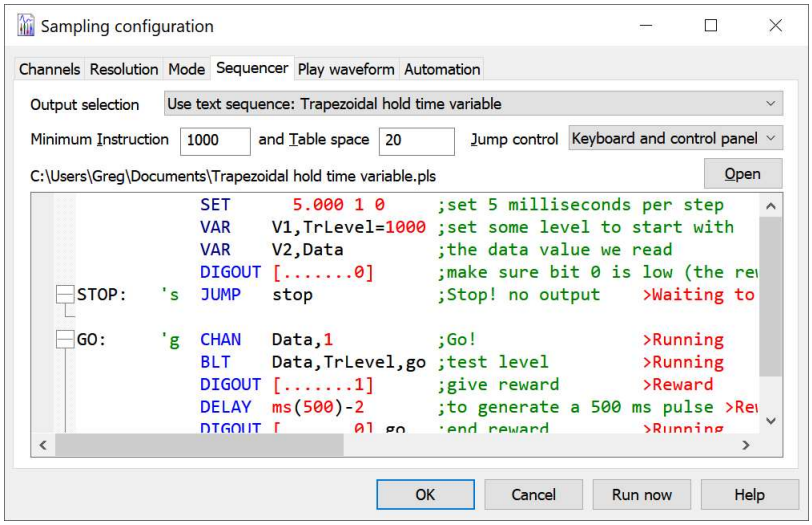
There is no output sequencer used in this sampling configuration.

Select a text sequence file

Opens a file selection dialog in which you can choose a text output sequence file (`*.pls`) that will be used when you sample. These files are created with the output sequence text editor or by exporting graphical editor sequences as text.

Use graphical sequence editor
This reconfigures the dialog to display the graphical sequence options.

Use text sequence: filename
This option is present when a sequence file is selected for use. The dialog displays the first few lines of the sequence (if it can be read) and you can click the Open button or double-click the displayed sequence to open it in a pulse editor view. If you want to edit the sequence you will have to close this dialog; alternatively you can hold down `Ctrl` and click Open or double-click the sequence to close the dialog and open the sequencer for editing.



Save text sequence 'filename' as part of configuration
This option is present when a text sequence file is selected for use. If you choose this option, a copy of the file is stored in the sampling configuration as text and the selected option will change to *Use text sequence in sampling configuration*. Storing the sequence as part of the configuration means that you only have the configuration file to keep track of rather than it and an output sequence file. However, using sequencer text stored in the configuration is usually used by script writers who generate sequences dynamically as it saves the need to write them to a file. This feature was added at version [10.15].

Use text sequence in sampling configuration
The display will show the first few lines of the sequence stored in the sampling configuration. If you double-click the displayed sequence or click the Open button, it will be written to the file `S2CFGSEQ$.PLS` in the current user data folder. You must close the dialog to modify the sequence. If you use the Current button in the output sequence editor view and the file name is set to `S2CFGSEQ$.PLS`, this will update the sequence stored in the configuration. If the sequencer text is empty and you close the dialog, the Output selection will revert to *None*. This feature was added at version [10.15].

Recent: filename
The list of options ends with a list of recently-used output sequence file names that you can select.

Minimum Instruction and Table space
Spike2 reserves space in the 1401 for the instructions and table space used by the sequence set in the sampling configuration. However, if you load new sequences during sampling or edit the graphical sequence during sampling, the replacement sequences may need more space. These two fields reserve extra space; the space set is the larger of that required for the initial sequence and the values you set in these fields. Unless you intend to replace the initial sequence with a text sequence during sampling you should set both these fields to 0. The script equivalent of these fields is `SampleSeqCtrl()`.

Jump control
The sequencer can be commanded to jump to specific places by keyboard commands, the Sequencer control panel , the Create TextMark dialog, the serial TextMark input and always by a script. You can limit the interactive options to prevent accidental jumps. Choose from:

	Keyboard	Control panel	Script
Keyboard and sequencer control panel	Yes	Yes	Yes
Sequencer control panel	No	Yes	Yes
Script commands only	No	No	Yes

Lower portion of the dialog

If a text sequence file is selected it is displayed in the lower portion of the dialog. To modify an output sequence file you must open it; the easiest way to do this is to double-click in the displayed file. You must close the Sampling Configuration dialog before you can edit the sequence.

If you select the graphical sequence editor, the lower portion of the dialog displays graphical sequencer control settings.

Changing the sequence during sampling

If you have selected the graphical sequence editor, you can change the editor settings and apply them during sampling by using the Sampling menu Graphical Sequence Editor... command. Each time you apply the settings, the sequence restarts with the new settings.

You can also use the Sampling menu Change Output Sequence command to apply a different Text Sequence during sampling.

In both cases, your new sequence must use the same, or fewer resources than the original sequence, or you must have set a Minimum Instruction and Table space, as described above.

Arbitrary waveform output

You can replay arbitrary waveforms during data capture. Up to 62 (20 with Micro2 and Micro3) different "Play wave" areas can be defined for output. Each area outputs waveforms on 1 to 4 DACs and is identified by a key code (in the range `Chr$(3)` to `Chr$(127)`), typically a printing character such as "A". No two areas may have the same code. You do not have to use a printing character, you can use a two digit hexadecimal code if you prefer, however codes `00`, `01` and `02` are not allowed, nor are codes above `7f`. The format of this code is the same as for marker codes. Because you can trigger waveform output by recording keyboard markers with this code you should make sure that your code usage is compatible with key codes used in the output sequencer. We suggest that you confine yourself to alphanumeric characters (upper and lower case A-Z and 0-9).

From [11.00] you can also trigger waveform areas with TextMark data, either from the Create TextMark dialog or from serial line TextMark data.

The waveforms are played from 1401 memory and are copied there just before sampling starts (when the Time window to display sampled data is created). This reduces the memory available for recording data. The maximum data you can store in the 1401 for replay is the free space in the 1401 less 256 KB which is the minimum Spike2 reserves for recording. The maximum size of a waveform area is 32 MB (assuming your 1401 has sufficient memory). Script users can update the memory dynamically during replay, so huge memories are not necessarily required!

The table summarises the capabilities of arbitrary waveform output for different 1401 types.

1401 Type	Micro2	Micro3	Micro4	Power1	Power2/3	Power3A
Wave space	0.7/1.7 MB	3.7 MB	31 MB	32 MB	32 MB	32 MB
Areas	20	20	62	62	62	62
Fastest rate	167 kHz	250 kHz	500 kHz	500 kHz	500 kHz	500 kHz
Slowest rate	0.00024 Hz	0.00024 Hz	0.048 Hz	0.00024 Hz	0.00024 Hz	0.00024 Hz
Resolution	0.1 µs	0.1 µs	0.05 µs	0.1/0.2 µs	0.1 µs	0.05 µs
Best 44.1 kHz	44.053 kHz	44.053 kHz	44.101 kHz	44.247 kHz	44.053 kHz	44.101 kHz

Wave space — The maximum memory space you can use to stored arbitrary waveform data in the 1401. Using the maximum space may make high-speed data capture more prone to overrun for Micro1401s as it leaves only 256 kB for data buffering. The 0.7 Micro2 figure is for 1MB of main memory, the 1.7 MB is for 2 MB of memory.

Areas	The maximum number of separate play wave areas you can define. The sum of the sizes of these areas must fit in the Wave space.
Fastest rate	Measured replaying two waveforms while sampling one at the same rate. If you sample or replay more channels or use the output sequencer, the maximum rate may be lower.
Slowest rate	The slowest the 1401 clock hardware can be programmed to run; around one sample every 20 seconds for the Micro4 and one sample every 400 seconds for all the other 1401 types.
Resolution	The accuracy that the output waveform clock can be set to.
Best 44.1 kHz	The closest we can get to one of the standard audio output rates due to the output clock tick resolution.

The final sample of the waveform area sets the output level after waveform output ends, so it is usually a good idea to make sure that the waveform output ends with a zero value. Note that the `VDAC0-VDAC7` sequencer variables are not updated by arbitrary waveform output.

Frequency resolution

The 1401 DAC output rate is derived by dividing down from a fixed frequency clock. The fixed frequency can be chosen from 1, 4 or 10 MHz for all 1401s and the Micro4 and Power3A also support 200 MHz. The division is by the product of two integers, both in the range 1 to 65535, except for the Power1 which uses the range 2 to 65535). Although you can request any rate, the actual rate will be the closest that the 1401 hardware can get to the requested rate.

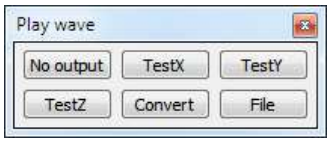
The table shows the timing resolution for output rates over 1 Hz (for slower rates, the resolution can drop as low as 1 microsecond). These limitations can be significant, particularly if you replay imported data. For example, 44.1 and 48 kHz (often used in `.WAV` files for sound recording) cannot be represented exactly. The table shows the closest each type of 1401 can get to 44.1 kHz. This is calculated as the clock rate divided by two divisors. The divisors are in the range 1-65535 for all but the Power1401-1 which is limited to the range 2-65535.

For example, if your 1401 has a sample rate resolution of 0.1 microseconds, and you wanted to achieve 44.1 KHz, the divisor from 10 MHz (the internal clock rate corresponding to 0.1 microseconds) is 10000000/44100 which is 226.76 to two decimal places. The closest divisor we can get is 227, so instead of 44.1 kHz you would get 44.053 kHz, an error of approximately 0.1%. In the case of the Power1401-1, as 227 is prime, the closest we can get is 226 (2 x 113), hence the larger error. The Micro4 and Power3A have a 200 MHz clock available, and they can get much closer to the desired rate.

You can use the `ChanSave()` script command to re-sample waveform data to a different rate using cubic spline interpolation if the sample rate error is significant for your application. During data capture you can use the `PlayWaveSpeed()` script command to read back the actual output rate.

You should never have any problem matching sampling rates when replaying data that was recorded with your 1401.

Play waveform toolbar



There is a dockable toolbar associated with the waveform output. This is enabled when you are sampling data with waveforms defined. The toolbar can be docked on any edge of the application and can be resized when it is floating. You can assign your own labels to the buttons, or you can let Spike2 generate labels itself of the form Wave 0, Wave 1 and so on. The first button in the toolbar is used to stop a currently playing waveform.

The order of the buttons in this toolbar is the order of the waveforms in the Sampling configuration Play Waveform tab. From [10.17a] you can change the order in the configuration by dragging the items.

From [10.03] you can disable this panel from starting waveforms from the Sampling configuration Play waveform Tab and also with the `PlayWaveCtrl()` script command.

From [10.20], if you hover the mouse pointer over a label, any comment for that item is displayed. If no comment is set, the tool tip displays the key for the area and the source of the waveforms.

Channels and DACs

Each play wave area contains from 1 to 4 data channels. You can select the 1401 DAC (Digital to Analogue Converter) each channel plays through. The maximum DAC number you can set varies with the type of 1401.

All Power1401 models allow DACs 0-7 and all variants of the Micro1401 allow DACs 0-1. It is not an error with a Power1401 to select DACs 4-7 when they are not present; of course, there is no output.

With multiple channels, all channels play at the same rate and all DACs update together. You can play data from waveform and WaveMark channels in a Spike2 data file, or data generated by a script. When data comes from a Spike2 file, the channels need not all have the same sample rate; Spike2 takes the rate of the first channel and interpolates data from the subsequent channels to make the rates the same. Script users can play arbitrarily long data by updating the waveforms in the 1401 online with the `PlayWaveCopy()` command.

The rate at which a wave plays can be modified in the range 4 times slower to 4 times faster than normal (as long as your hardware can output fast/slow enough). From a script, you can change the rate during sampling, even while the wave is playing.

Waveform storage

Waveforms for output can be stored in the sampling configuration several ways:

Reference to channels in a data file

This is efficient, as it uses very little space in the configuration file. It has the disadvantages that the data file must exist in the same location when sampling starts and that if you refer to a memory buffer channel or a Virtual channel, the data file must be open in Spike2 when recording is initiated.

Stored in the configuration

You can convert a file reference to the output waveform and save it in the sampling configuration. This can be convenient for short waveform fragments, but can become very bulky with large ones, particularly as configuration files store data as XML (basically as text).

As a promise that the script will fulfill

You can reserve space for output waveforms, expecting that a script will be running during data capture that will set the data for output using the `PlayWaveCopy()` command.

Waveform scaling

The 1401 DACs play 16-bit waveform data. The source of DAC data can be a Waveform, WaveMark or RealWave data channel, or can be set from the script language with the `PlayWave...()` family of commands.

Waveform or WaveMark channel

When the source of a waveform is a Waveform or a WaveMark channel, this data is already stored as 16-bit integer data. The source channel values are copies, as is, to the DACs. The current channel scale and offset values make no difference. If you want to scale the result, one way to do this is to make a copy of the channel as a RealWave channel (either as a channel saved to disk or as a memory buffer channel or as a Virtual channel). To convert a channel from Waveform to RealWave, the first step is usually to make a Virtual channel using the Waveform channel as a source, then Save the channel to disk or import it into a Memory buffer.

RealWave channel

When the source of a waveform is a RealWave channel, this is stored as 32-bit floating point values. We use the channel scale and offset to convert these values to 16-bit values to play from the DACs. If the result of this conversion exceeds 16-bit range, the values are limited to 16-bit range (-32768 to 32767). Because of this conversion, you can scale and shift the data by changing the source channel Scale and Offset values.

Imported data

If you import data (often from a text file) into Spike2 and want to replay it, it is easy to get confused by the scaling. If you import text data as a Waveform, the importer scans the data to find the range, then maps this into the 16-bit data range (from +32767 to -32768) so that the entire range of the data is represented and allowing for a little headroom. The channel scale and offset are set so that the displayed data matches the input values. If you select this channel for output, it will replay at whatever size it was saved at and the maximum amplitude will lie somewhere between half and full scale.

If you import text data as a RealWave channel, the data is saved as the values read from the channel. The scale and offset are set so that if the signal were converted to 16-bit data, it would use the full 16-bit range (with the proviso that if the signal crosses zero, the offset is set to zero). If you play this signal, the DAC output will likely go to positive or negative full scale (or both). However, by changing the scale (and offset) you can make the DAC output span a lesser range.

A concrete example may make this easier to understand. The standard range of the 1401 DACs is from minus 5 Volts to just under 5 Volts. Let us imagine you have imported a text file that holds a waveform with the desired output values, in Volts. Let us suppose that these values are in the range -1 to +1 Volts. If you import this channel as a RealWave channel, it will import and display the data in the range -1 to +1 Volts, as you would expect. However, the channel scale factor will be 0.2 and the offset 0. These are set to give the best possible (most precise) representation of the data if the channel were converted to a Waveform. However, if the channel were output, this scaling would cause it to span the entire range of the DAC. To make the DAC output span the range -1 to 1 Volt you must set the channel scale to 1.0.

If you imported this channel as a Waveform, the channel is already stored as 16-bit data. If you play this channel, the output will span between half and all the DAC range. To make the replayed waveform match the input you must convert the channel to a RealWave channel (as described above) then set the scale factor to 1.

Setting waves from a script

You can use `PlayWaveCopy()` to move a real or integer data array to an area for output. If you use an integer array, values in the range -32768 to 32767 are written to the DAC (spanning the full range). Values outside this range are limited to -32768 or 32767. If you use a real array, we assume that the DAC output range spans -5 to just less than 5 Volts and map the values accordingly, limiting values outside the range -5 to 5 to the limits. If your 1401 is set to a 10 Volt range the mapping is the unchanged, the output in Volts is doubled.

Playing waves

There are several ways to initiate output of a wave during data sampling:

- Click the associated button in the Play waveform toolbar
- Record the associated key code or key set by `PlayWaveKey2$()` in the Keyboard marker channel
- Use the script language `SampleKey()` command to record the key code
- Use the output sequencer `WAVEGO` command

Unless you use the output sequencer to start the output, the associated key is recorded in the Keyboard channel and marks the time at which output was requested.

When a wave starts to play, there is a time delay of one waveform output between the moment that output is requested and the first data point appearing. This is not usually important, but at a slow replay rate, a one sample delay could be significant. This delay can be useful if you are using the output sequencer as it gives the output sequencer the opportunity to know about a change in the DAC outputs *before* it happens.

Triggered output

Each wave you play can be marked as “Triggered”. In triggered mode, when the waveform play is requested, output does not start immediately. Instead, the 1401 hardware waits for a trigger input (high to low edge) on the Trigger input for the Micro1401 and Power1401 unless this is routed to the rear panel Event connector pin 4 (Ground is pins 9-15) by the Edit menu Preferences. In triggered mode, the first data point is output at the time of the trigger.

If you use the output sequencer you choose between triggered and non-triggered in the `WAVEGO` instruction. You can also trigger the wave from the sequencer with the `WAVEST` instruction and detect if the wave has started to play with the `WAVEBR` instruction.

Repeated plays and links

Each wave can be set to play cyclically for a set number of times. You can also link waves together if they have the same DAC output list. Linked waves play at the rate of the first wave, that is the sample rate of subsequent waves is ignored.

For example, you might have a sound output that needs to ramp up, stay at a constant level, then ramp down. This could be done with three waves. The first holds the ramp up waveform, the second which repeats cyclically many times would hold the constant sound, and the final part would ramp down.

Scripts and the output sequencer can command a wave with many cycles to finish the current cycle then continue to the next linked area. You could use this to simulate a blood pressure signal with an occasional errant beat by having two waves, one with a normal beat set to repeat many times and one with the errant beat set to play once. By linking the two areas to each other, you get one errant beat after a fixed number of normal ones. Further, by using the randomization functions in the output sequencer you could produce errant beats randomly and mark the times at which they occurred.

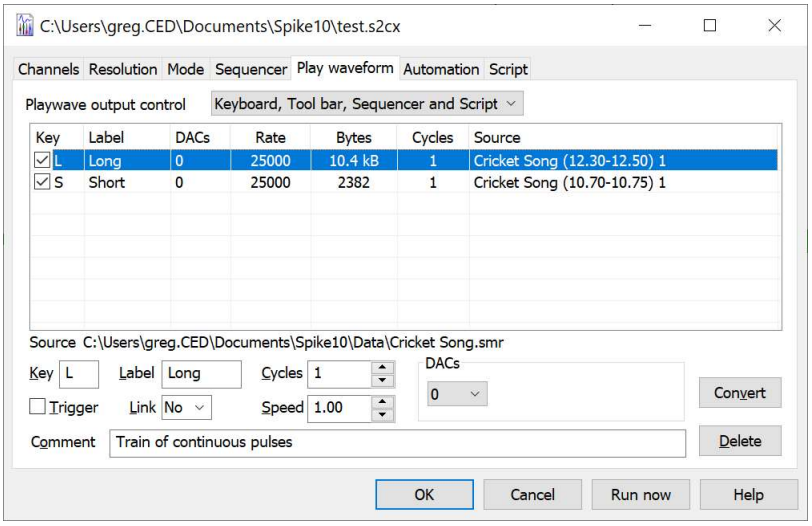
Script-only features

The `PlayWaveCopy()` command allows you to modify the contents of an area while sampling is in progress.

`PlayWavePoints()` allows you to change the size of an area dynamically (but sizes can only be less than or the same as the original size). This lets you allocate an area, then use it to play back waveforms of different lengths. The `PlayWaveKey2$()` command lets you associate a second key with an area. This can be used to play the area, just as the standard key code does, but this code can be changed dynamically, enabling separate key codes for different waveforms played from one area.

Play waveform

The Sampling configuration Play waveform tab lists waveforms for on-line output. Add waves with the Sample menu Offline waveform output dialog Add to Online button and by scripts. The settings are used the next time you sample. Spike2 moves waveforms to the 1401 when the Sampling document is created; this can take a noticeable time if there is a lot of data. Script users can update waveforms during data capture.



The order of the waveforms in the list determines the order of the buttons in the Play waveform toolbar that can be used during sampling to command the waves to play. From Spike2 version [10.17a] you can change the order by clicking on a waveform and dragging to a different position in the list.

Playwave output control (new in [10.03])

This field controls which methods can be used to select the current wave for output. There are 4 possible ways to command a wave to play:

1. By a keyboard key press when the sampling time view has the input focus or from [11.00] from the Create TextMark dialog or from TextMark data from a serial line.
2. By clicking in the Play waveform tool bar to select a wave.
3. By the SampleKey() script command of from [11.00] by the `SampleText()` script command
4. By an output sequencer command (graphical or text).

We presume that any action by a script or sequencer is always allowed, but you may want to prevent keypresses, or even inadvertent clicks on a toolbar from disturbing the waveform output. The standard state before [10.03] was to allow all methods. You can now disable the keyboard, or the keyboard and the play wave toolbar. The script language equivalent of this field is `PlayWaveCtrl()`. Note that triggers from the TextMark dialog or from serial line TextMark data have separate enables and are not affected by this control.

Modify selected wave

Editable fields for the currently selected wave are at the bottom of the dialog:

Key — Wave identifier. A character or two hexadecimal digits in marker code format; you may not repeat a code or use 00. It is added to the Keyboard marker channel on a Play wave toolbar button click.

If a wave is checked in the list, space is reserved for it in the 1401 and if a waveform is defined, it is copied to the 1401 before sampling starts. It can take a noticeable time to copy many megabytes.

Label — Up to 7 characters to label buttons in the Play wave toolbar. If you use `&`, the next letter is underlined and you can use it as a short-cut to the key when the toolbar has the input focus.

DACs — A list of the Digital to Analogue Converters to play your waveform out of. You can change the list with the drop down lists inside the group. You cannot set two channels to use the same DAC output.

Rate — The number of samples to output per second per channel, set when the wave is added to the list. You can vary the replay rate with the Speed control in the range 0.25 to 4.00. If the speed control is set to any value other than 1.00 the Rate field shows the multiplying factor as well as the rate. The graphical sequence editor takes account of Speed when displaying waves, but does not (currently) adjust the wave duration to allow for it.

Size — This is the number of bytes of 1401 memory needed to hold the wave.

Cycles — The times to play the wave. Use 0 for a very large number (about 4 billion). The graphical sequence editor assumes Cycles is 1 when displaying waves.

Source — Where the waveform (if supplied) is stored. Below, Name is a data file name, sTime and eTime are the start and end times of the data and chans is the list of channels to read the wave from.

Name (sTime-eTime) chans: a wave in a data file. The full path appears in the Source field. If the data file is open in Spike2 the channels can be any channel, including Virtual channels, memory channels, duplicate channels and channels with attached channel processes. If the file is not open in Spike2, only disk-based, permanent channels can be used.

Wave from Name (sTime-eTime) chans: a wave originally read from a file, held in memory and saved in the sampling configuration. The full path to the file appears in the Source field. You can release the memory and revert to reading data from the file with the Convert button.

Wave from script: a script generated wave, held in memory, saved in the sampling configuration.

Place holder for script: a script has reserved space. The wave will be generated on line by a script.

Trigger — Check this box for waveform output that is enabled by play requests and that starts on the Trigger input. If this is not checked, a play request starts the waveform playing immediately.

Link — You can link waves with identical DAC channel lists together. Linked areas play in order with no time gap between them at the rate set by the first wave played. The graphical sequence editor does not allow for links when displaying waves.

Comment — If you set a comment, it is used as the tool tip for the corresponding button in the Play wave bar. The comment an be read and set with the `PlayWaveComment$()` script command.

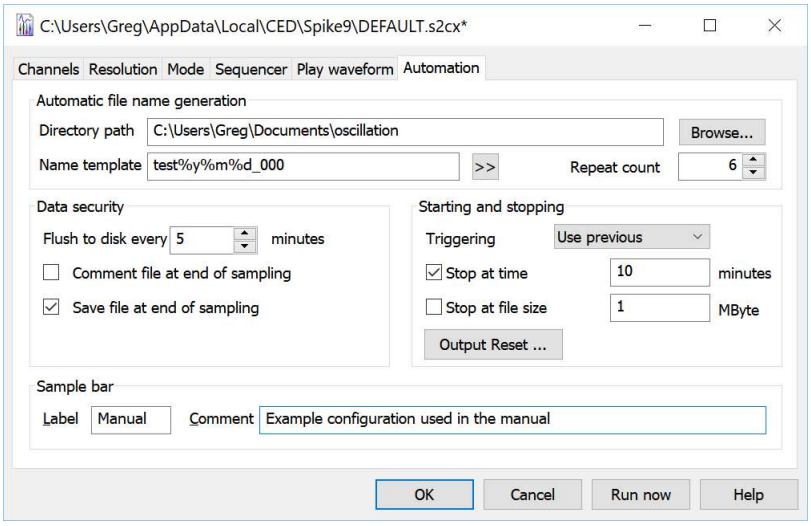
Convert — This button changes data held in a file into data held in memory and vice-versa. If this option fails because the channels numbers refer to a virtual or duplicate channel, you may be able to make it work by opening the file within Spike2 to create the missing channels.

Update — Applies any changes you have made to the current wave.

Delete — Delete the current wave.

Automation

The sampling configuration dialog Automation page sets the file path and name for automatic data filing, sets the security parameters for new data files and restricts the total sampling time and the size of the data file. It also contains the label and comment used when you add the sampling configuration to the Sample bar.



Automatic file name generation

Spike2 samples data to temporary files in the folder set by the Edit menu Preferences. Unless you set a Name template, these files have names like `Data1`, `Data2` with no extension. When sampling ends, you save the file to a name of your choice. You can automate file naming by setting a Name template of up to 40 characters (Spike2 versions before [9.01] limited this to 23 characters) or using the `SampleAutoName$()` script command, and Spike2 will generate a sequence of names from it. A blank Name template disables automatic name generation. The file extension is determined by the Output file format set in the Resolution page and should not be set here.

If the template does not end in a number, `000` is added before using it. To make a file name, Spike2 increments the number until a name is formed that is not in use in the Directory path folder (which *must* exist and cannot be blank).

A template of `test` generates `test000` to `test999`. A template of `test10` generates `test10` to `test99`. The Directory path must be less than 200 characters long. If the folders set by Directory path and in the Edit menu Preferences option are in the same disk volume, sampled files are renamed rather than copied, which can save a lot of time, especially if you are using automatic sampling on a sequence of files.

The script equivalent of Directory path is the `FilePathSet()` command with `opt%` set to 3.

With a file name template set, the generated name is used when the data file is saved. A different name can be specified using the File Save As… command. If there are no free names, or the path set for file saving does not exist or is blank, **you are not allowed to close the dialog until you fix the problem** or use Cancel.

Illegal file name characters

You may not use any of the following characters in the file name template: `< > : " / \ | ? *`

Date and time specifiers

Within the file name template, the character sequence `%` followed by one of `y`, `m`, `d`, `H`, `M` or `S` is converted into a two digit representation of the year, month, day, hours, minutes or seconds. `%Y` is replaced by the year as 4 digits. `%D` is shorthand for `%y%m%d` and `%T` is shorthand for `%H%M%S` (we set the digits this way so that sorted names are in time order). If the name ends with a date or time specifier, Spike2 adds `_` to the name so that a date or time is not incremented when searching for an unused file name. Of course, as the name does then not end in a number, `000` is also added. For example, if you set the Name template to `test%D` and you use it on December 19, 2013, the name expands to `test131219_000`. The >> button to the right of the field lets you insert date and time specifiers at the caret or to replace the selection in the Name template field. You may not use the slash (`/`) character to separate date fields or the colon (`:`) character to separate time fields; these are not allowed in file names (see the *Illegal file name characters*, above).

Automatic sampling of a sequence of files

If you set a Name template and a Directory path, check the Save file at end of sampling box and one or both of the Stop at time or file size boxes, you can enable automatic sampling of a sequence of files by setting the Repeat count field. Values of 0 or 1 sample a single file, larger numbers sample a file sequence.

Sampling each file starts and stops based on the conditions in the Starting and stopping box. For each file except the last, when sampling stops the file is closed, and all result and XY views created by processing from it are closed. Sampling resumes with the next file in the sequence. The file sequence stops when:

- The number of files set in the Repeat count have been sampled
- There is no free disk space
- The name template cannot be incremented or there are no unused names
- There is an error during sampling
- The user clicks on Stop or Abort in the sampling control panel.

Data security

For efficient sampling and to allow us to retrospectively choose to save data, Spike2 buffers several megabytes of the most recently sampled data in memory and writes data when the buffers are full. The operating system then buffers this data and writes it to disk whenever it chooses. So, if the power failed, unwritten data would be lost.

The Flush to disk every n minutes field sets how often the data buffered in memory is forced out to the physical disk to guarantee that your data is safe. There is a significant time penalty for doing this (can be seconds) and it interacts with turning write to disk on and off. If you want the fastest possible sample rate you should turn this feature off (by setting a zero period). However, with it on, even if the computer power is lost or the computer crashes, your data should be safe up to at least the last flush time; you will be prompted to save the interrupted file the next time you start Spike2. You must run `S64Fix` (for 64-bit files) or `SonFix` (for 32-bit files) on such a data file to complete the data recovery and to tidy up data blocks written after the last flush.

From version [10.07] onwards you can set this field to -1 to request *Write through* mode. This requests that all writes to the file to be written to disk immediately, so should be slower. However, if you are writing a sequence of files at a high sample rate, this can greatly reduce the time gap between the end of one file and the start of the next as this limits the quantity of unwritten data when the file closes to the size of the internal Spike2 data buffer (used for retrospective write to disk decisions, currently 8 MB) rather than to all the data written since the last flush. However, it means that in the case of a power failure, up to 8 MB of data could be lost (which in the case of a sparse event channel could be all the data).

If the shortest interval between repeated files is important, you should test the difference between using the values -1, 0 and 1 for the Flush to disk every field.

If the Comment file at end of sampling box is checked, you will be prompted to provide a file comment when sampling finishes. This is disabled when sampling a sequence of files as prompting for a comment would interrupt the sequence.

If the Save file at end of sampling box is checked, the new data file is saved to disk automatically when sampling finishes. If automatic file name generation is in use, the generated file name is used, otherwise the usual prompts for a file name are provided. This box must be checked if you want to sample a sequence of files automatically.

Starting and Stopping

When you sample with a 1401 interface, you have the options of starting to sample on a digital pulse on the front panel Trigger input, or the rear panel E3 event input. The Triggering field allows you to preset the state of the sampling control toolbar Trigger check box:

Use previous	Uses the current state of the Trigger check box
Not Triggered	Forces sampling to start immediately; clears the Trigger check box
Triggered	The first sampled file is triggered, any repeats are untriggered
All triggered	The first file and all repeats are triggered

Stop sampling

You can cause sampling to stop automatically at a set run time, or when the data file is a set size. If you do not check a box, the associated limit is not used. In general, you will get a little more data than implied by an active time or data limit as data buffered in the 1401 at the time the limit was reached will be added to the file. To sample a sequence of files automatically you must check at least one of Stop at time or Stop at file size.

Output Reset...

You can choose to have the DAC and digital outputs set to known values when a sampling configuration is loaded, and more usefully just before sampling starts and after sampling ends. Click this button to open the

Output Reset dialog. Values set from here are stored in the sampling configuration and override application settings set from the Edit menu Preferences in the Sampling tab.

Sample Bar

The Automation page also holds a label of up to 8 characters and a comment of up to 80 characters for the Sample bar. When configurations are saved to a `.s2cx` file, they can be added to the Sample bar by the Sample menu Sample Bar List… command and any label or comment is used to provide information about the configuration.

If you include an ampersand (&) in the label, the following character can be used with the `Alt` key as a short-cut to load and run the configuration. See the Script bar documentation for details and caveats.

Once a configuration is in the Sample bar, you can open a new data file with a mouse click or `Alt`+key.

Output Reset

The normal reset state of a 1401 is all DAC outputs at 0 Volts and all digital outputs set low. Spike2 leaves all 1401 outputs alone except when sampling. However, in some situations it can be important that the 1401 outputs are returned to a known state after sampling in case sampling was interrupted; this is where you would use the Output Reset dialog.

There are two sets of Output Reset information. One is associated with the Spike2 application, the other with the sampling configuration. The effect of the two sets of information is as if the applications settings were applied followed by the sampling configuration settings.

Not all features are implemented. The Ramp DACs feature is a future enhancement and is likely to require hardware changes to allow it to be implemented, so will be for specific 1401 types only.

Feature	Application	Sampling Configuration
Opened from	Edit menu Preferences Sampling tab	Sampling configuration Automation tab
Dialog title	Application Output Reset	Output Reset
Where is data saved	In the registry	In the sampling configuration
Apply on load	When the program starts	When sampling configurations load
Priority	Values may be over-ridden	Values are always used
Use for	State you always need applied	State dependent on the sampling configuration

The image shows the dialog when opened from the Sampling Configuration: Automation tab. When opened from the Edit menu Preferences: Sampling tab the dialog title changes to Application Output Reset and the Apply when sampling configuration is loaded check box becomes Apply when application starts.

DAC values

Spike2 supports up to 8 DACs per 1401 interface. You can set the reset value (in Volts at the 1401 output) for each DAC. Only DACs that are checked (and exist in your interface) will be reset. The normal reset level is 0 Volts, but you can set any value within the range of the DACs.

Dig Out bits

This field sets the reset values for bits 15-8 (from left to right) of the digital outputs as 8 characters. Each character can be 0, 1 or `x` standing for low, high and no change.

Dig Low bits

This field sets the reset value for bits 7-0 (from left to right) of the digital outputs as 8 characters. Each character can be 0, 1 or `x` standing for low, high and no change.

Ramp DACs to desired voltage over ... seconds (if possible)

Currently this field is inoperative and the DACs change to the reset value in one step. When implemented, the DACs will ramp from their current value to the reset value over the time period specified.

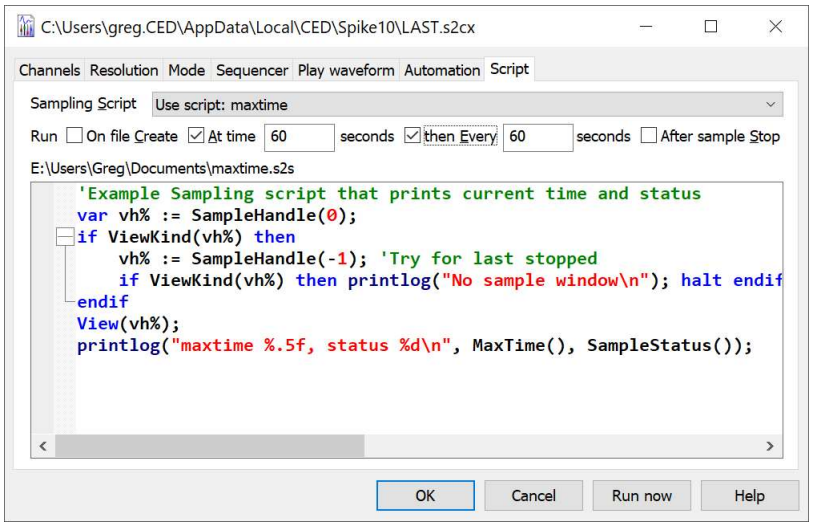
Apply...
If you do not check any of the Apply... boxes, this dialog has no effect. You can choose when to apply the reset state: when the Output Reset information is loaded, before sampling starts, after sampling ends, or at any combination of these times.

Script

The Sampling configuration Script tab (new at [10.09]) allows you to associate a script file with the sampling configuration. This sampling script runs at user-defined times during sampling. This script will typically run for a short time and end. Only one script can run at a time, so if another script is running, the script will not run. A typical use of such a script would be to save sections of a large data file periodically to allow other programs to analyse data while the data is sampled. Another use would be to run a particular stimulation or analysis protocol during a long sampling session.

An advantage of running a script periodically rather than having a script that runs all the time and allows idle time for the user to interact with the data is that between script runs, there is no limitation on user actions. The user can even run other scripts as long as they do not prevent the Sample Script from running. A disadvantage is that any information that must be carried between script runs must be saved separately, usually using the `Profile()` script command or in a file.

You can also set scripts associated with Timed and Triggered sampling from the Sampling configuration Mode tab.



If a script is selected, it is displayed in the large panel that occupies most of the display. If you double-click in this panel, this will open the script in a separate window. Above this is a line of check boxes where you can choose when the script runs.

Sampling Script
If you drop this list down, you have a choice of None, Select script file, the current script (if any) and up to 10 previously used scripts.

None
No script is used.

Select script file
This opens up a file selection dialog in which you can choose a script to add to the sampling configuration. It is entirely up to you what this script does; it has full control over the system. If the script runs for any appreciable time, it should release some idle time, otherwise the display of sampled data will not update.

Use script: ScriptName
This has no effect as it selects the current script.

Recent file: Name
There can be up to 10 of these entries. These are scripts that you have used previously. The list of used script is saved in the registry.

Run

This section determines when the script should run. For a script to run, Spike2 must have idle time and a script must not be running already. Note that it can take a while for a script to run after the request is made, especially if Spike2 is busy with other tasks. When the script runs you can get a handle to the active/first view of the sampling file with the `SampleHandle()` script command. You can use the `SampleStatus()` script command to get the sampling state. Scripts can be set to run at various times by checking the boxes:

On file Create
The request to run the script is made when the file opens and before it starts to run. If this is part of a repeated sequence of files, this is only called for the first file of the sequence. When a file is in the created state, but sampling has not yet started, the `MaxTime()` value is returned as one clock tick less than zero and `SampleStatus()` returns 0.

At time
You can choose a time into sampling, in seconds, at which to request that the script runs. You can set any time that is in the range 0.0 to 1,000,000,000.0 seconds (a bit under 30 years).

then Every
Check the box to run multiple times. The Every field is how often to attempt to run the script after the first time. The shortest interval we currently allow is 1 second. Each time a request is made to run the script, the time for the next run is calculated as the next multiple of the Every time after the At time. If the script takes longer than the Every time to run and the script releases no idle time, it is possible to hang up, much as a script that loops forever with no idle time. You can use the `Esc` key to break out of the script.

After sample Stop
This requests that the script runs when sampling stops (including after each file in a repeated sequence). If no file is sampling, `SampleHandle(0)` will return 0, but you can use `SampleHandle(-1)` to get the handle of the last sampled file, as long as it has not been closed.

Sampling scripts

From Spike2 version [10.09] you can nominate scripts to run at specific times during sampling from both the Sampling configuration Mode tab and from the Script tab. There are three ways to set this up:

Mode tab Timed sampling
In timed sampling mode you can nominate a script that is run at the start of each timed data capture, or at the end of each timed data capture.

Mode tab Triggered sampling
In triggered sampling mode, each trigger can have an associated script that runs at the trigger time, or at the end of the associated data capture (if it is at a later time than the trigger).

Script tab
The Script tab can set a script that runs at user-selected times: each time a data file opens, when sampling stops, at a particular time during sampling and at given intervals after that time. This script can run in addition to the timed or triggered scripts.

Although it is possible to have multiple scripts set, there is some timing advantage to running with a single sampling script. This is because Spike2 'caches' the last compiled script, so if you have a single script, there is no need to compile it each time it runs. This saves the compilation time, which though usually a fraction of a second, can be significant if you are running a script in response to a fairly rapid event.

Script limitations

Sampling scripts are no different from any other script, so there are no limitations on what they can do. However, to be useful, they need to be written carefully so as not to mess up the sampling. Most sampling scripts will take very little time to run. They will perform whatever action is required, then exit.

Unless they are set to run when the sampling file opens and before sampling starts, or at the end of sampling, they usually have no user interface. This is particularly true for scripts that run frequently, such as triggered scripts or scripts run by a timer.

Useful script functions

Each time a sampling script runs, it starts with a clean slate of script variables. Scripts run in the context of the user interface thread of execution but may be requested to run from the sampling thread of execution (for example when triggered on an event). Script running will be delayed by the time needed to complete outstanding Windows messages and ongoing drawing operations and compilation (if required) of the script. In my test cases (with relatively slow sample rates and a pre-compiled script), scripts run around 20 milliseconds from the request. The worst case delays can be considerably more and scripts will not run at all if another script is running and releases idle time (`Yield()`, `Toolbar()`, `Interact()`, `DlgShow()`...).

You need to locate the sampling file and know the current state of sampling. Remember that time has passed since the event that caused the script to run. In the worst case, the data file may not exist. The following routines are useful:

`SampleStatus()`
This lets you know the current state of the sampling system. If you set a script to run when the data file opens, and `SampleStatus()` returns 0, you can use `SampleStart()` to begin sampling. If sampling is shutting down, you may want to wait for it to have stopped (if you wait in a loop, remember to call `Yield()` inside the loop to release time to allow the system to close down the file).

`SampleHandle(0)`
This locates the current sampling window, which you will usually require. Note that when sampling stops, there will no longer be a sampling file, but you can use `SampleHandle(-1)` to get the handle of the last sampled file as long as it has not been closed. In the case of sampling an automated sequence of files, each file in the sequence keeps the same sample handle. This also requires care as although the file handle remains the same, the file it refers to can change.

`SampleRepeats(-1)`
If you automate sampling of a sequence of files you can use this command with a negative argument to get information about the repeats done and remaining and the current file number. The automatic sequencing requires idle time to work. Some care needs taking with file sequencing as if the user opens a Modal dialog (a dialog that requires closing before the program will continue), the automatic file sequencing will stop until the dialog closes. Similarly, running a script that does not release idle time will cause automatic sequencing to pause at the end of a file until idle time becomes available.

`Profile()`
This command can be useful when you need to save information about a program state between runs. Larger amounts of data can be saved in text or binary files.

Example Script tab script

The following script is suitable to be set in the Script tab with the At time and Every fields checked and set to at least 10 seconds. Each time the script runs it saves the last `tSave` (set to 10 in the example) seconds of data to a file. If the file already has a name (for the case of a set of file names defined in the Automation tab), the saved name is formed by adding a sequence number to the file name. Otherwise, we use a sequence of files called `0.smrx`, `1.smrx` etc in the folder set for Spike2 sampled data files (`FilePath$(1)`), and if that is not set, in the folder for Spike2 data (`FilePath$(-4)`).

```
'$ExportLast|Script to save recent data to a file as a Script tab timed operation
const first := SampleScript(-2);    'Time in seconds at which to save first
const every := SampleScript(-3);    'Interval to save at
const tSave := 10;                  'Length of data to save (< first and every)
if (tSave > first) or (tSave > every) then PrintLog("bad tSave\n"); halt endif;

if (SampleStatus() <> 2) then halt endif;   'must be sampling
var sh% := SampleHandle(0);         'Get the handle of the file
```

```
if sh% <= 0 then halt endif;        'If none, then give up
View(sh%);
var tNow := MaxTime();              'Where we have reached
var n% := (tNow - first) / every;   'file number of save file
if n% < 0 then halt endif;          'Calculate file index
var tSaveEnd := first + n% * every; 'End of save

'Set output file name
var fName$ := FileName$(-3);        'Get current file path (if set)
if Len(fName$) = 0 then fName$ := FilePath$(1) endif; 'data folder
if Len(fName$) = 0 then fName$ := FilePath$(-4) endif; 'user data folder
fName$ += Str$(n%) + ".smrx";       'Build output file name

ExportChanList(1);                  'Export each file at time 0
ExportChanList(tSaveEnd - tSave, tSaveEnd, -1); 'Save all channels
var ok% := FileSaveAs(fName$, 0, 1, "");   'Export to match source
if ok% < 0 then
    PrintLog("Save failed to: %s\n%s\n", fName$, Error$(ok%));
else
    PrintLog("%3d %6.0f %s\n", n%, tSaveEnd - tSave, fName$);
endif
```

Data buffering

When sampling, Spike2 keeps several megabytes of the most recently sampled data in memory. Having buffered data allows you to run with "write to disk" disabled, yet still allowing you to see recent data and giving you the possibility of retrospectively deciding if data is worth saving or not. How this is done is different between the old 32-bit files and the newer 64-bit files. In both cases, buffer space is allocated to channels in proportion to the expected data rates for each channel (as set, by you, in the sampling configuration). For waveform channels, the expected rate is the sample rate multiplied by the size of each sample. For Event, Marker and extended Marker data (TextMark, RealMark, WaveMark), the expected data rate is the Maximum event rate field multiplied by the size of each item. It is important that you set this field to a realistic peak event rate over say 10 seconds; setting the peak instantaneous rate will result in inefficient allocation of resources.

Having buffers allows us to allow retrospective changes to what data should be saved. Changes are additive; that is you can have saving logically disabled then mark regions for saving with the `SampleWrite()` script command or triggered or timed sampling.

The details of buffering depend on which output file format was selected in the Resolution tab of the Sampling configuration dialog.

64-bit (.smrx) file buffering

During sampling, each channel is allocated a circular buffer to hold the most recent data. The channel also has a save/no save state and list of times at which the save/no save state changes. When the circular buffer becomes full, the oldest data is ejected from the buffer. If it is not marked for saving, it is lost. If it is marked for saving, the data is added to the channel write buffer (a 64 kB buffer that is an image of a disk block for the channel). When the channel write buffer becomes full, it is written to the disk. If you use the `Modified(0,x)` command or set automatic flush to disk, it has the effect of forcing all data marked for saving to the channel write buffer (which causes it to be written if it becomes full) and then if the channel write buffer is partially full, the partial buffer is also written. Times before the last saved data time are removed from the save/no save list (as once data is committed to the channel write buffer you can no longer change the save/no save state for earlier data). However, any data in the circular buffer that is not marked for saving is preserved.

64-bit data saving is sample accurate. Only data marked for writing is saved.

32-bit (.smr) file buffering

Buffer space is allocated in terms of complete disk blocks, as written to the data file. These blocks are typically a multiple of 4096 bytes up to a maximum size of 32768 bytes. Each channel has a list of buffers it can use, and these are filled in sequence. Each buffer can be marked for saving or not saving. When all buffers become full and more data arrives, the oldest block is written to disk if it is marked for saving, then that block is used for

new data. If any data in a block is marked for saving, the entire block is saved. If you use the `Modified(0,x)` script command or set automatic flush to disk, and it causes any data to be saved, any older blocks that were marked as not saving are emptied.

32-bit data saving is by data block; if any data in a block is marked for writing, all data in the block is written.

Flushing data to disk

On modern operating systems, when an application writes data to a disk file, this will not usually write data to the physical storage medium immediately. For performance reasons, data is written into a complex buffering system which is designed to keep the entire system responsive. Should the application crash, the operating system will close down open files and the files will be intact. However, if the system fails (due to power loss or a system crash), all the buffered data in the system does not get written, which can cause significant data loss.

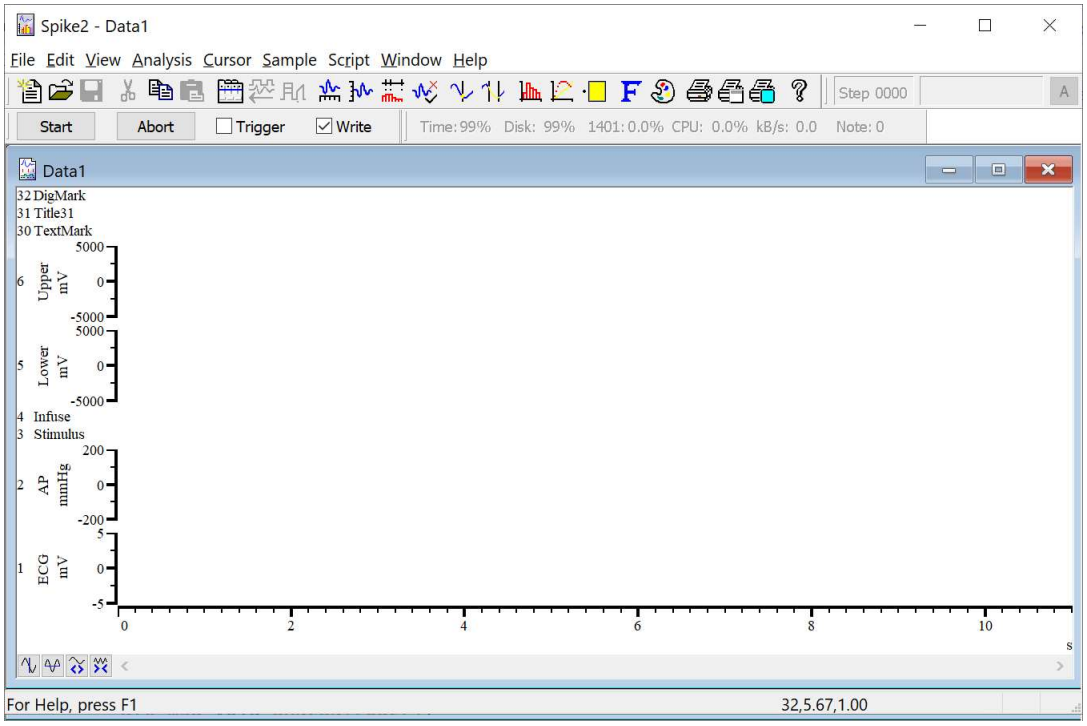
To work around this, Spike2 supports an automatic flush to disk option or you can use the `Modified(0,1)` script command to request that the file data is transferred to the physical disk.

This used to work well, but more modern disk systems now include huge memory buffers internally. It turns out that if you have a "primitive" disk system, when you ask the operating system to flush the buffers for a file through to the disk, this works very quickly as the system "knows" which buffers belong to the file and need to be written to which physical disk sectors. Unfortunately, when the disk itself buffers vast quantities of data, the disk has no idea which buffers refer to which files. The only way to guarantee that the data for a file is on disk is command the disk to write ALL buffered data. We had one case of a user who upgraded to a new, vastly faster computer and found that the first disk flush took 15 seconds when their previous, ancient machine did it instantly.

If you hit a problem like this, one way around it is to make sure that your disk has an uninterruptible power supply and then use `Modified(0,0)` at suitable moments to get data through to the operating system without triggering the flush to disk. Even if your system should fail, the disk should be able to complete the physical writes. You could consider using `Modified(0,1)` before you start sampling in case there is a lot of pending data.

Opening a new document

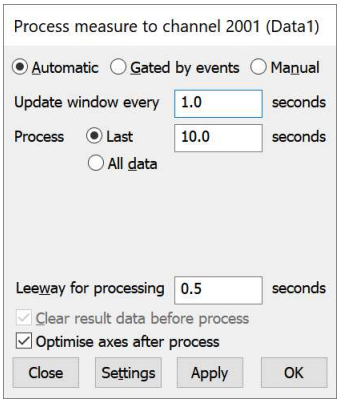
Once you have set the sampling configuration you can open a new data document. Select New from the File menu, then Data Document for the file type. Data documents differ from all other Spike2 documents as they are always stored on disk. Other document types are kept in memory until you save them. We keep data documents on disk because they can be very large. When you save a new data document after sampling, Spike2 moves it to the disk volume and directory you specify. When you use the File menu New command, Spike2 creates a temporary file in the directory specified in the Edit menu Preferences. If you do not specify a directory in the preferences, the location of the temporary file is system dependent.



The exact appearance varies, depending on the configuration. Sampling begins when you click Start in the Sample control toolbar (the Sample menu duplicates the controls in this window). If the Trigger box is checked, sampling waits for an external signal.

You can set the display and analyses required before sampling. For example, to set an interval histogram you can select that analysis exactly as you did in the *Getting started* chapter. There is a difference, however. When you click on New, a new dialog appears.

Process dialog for a new file



You create result views with the Analysis menu New Result View command in the same way as when working off-line. However, the Process dialog, which controls when and how to update the result window, has additional options to work with a data file that grows in length. The radio buttons select Automatic, Gated by events or Manual updates (see the *Analysis menu* for a full description).

The standard on-line mode is Automatic. This mode adds new data to the result at a user-defined interval. You can choose to accumulate a result for all the data, or produce a result for the last few seconds. The other two modes are for specialised uses and should not be used unless you are certain they are what you want.

This dialog disappears once you select either OK or Cancel, however you can recall it with the Process command in the Analysis menu.

Sample control toolbar

The Sample control toolbar holds several buttons and a check box and controls the data sampling process. You can dock this bar to any edge of the Spike2 application window or leave it floating. The toolbar becomes visible (if it was invisible) whenever sampling starts unless the start command comes from the script language or the Edit Preference to prevent this is set. The position of the toolbar is saved in the sampling configuration. However, if it is visible and docked, we do not reposition it as we assume it is where the user wants it.

Initial display

Normally, when you create a new file to sample, the sampling process holds waiting for you to click the Start button, or for the script `SampleStart()` command. The Start text moves from side to side to remind you that sampling is waiting for you. The Trigger field controls whether sampling starts immediately when you click Start or if it waits for a low-going TTL compatible trigger pulse. If you have selected the *Sample and Play wave trigger on rear events* option in the Edit menu Preferences, the Trigger text is replaced by E3 Trig to remind the user that the rear panel trigger is selected. The Triggering field in the Automation tab of the sampling configuration dialog sets the initial Trigger state. If you are sampling without a 1401, the Trigger check box is disabled and the text is replaced by (Trigger) or (E3 Trig) as triggered start requires a 1401 interface. If you use the Sample bar to select between sampling configurations, you can skip this initial display by setting the Immediate start option.

Display after Start with trigger enabled

If you click Start with Trigger (or E3 Trig) checked, Trigger (or E3 Trig) flashes in place of the Start button and sampling holds at time 0 until a suitable signal is applied to the Trigger input. Both the front panel Trigger and rear panel E3 input (Events connector pin 4, pins 9-15 are grounds) expect a low-going TTL compatible pulse of at least 1 microsecond duration; connecting the Trigger or E3 input to ground will work. The Trigger (or E3 Trig) check box is replaced by a Reset button Use this method to synchronize the start of sampling with an external event. Sampling starts within 1 or 2 microseconds of the external signal. Currently, using a triggered start requires a 1401 interface.

Display during sampling

Once sampling has started, the Start button is replaced by a Stop button and the Trigger (or E3 Trig) check box is replaced by Reset. You can use the Write check box to decide which portions of your data are to be saved on disk, and which portions are of only transitory interest; you can read more about the Write control, below. The controls are duplicated in the Sample menu; however the Sample control toolbar needs fewer mouse clicks.

Button actions

Start — This is displayed before data capture starts. Click the button to start sampling. If Trigger is checked, Spike2 waits for the trigger before continuing.

Stop — This is displayed while data is sampled. Click this button to stop sampling and keep the data. If no data was saved, the empty file is discarded. If you attempt to close the file after stopping you will be prompted to save it unless the file is less than 5 seconds long.

Abort — This button is used to abandon sampling and discard the new file. You can use this button before sampling starts, or while sampling is in progress. If you use this interactively, you will be asked if you are sure you want to lose sampled data if any data has been captured and you have been sampling for at least 5 seconds.

Reset — This button appears when you click Start. It stops sampling, discards any saved data, and waits for you to start sampling again with the same document. You are warned that you will lose data if you have captured any data and have sampled for at least 5 seconds.

Write check box and Not saving

The Write check box is normally checked, to save data to the data document. If you clear this box, no data is saved until it is checked again. If you are running in Triggered or Timed sample modes, clearing this check box will also disable writing for these modes (even if a trigger or time interval occurs).

Spike2 keeps a certain amount of data in buffers in memory, so it can display recent data on screen even if you have decided not to save it to disk. This saved data is used in triggered mode to save pre-trigger data and can also be marked for saving with the `SampleWrite()` script command.

You can write enable and disable individual channels from the `SampleWrite()` script language command. This command can also mark data regions for saving when the channel save state is set to not saving. If you use this command to set a state where some channels are enabled and some are disabled, the check box is drawn in an intermediate state (neither checked nor clear).

Whenever you change the state of the check box or use the script command to enable or disable all channels, a marker is added to the keyboard channel (code 00 when writing is enabled and code 01 when it is disabled).

Whenever any channels are not saving, the Write text moves from side to side to alert you and new data for those channels (apart from WaveMark data and Sonograms) draws in the Not saving to disk colour as a warning that data is not being saved.

Save data file

When you stop sampling, all data that is held in memory buffers is written through to the data file. Unless you have enabled Automatic file naming in the sampling configuration, your file is held on disk in the folder set in the Edit menu Preferences Sampling Tab with the name `datan` (`n` is a sequence number) and no file extension. You can use the File menu Save As... command to give this file a name.

If you attempt to close the file without using the Save As... command, and the file holds some data and was sampled for at least 5 seconds, you are prompted to save the file. If you don't choose to save it, the file is moved to the Recycle Bin (just in case you really did mean to save it). The Recycle Bin will allow you to use up a sensible fraction of your free disk space to save files; you can adjust the Recycle Bin Properties to limit the disk space to be used in this way. There are instructions to recover a data file in the Frequently Asked Questions.

Although it is a bit wasteful of disk space to keep all sampled data, it is surprising how many users accidentally fail to save their data!

If you Abort sampling, and confirm that you really want to abort, we do not save the file.

Disaster during sampling

The most common disaster is loss of power or a large electrical transient causing the computer to lock up. If this happens and you are forced to restart the computer without saving your file, you still have the possibility of recovering the data that was written to disk. The next time you start Spike2, it will detect that the last file sampled was not closed and will offer to recover it for you. Once recovered, you will likely need to fix the file by running S64Fix (for a `.smrx` file) or SonFix (for a `.smr` file).

High sampling rates

The code that transfers sampled data to disk runs in a separate, high priority thread. This ensures that data is saved unless another program creates a higher priority process that uses all the computer time. Buffer overflow can occur if the data rate is so high that the 1401 device driver cannot empty 1401 memory before it has become full. This should not occur with USB2 interfaces unless the overall data rate is very high or there are bandwidth bottlenecks in the disk system. Spike2 detects buffer overflow and stops sampling if this happens.

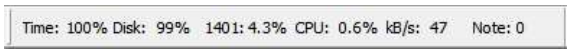
If you suffer from buffer overflow problems, please check the following:

- Check that you have the latest 1401 firmware. The Help menu About Spike2 dialog box will tell you if more recent firmware is available. Spike2 will refuse to sample if your firmware is seriously out of date.
- If you use USB to connect your 1401 and your 1401 has a USB2 interface, make sure it is connected to a USB2 port on your computer. Using a USB1 port (some older computers have both USB1 and USB2 ports) will reduce the maximum transfer rate by a factor between 5 and 40 depending on the type of 1401.
- You can get a measure of the fastest data transfer rates available for your combination of 1401, interface and computer by using the Try1401 program. Select Show estimated transfer speeds in the Tests menu Settings command and then run the DMA test.

If the 1401 detects that the host computer is slow writing to disk, it requests a “catch-up” mode where Spike2 abandons on-line display of new data (the display for new data is greyed out) to avoid competition between writes and reads in the disk system.

If the 1401 detects that the input event rate is too high for the 1401 to process, the special keyboard marker code `FF` is added to the keyboard channel. Sampling does not stop, as subsequent event times will be correct. If this happens, check that you have set realistic expected sample rates for the event channels.

The Sample Status bar



The Sample Status bar gives you an indication of how hard the 1401 and the host PC are working to capture and transfer data to disk. The bar also displays how much sampling time and disk space remains before sampling will stop. If the bar is not visible during sampling you can show it from the Sample menu or by right-clicking in the toolbar area and selecting Sample Status. The bar is normally made visible when a data file is opened for sampling unless the data file was created by a script or an Edit menu Preferences option is set to prevent it opening. The bar shows five values:

Time Percentage of sampling time remaining before sampling stops. You can limit the sampling time in the Automation tab of the Sampling Configuration. If you do not limit the time, you can run for up to 8 x 10^{18} clock ticks when sampling to a 64-bit file or for some 2 billion clock ticks when sampling to a 32-bit file (you can see how long this is in years, days, hours minutes and seconds in the Resolution tab of the Sampling configuration).

Disk Percentage of disk space remaining before sampling stops. You can limit the disk space in the Automation tab of the Sampling Configuration. If you do not limit the disk space, for 32-bit files the limit is set to 1 TB in big file mode or 2 GB if not in big file mode and no limit for 64-bit files. The free space on the disk is checked just before sampling starts and the limits are reduced if there is less free space than the limit. Sampling will stop if you get within a few megabytes of the free disk space when you started. If other activities are also filling the disk sampling may stop sooner, and possibly in a less controlled manner.

1401 The percentage of the available time that the 1401 interface is using to transfer captured data back to the host computer and match any spikes to templates. This should be fairly constant with waveform channels, but will increase when the rates of event or WaveMark channels increases. It can also be increased by a slow host interface or problems with the host computer disk system. If it heads towards 100% you are in trouble.

CPU The percentage of the available time that the data capture thread in the computer is using to process the incoming data from the 1401 and write it to the data file. If you have a computer with multiple processors, this is the proportion of the time on a single processor. This will usually be less than 10%.

kB/s The number of kB (kilobytes) of sampled data that is being transferred to the host per second.

Note The number of warning messages written to the Sampling Notes log. Double click this field to open the Sampling Notes window during sampling.

The 1401, CPU and kB/s fields display as n/a if the 1401 firmware is not sufficiently up to date, but this should never happen as we refuse to sample if old firmware is detected. You can download firmware updates from the CED web site firmware page.

Saving configurations

It would be very tedious if you had to set up the exact screen configuration you wanted each time you sampled data. To avoid this, you can save and load sampling configurations from the File menu. The saved sampling configuration file has the extension `.s2cx` and defines:

- The position and size of the application window
- The list of channels set for sampling and their sampling parameters
- Derived channels and real-time channel processing.
- The position of all windows associated with the new file
- The displayed channels and event display modes of the channels in time windows
- The name of any output sequence document to be used during sampling
- The list of waves and any associated waveform data for on-line waveform output
- Script files to run at specific times or on specific events during sampling
- File automation controls that allow you to sample a sequence of files

- The processing and update modes and positions of all result windows

The configuration does not include the contents of result windows. Whenever sampling finishes, the application saves the configuration as `LAST.s2cx`. When you run Spike2, it searches for and loads the configuration file `DEFAULT.s2cx`. If this cannot be found, it uses `LAST.s2cx`. These files are kept in the user application data folder (in old versions of Spike2 they were kept in the folder from which Spike2 was run). Remember that you can always recall the configuration that you used most recently, even if you forgot to save it.

If you have several configurations that you use very regularly, you can add them to the Sample bar with the Sample menu Sample Bar List… command. Once you have done this you will be able to start sampling with a saved configuration by clicking a button on the Sample bar (see the *Sample menu* chapter for a full description of the Sample bar). Alternatively, you could keep short cuts to configuration files on your desktop and start Spike2 by double clicking them.

You can also see a list of the last used 10 configuration files from the File menu, Load configuration command.

Warning
We suggest that you do not rely on `LAST.s2cx` for important sampling configurations as it is overwritten each time you sample. It is better to save your configuration to a named file and load it using the File menu or the Sample Bar or by double clicking the file. From [10.07], Spike2 saves the last-known configuration that the current configuration was derived from and this can be accessed by right-clicking the Sampling Configuration dialog title bar and with the `SampleConfig$(-1)` script command.

Search order for the last and default file

Over time, there have been several places where the `last` and `default` files were stored. We attempt to not break older installations by searching for a configuration to use in the following order. Note that we always write `last.s2cx` to the user application data folder.

1. `default.s2cx` in the user application data folder.
2. `default.s2cx` in the application data folder.
3. `default.s2cx` in the folder from which Spike2 was run (for backwards compatibility).
4. `last.s2cx` in the user application data folder.
5. `last.s2cx` in the application data folder.
6. `last.s2cx` in the folder from which Spike2 was run (for backwards compatibility).

The search stop at the first found file. If you are in doubt as to where your current configuration came from, open the Sampling Configuration dialog. The path is shown in the dialog title (and can be copied to the clipboard by right-clicking on it). At some future version we will stop searching the folder from which Spike2 was run.

Old format .s2c configuration files

Versions of Spike2 up to 7.10 (December 2009) used binary configuration files with the extension `.s2c`. Although compact, this file format was inflexible (often being a copy of the memory structures used inside Spike2), and made adding new features and enhancing old features more difficult than it needed to be. Spike2 versions from 7.11 onwards write configuration data to `.s2cx` files, which are coded in XML and though somewhat verbose, make it relatively easy to add new features. Spike2 versions 7.11 onwards and all versions of 8 still read the old `.s2c` formats, but do not write them. Spike2 versions 9.00 and later do not read `.s2c` files, which allowed us to simplify and rationalise a lot of code. If you come across `.s2c` files, they are likely to come from Spike2 version 6 (from 2006) or earlier as a version 7 user who kept up to date would be using the new format.

How to convert a .s2c file to .s2cx

To do this run Spike2 version 8 (supplied on the same disk as Spike2 version 11) and use the File menu Load Configuration command to open a `.s2c` file, sample data (briefly), then use the Save Configuration command to write the equivalent `.s2cx` file.

See also

Change to Resources

Symbolic names in sampling configuration path

At Spike2 version [10.15] we added an experimental feature to configuration files to use symbolic names in file paths that are part of the configuration. These are paths to: output sequencer files (`*pls`), script files (`*.s2s`) and data files for arbitrary waveform output (`*.smr`, `*.smrx`). As this is experimental, you can disable this feature with a new Edit menu Preferences, Compatibility Tab option.

When saving a sampling configuration, Spike2 now scans the file paths, and if these start with a path to a known place, we replace the known place with `$(SymName)`. Spike2 also saves the path without this change so that the configuration file is compatible with previous versions of Spike2. When a configuration loads, if there is a symbolic path for a file, we try that first, but if this does not locate the file, we try the original path.

This should have three effects:

1. It should make it easier to move configuration files between users when referenced files are stored in folders with locations that depend on the user account name.
2. It should make it easier to move configuration files between Spike2 versions as several paths include `Spike<Version>`.
3. If you keep your configuration file (`*.s2cx`) and associated output sequence and arbitrary waveform files in a folder, moving the folder should leave the configuration file in a working state.

List of Symbolic paths

The initial list of paths that are replaced by symbolic names includes the following.

SymName	Typical value in Spike11
Here	Path to the configuration file that holds the symbolic name
AppData	"C:\ProgramData\CED\Spike11\"
AppShared	"C:\Users\Public\Documents\Spike11Shared\"
UserLocalData	"C:\Users\<User>\AppData\Local\CED\Spike11\"
UserAppData	"C:\Users\<User>\Documents\Spike11\"
MyDocuments	"C:\Users\<User>\Documents\"
Desktop	"C:\Users\<User>\Desktop\"
Install	Spike2 installation path

`<User>` stands for the user name (and domain). The list is searched for matches in table order, so longer matches are preferred, but paths relative to the configuration file are always matched first.

Why have we done this?

There is a common problem when a user shares a configuration file with another user that the configuration fails to load because a referenced file, often the output sequence, cannot be found. This can be perplexing, especially when the file was saved in the same folder as the sampling configuration file, and still is in the same folder, only the user has changed. The problem is that configuration stores the full path to the file, so if user John in domain XYX saves a file in his Documents folder, it may have the path: `C:\Users\john.XYZ\Documents\Test\sequence.pls`, but user Sarah at domain ABC will be using `C:\Users\sarah.ABC\Documents\Test\sequence.pls`, so the configuration will not be able to find it, even though the files have the same relative positions.

After the change, we will still save the original path, but we also store: `$(MyDocuments)Test\sequence.pls`, which should work for both users if they both have a `Test` folder in their `Documents` folder with the sequencer file.

We also allow paths to be relative to the sampling configuration file as users commonly put all the files used for sampling in the same folder.

Sequence of operations to set the configuration

This section describes a sequence of operations that build a new sampling configuration from scratch. You will find that once you have built a few configurations, it is much simpler to load an existing configuration and change the sections that do not fit your requirements, rather than re-build entirely. The steps are:

1. Open the Sampling Configuration dialog from the Sample menu.
2. In the Channels tab set the channels to be sampled and their sampling rates.
3. Set the Resolution values to give the best fit of run time and sampling rates for waveform channels.
4. Select the appropriate Sampling Mode for your needs.
5. In the Sequencer tab select an output sequence file or create a graphical sequence or select None.
6. Set the Automation values as required by your application.
7. In the Play waveform tab select any waves needed for waveform output.
8. Click the Run now button.
9. Arrange the time view as you require and add any duplicate windows.
10. Add required Analysis processes generate result windows, set their update mode and screen position.
11. Add Measurement processes to generate XY views or send their results back to the original time view.
12. Use the File menu Save Configuration command to save the configuration.

Once you have saved a configuration, you can re-use it by loading it before you use the File menu New command to start a new data file. You can combine the loading and opening a new file by adding the saved configuration to the Sample bar with the Sample menu Sample Bar List… command; this lets you open a new file (and even start sampling) with a single mouse click or `Alt` key combination in the Sample Bar.

Recording sampling setup as a script

You can generate a script only version of the sampling configuration (with some limitations, for example Talker setup does not record), by adding two steps to the actions above:

- After step 7 open the Script menu and select: Turn Recording On.
- After step 11 open the Script menu and select: Turn Recording Off.

It is usually easier to save the desired configuration as a configuration file, the use the `FileOpen$(name$,6);` command to load the configuration followed by the `FileNew(0,3);` command to generate a new file, ready to sample.

5: Output sequencer

Output sequencer

While sampling data you can generate precisely timed digital pulses and analogue voltages, monitor your experiment and respond to input data in real time with the Spike2 output sequencer.

Overview

An output sequence is a list of up to 8191 instructions. It runs at a constant, user-defined rate of up to 100 instructions per millisecond (250 with the Power3). The sequencer has the following features:

- It controls digital output bits 15-8 with changes on the next clock tick and bits 7-0 as unclocked changes.
- It controls the 1401 DACs (Digital to Analogue Converters) to produce voltage pulses and ramps.
- It can play cosine waves at variable speed and amplitude through the DACs.
- It can play trains of digital pulses that run in parallel with other operations.
- It can test digital input bits 7-0 and branch on the result.
- It can record the digital input state or an 8-bit code to the digital marker channel.
- It supports loops and branches and can randomise delays and stimuli.
- It has 256 variables (`V1` to `V256`) that can be read and set by on-line scripts.
- It supports a user-defined table of values for fast information transfer from a script.
- You can command the sequencer to jump to particular actions with keyboard commands, TextMark input and from the script language.
- It can read the latest value from a waveform channel and the number of events from an event channel. Using this information, real-time (fractions of a millisecond) responses to input data changes are possible.
- It can control and monitor the arbitrary waveform output.
- You can replace the output sequence with a different sequence during sampling.

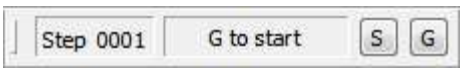
You write sequences with the text editor where each line of text generates one instruction or with the graphical editor where each graphical item generates one or more instructions. Text sequences are usually held in files with the extension `.pls`. From Spike2 version [10.15] onwards you can also save a sequence as text as part of the sampling configuration; this avoids the problem of a sampling configuration getting separated from the file holding the required sequence and can be more convenient when a script is used to generate a sequence.

Sequencer control panel

You show and hide the control panel with the Sequencer Controls option in the Sample menu or by right clicking on any toolbar to activate the context menu. You can also dock it on any edge of the Spike2 application window. When undocked, the control panel displays sequence entry points as the key that activates them and a descriptive comment. When docked, the keys display as buttons and the comment is hidden; move the mouse pointer over a key to see the comment as pop-up text. Click the mouse on a key and the sequencer will jump to the instruction associated with the key. The key is also stored as a keyboard marker. This is equivalent to pressing the same key in the time window or using the script language `SampleKey()` routine.

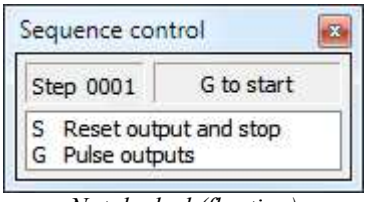
Side dock

The control panel displays the next sequence step and any display string associated with it. You can use display strings to prompt the user for an action or to tell the user what the sequence is doing. Display strings are set with the text editor or the graphical editor by appending `>Display text` to the end of a comment. The sequencer always starts at the first instruction. You cannot re-route the sequencer until sampling has started.



Docked to top or bottom edge

The control panel is usually displayed if you start sampling from a Spike2 menu command. If the sample command comes from the script language or an Edit Preference is set to prevent it opening, the control panel visible state does not change. The control panel position is saved in the sampling configuration. However, the position is restored only if the control is currently invisible or floating, and the saved position was floating; if the control panel is docked, we assume it is positioned where you want it. When floating, you can resize the control to display longer comments.



Not docked (floating)

Special keyboard characters

Please note that keyboard marker codes `00`, `01` and `02` and codes above `7F` do *not* trigger other actions as they are reserved for special purposes.

Sequence jump disable

Sometimes you may want to stop users activating sequence sections with the keyboard or from the control panel. The Sequencer jumps controlled by field in the Sequencer tab of the Sampling configuration dialog lets you do this. The script language `SampleKey()` command can always activate sequencer sections; from [11.00] the Create TextMark dialog, serial line TextMark input and the `SampleText()` command can activate sequencer actions as an option.

Creating sequences

There are two ways to create output sequences: as an output sequence text file with the `.pls` extension or as part of the sampling configuration using the graphical sequence editor. The table lists the main differences between them.

	Text sequence	Graphical sequence
Edited with	Built-in text editor	Built-in graphical editor
Visualise output	No	Yes
Create with script	Yes	No
Stored as	`.pls` text files or in configuration	Part of the sampling configuration
Implemented by	Machine code like language	Drag and drop editing
Ease of use	Takes time to learn	Very easy to learn and use
Flexibility	All sequencer features available	Uses pre-set building blocks
Timing	One instruction per text line	Several instructions per item
Update during sample	Yes; load new `.pls` sequence/text	Yes; interactive edit and apply

Graphical sequences are much easier to generate than text sequences. However, they have limitations and you can write more complex sequences using all sequencer features with the text editor. Sequences produced by the graphical editor are converted internally to the `.pls` file format before they are used. You can save a graphical sequence as a `.pls` file that you can edit with the text editor; you cannot convert a `.pls` file into a graphical sequence.

The rest of this chapter describes the graphical output sequence editor, then the text editor, and finally the low level instructions used by both. You do not need to read about the low level instructions to use the graphical editor. However, knowledge of how the sequencer works will give you a better understanding of its capabilities and limitations.

Sequencer speed

The output sequencer runs at a rate set by a clock inside the 1401. You set the clock rate in milliseconds per tick in the Sequencer Tab of the Sampling configuration dialog when using the graphical editor or using the `SET or SCLK` directives in the text editor. The script command `SampleSeqClock()` can also change the rate. Supported 1401s allow intervals from 0.01 (Power3 and Micro4 allow 0.004) up to 3000 ms (older 1401s had a lower limit of 0.05 ms). If you set intervals less than a millisecond or use the sequencer text editor you should read the following information.

Sequencer technical information

The sequencer clock starts within a microsecond of recording time zero and is time locked to the 1401 event timing and waveform channel recording. Each clock tick books an interrupt to run the next sequencer instruction and updates digital output bits 15-8 if they were changed by the previous instruction.

An interrupt is a request to the 1401 processor to stop what it is doing at the earliest opportunity and do something else, then continue the original task. The time delay between the interrupt request and the instruction running depends on what the 1401 is doing when the clock ticks and the speed of the 1401. This delay is typically less than a microsecond so instructions do not occur precisely at the clock ticks; changes to digital output bits 15-8 are set just after a clock tick and occur exactly on the next tick (*clocked* changes). Changes made by the sequencer to the 1401 DACs and digital output bits 7-0 occur after the clock tick.

The sequencer runs one instruction per interrupt. At the start of each interrupt the sequencer checks to see if there are any cosine output, RAMP or digital pulse train outputs running and takes care of these before running the instruction.

The table shows the minimum clock interval, the timing resolution, the approximate time per step, the estimated extra time used for cosine, ramp and digital pulse train output and the time penalty for using the relatively slow `DIV` and `RECIP` instructions for the Power3, Power1 and the Micro2 and 3. The Power2 lies between the Power1 and Power3 in speed, the Micro3 is about 20% faster than the Micro2. The Micro4 is several times faster than a Micro3 and has a built-in divide instruction. Notice that the first two are in units of milliseconds (as you set the period in ms) and remainder are in microseconds.

	Power3	Power1	Micro2/3	Micro4
Minimum tick interval	.004 ms	.010 ms	.010 ms	.004 ms
Tick interval resolution	.001 ms	.001 ms	.001 ms	.001 ms
Time used per tick	<1 µs	<1 µs	~1 µs	<1 µs
Cosine cost per tick per channel	.2 µs	0.55 µs	~1 µs	<1 µs
`RAMP` cost per tick per channel	.1 µs	0.5 µs	0.7 µs	.1 µs
`DIGPS` cost per tick per bit	.1 µs	0.5 µs	0.7 µs	.1 µs
`DIV, RECIP` penalty	.2 µs	<1 µs	<3 µs	none

The Minimum tick is the shortest interval we allow you to set. The Time used per tick is how long it takes to process a typical instruction. The Cosine penalty/tick is the extra time taken per cosine output. The Ramp penalty/tick is the extra time taken per ramped DAC. Time used by the sequencer is time that is not available for sampling, spike sorting or arbitrary waveform output. To make best use of the capabilities of your 1401 you should set the slowest sequencer step rate that is fast enough for your purposes.

The interval you set must be a multiple of the Resolution field for your 1401. This is 0.001 ms for the Power1401 and Micro1401. Spike2 will not use a sequence if the interval is not an exact multiple of an achievable resolution for your 1401 as this would lead to inaccurate timing.

What happens if you run the sequencer too fast

There are many competing processes within the 1401 interface, and if you try to do too much at the same time, some of these processes have to wait. If the output sequencer cannot complete a step in the clock time you set (this will usually be due to other activities such as event or digital marker input or arbitrary waveform output), the next step is delayed. If this happens, it is detected by the 1401 hardware as a problem. What happens next depends on the type of the 1401 and the revision of the Spike2 software.

Power3, Power2, Micro3, Micro4

The error is detected, and Spike2 sets a marker into the Keyboard marker channel. The first marker code is 0xFF and the second is 0x02. When sampling ends, a warning message is displayed. The time of the marker will be the current time as seen by the sampling program, which is typically a few milliseconds behind the 1401 time. The time at which the error was detected can be before or after this time, usually within a few milliseconds.

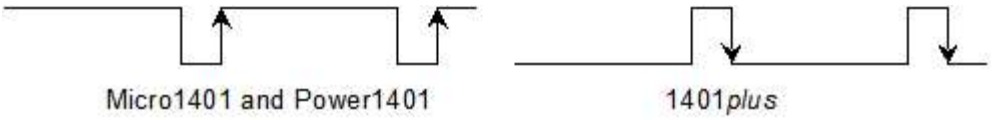
Power1, Micro2

Sampling stops with an Interrupt overrun error message.

Sequencer clock output

The output of the clock that controls the sequencer is available on the 1401 front panel Clock connector. This TTL output signal starts high and the sequencer steps are synchronised with the rising edges of this output. The

minimum pulse width is 1 microsecond. If you are upgrading from a 1401*plus* (last supported for sampling in Spike2 version 7), note that this had the inverse output.



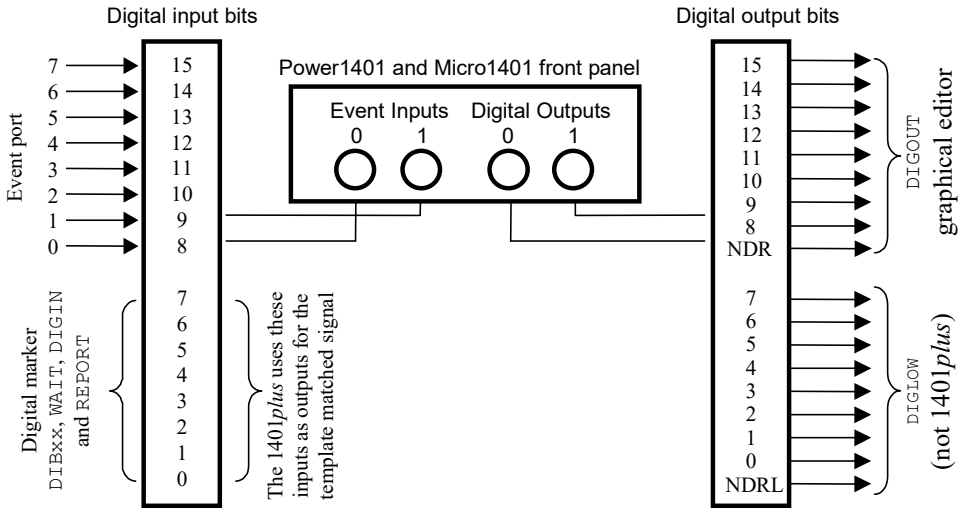
This clock runs whenever you sample with an output sequence. It can be used to synchronise external equipment. For example, if you connect this output to a counter, and place the counter in the field of view of a video camera that is used to record some other aspect of your experiment, the number on the counter links visual data with an exact time in the Spike2 file. If your sequencer ran at 1 millisecond per step, the counter would display sampling time in milliseconds.

Digital input and output

The 1401 family has 16 digital inputs and 16 digital outputs. Digital input bits 15-8 are the event ports and are not used by the output sequencer. Digital input bits 7-0 are used for the Digital marker channel; you can also test these bits from the sequencer.

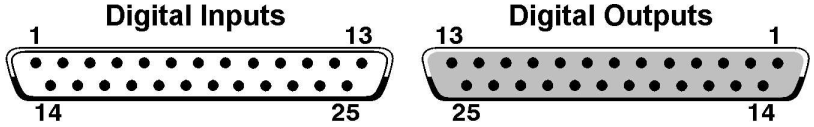
The sequencer controls digital output bits 15-8 individually to generate accurately timed pulses. Output bits 7-0 can be set with the `DIGLOW` instruction. The on-line template matching code can also use bits 7-0 of the output to signal spikes that match templates.

In a text-based sequence, the digital outputs are controlled by the `DIGOUT` and `DIGLOW` instructions and the digital inputs are tested with `DIBxx`, `WAIT`, `DIGIN` and `REPORT`. The graphical editor controls output bits 15-8 and tests input bits 7-0 with the delay and branching instructions.



The digital input and output ports are 25-way connectors. The data and ground pins are the same on both. See your *1401 Owners Manual* for full details of all pins. The output connector is a socket (with holes), the input connector is a plug (with pins).

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	GND
Pin	1	14	2	15	3	16	4	17	5	18	6	19	7	20	8	21	13



Digital output bits 8 and 9 are connected to the front panel as Digital Outputs 0 and 1 in addition to the rear panel. If you have a Spike2 top box (Micro1401 and Power1401), the remaining digital output bits 10-15 are also available on the front panel as Digital Outputs 2 to 7. The NDR output (New Data Ready, output connector pin 12) pulses for 1 microsecond after a change to the digital output bits 8 to 15. NDRL (output connector pin 23) pulses for 1 microsecond after a change to output bits 0 to 7. The pulses are low-going for a Power1401mk 1 and high-going for all other 1401s.

Digital input bits 8 and 9 are also connectable to the front panel as Event inputs 0 and 1. They can also be routed to the rear via a preferences option. The Micro1401 Spike2 top box also brings digital input bits 10-15 to the front panel. See the Marker data description for details of digital marker data capture.

The graphical editor

With the graphical editor, you can generate complex sequences without any knowledge of the underlying control language as used in the output sequencer text editor. You can also modify the graphical sequence while you are sampling. There are limits on what can be achieved with the graphical editor. For example, the graphical editor does not have access to tables of values. In complex cases you can use the graphical editor to generate most of the code, then use the text editor for the final touches.

The graphical sequence is converted to a text sequence, just before it is used. However, unless you are really interested in how it works, or you want to extend the sequence beyond the capabilities of the graphical sequence, you never need to look at this. You might need to look at the sequence if timing latencies are very important to you. You can generate a text sequence version using the Write as text sequence... button in the Sequencer Tab of the Sampling Configuration.

Structure of a graphical sequence

A graphical sequence is defined in Sections. Each section has a duration, a repeat count, an optional key that can jump control to the section and an action to take when the end of the section (and any repeats) are completed (stop, or jump to another section). There is a list of up to 27 sections that you can define: the Initial section (which runs when the sequence starts and may be the only one you need) plus 26 more called Section A, Section B, and so on to Section Z. Most graphical sequences use only a few sections.

Structure of a Section

Most sections are quite simple and contain timed digital output and DAC changes. If you assign a key to a section, the section can be started by a user key press or by clicking the appropriate button in the sequencer control panel. Most sections run linearly from start to finish executing the instructions that you have configured in time order. You can make a section non-linear in two ways:

1. By making the section wait for a condition (during which period the section time base is effectively paused)
2. By making the section branch to a label within the section, or to the start of another section

Getting started

If you are new to the graphical editor, start by following the Getting started tutorial, which will introduce you to graphical editing. You can then work through the more detailed reference that follows, starting with a description of the Graphical editor Tab and then the section on Graphical editing and the Graphical palette.

Getting started

To get accustomed to the graphical editor, we will produce 10 millisecond wide TTL pulses at 1-second intervals from digital output bit 8 (available on 1401 front panel Digital Output 0). We will associate the key `G` for Go with the pulses, and the key `S` for Stop will stop them.

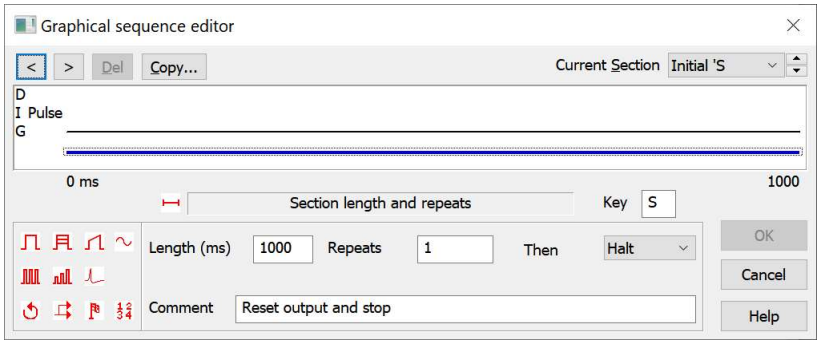
Open the Sampling configuration dialog Sequencer tab and select Use graphical editor. Click the Clear graphical editor button to remove any previous sequence. Set the Sequencer time resolution to 10 milliseconds and check the Show time as milliseconds box. We do not need any DAC outputs, so clear all the DAC outputs check boxes. We only need one digital output, so check the digital output 8 box and clear all the others. Now click the Graphical editor button.

At the top right of the new dialog there is a drop down list to select the current sequence section. A section has a length and repeat count. When the repeats are done, the section either stops or jumps to another section. To start with, make sure Initial is selected as the Current section. The Initial section always runs first and it sets the initial conditions.

The graphical representation of a section always contains a *control track* drawn as a thick line at the bottom. We choose output on digital bit 8 only, so there is a single digital output in the remainder of the space. There is

always one item selected in this area; the selected item has a grey rectangle around it. You can change to background and foreground colours of the graphical region from the Application colours dialog.

Click on the control track now.



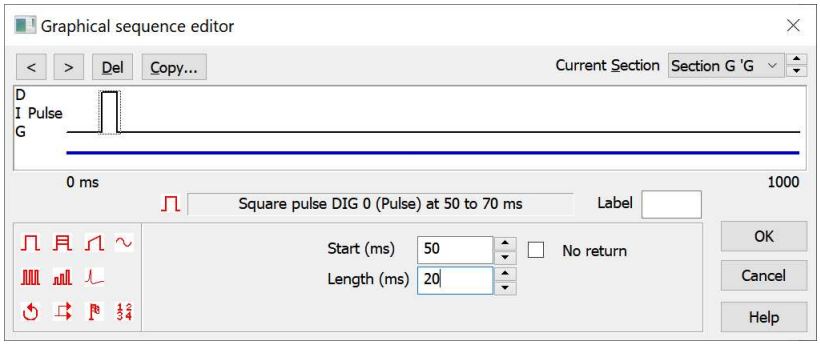
In addition to setting the initial state, we will use this section to stop our pulse output, so set the Key field to `S`, and set the Comment field to `Reset output and stop`. This comment will appear in the sequencer control panel for the `S` key. You can leave the Length, Repeats and Then fields at their default values (`1000`, `1` and Halt).

The next task is to create the pulse train and associate it with the `G` key. Select Section G in the Current section field (or any other section apart from Initial). The control track will be selected; click on it if it is not selected. Set the Key field to `G`. Set Repeats to 0 to mean repeat forever and the section length to 1000 milliseconds. The Then field will grey out as the repeats never end. Set the comment to `Pulse outputs`. There is no intrinsic link between section G and the `G` key; however, it is often convenient to use sections with the same identifier as the key used to trigger them.

There is a palette at the bottom left of the dialog; you can drag icons from the palette and drop them on the DAC, digital output and control tracks. Click on the top left item in the palette and drag it to the digital output track and release it to create a pulse. Edit the start time of the pulse to 50 and the length to 10 milliseconds.

If you have multiple Digital or DAC outputs enabled it can be useful to give them names rather than numbers. To do this, click the mouse on the port number and type a label of up to 5 characters (we have set it to Pulse in the example).

This completes the pulse set up. Click the OK button to return to the Sampling configuration dialog. Click the Run now button to start sampling with the output sequence you have just created. The sequencer control panel will display two items that you can select: G Pulse outputs and S Reset output and stop.



Start sampling – the sequencer will run the Initial section, which sets the starting values for all the digital and DAC outputs we have chosen to control. We did not request any other action in this section so nothing will happen until you use the G button in the sequencer control panel or select the sampling window and press the `G` key to start the output pulses from digital output 8. You can stop the pulses with the S button in the control panel or the `S` key on the keyboard.

Modify sequence while sampling

You can open and display this dialog during data acquisition if sampling starts with an active graphical sequence. If you do this, the OK button changes to Apply and you can edit the sequence. Each time you click Apply, the sequence will start again at the Initial section. We anticipate that this will be used to "tweak" settings, such as pulse widths.

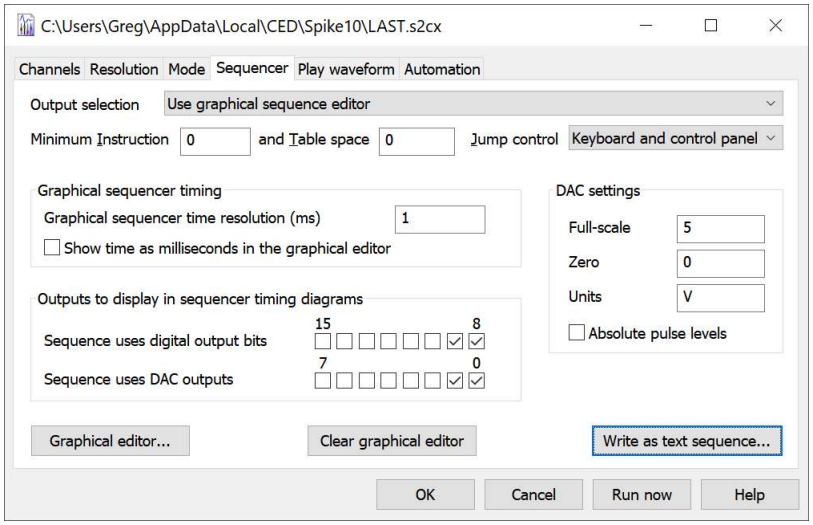
To try this out, use the Sampling menu Graphical Sequence Editor... option, select Section G. Click on the pulse shape in the DIG 8 track, edit the Length field to 100 ms, then click on Apply. This will transmit the updated sequence to the 1401 and run the Initial section. If you use the G button the pulse will now be wider.

You can also open the graphical sequencer during sampling by clicking this button in the system toolbar.

Space is reserved in the 1401 for the sequencer based on the size of the sequence set when sampling starts. If you want to add instructions to the sequence while sampling you should reserve extra space using the Sequencer tab of the Sampling Configuration dialog.

Graphical editor tab

To view the graphical editor, open the Sampling configuration dialog and select the Sequencer tab. Then select Use graphical editor from the Output selection drop down list. The editable fields in this dialog set values that apply to the entire sequence:



Sequencer jumps controlled by

This field applies to graphical and text sequences. It allows you to stop interactive control of sequence jumps to prevent accidental changes caused by a user key press or mouse click in the sequence control panel. The `SampleKey()` script command can always cause the sequence to jump. You can choose from: Keyboard and sequencer control panel, Sequencer control panel and Script commands only. The script language equivalent is `SampleSeqCtrl()`.

From [11.00], sequencer jumps can also be commanded by the TextMark channel, both for manually entered TextMark data and for TextMark data logged from a serial line. These have separate enables and are not affected by this contol.

Sequencer time resolution

This sets the time resolution of your sequence and the clock interval of the sequencer clock. This is also the minimum duration of any pulse. All actions in the sequence occur at integer multiples of the time you set here. You can set values in the range 0.01 to 3000 milliseconds. You need a Power1401 or Micro1401 mk II or -3 to set values less than 0.05 milliseconds. Some actions take more than one clock interval.

Show time as milliseconds

Check this box to display and edit time in the graphical editor as milliseconds and not seconds. This is purely for your convenience; if your sequence sections are all less than a second you will probably find it more convenient to use milliseconds.

Sequence uses digital output bits

Check the boxes for the dedicated digital outputs that you will use. Only these outputs will appear in the editor, reducing visual clutter. If you do not require any digital outputs, clearing all the check boxes will save an instruction at the start of each sequencer section. The sequence uses the upper 8 digital output bits (15-8). Bits 8

and 9 are available on the 1401 front panel as Digital outputs 0 and 1. All bits are also available on the 1401 rear panel Digital Output connector.

Sequence uses DAC outputs

Check the boxes for the Digital to Analogue Converters (voltage output devices) that you will use in your sequence. Unused DACs are not included in the graphical editor (to reduce visual clutter) and no sequence code is generated for them, which saves instructions at the start of each sequence section.

If you use one or more DACs for arbitrary waveform output only do not include them here unless you want to be certain that they have a defined value when you enter a sequence section. All 1401s have at least DACs 0 and 1; the Power1401 has DACs 0 to 3. Top boxes allowing more DACs can be added to Power1401s. The sequencer supports DAC numbers 0-7.

DAC full-scale, zero and units

You can define the DAC outputs in units of your choice. 1401 DACs normally have a range of ±5 Volts, but ±10 Volts systems exist. Set Units to the units you want to use. Set Full-scale to the value in these units that corresponds to the maximum DAC output. Set Zero to the value in your units that corresponds to a DAC output of 0 Volts. For a ±5 Volt system calibrated in Volts, set Full-scale to 5, Zero to 0 and Units to `V`. If you want the output in millivolts, set Full-scale to 5000, Zero to 0 and Units to `mV`.

Absolute pulse levels

The DAC pulses take their starting level as the current DAC value at the pulse start time. The DAC then changes to another value, then back to the original level. Normally you define pulses in terms of the pulse amplitude relative to the starting level and all pulses add. If you check this box, then you set the absolute level for the DAC to change to. Data is written for each sequencer section that has any output defined or that has a key set.

Write as text sequence…

You can use this button to output the graphical sequence as a text sequence. You might do this if the graphical interface almost does what you need and you need to hand-edit a few extra instructions, if you want to run multiple graphical sequences, or if you need to save the sequence for documentation purposes. A file selector dialog opens for you to choose a name for the `.PLS` file.

When you use a graphical sequence, the first thing that happens is that it is converted to a text sequence, which is saved in the file `S2PSEQ$.PLS` in the current user application data folder. This is then loaded for use in exactly the same way as a user-written text sequence.

Clear graphical editor…

This wipes all graphical sequences. You must confirm this action as you cannot undo it. This also clears the keys associated with each section.

Graphical editor…

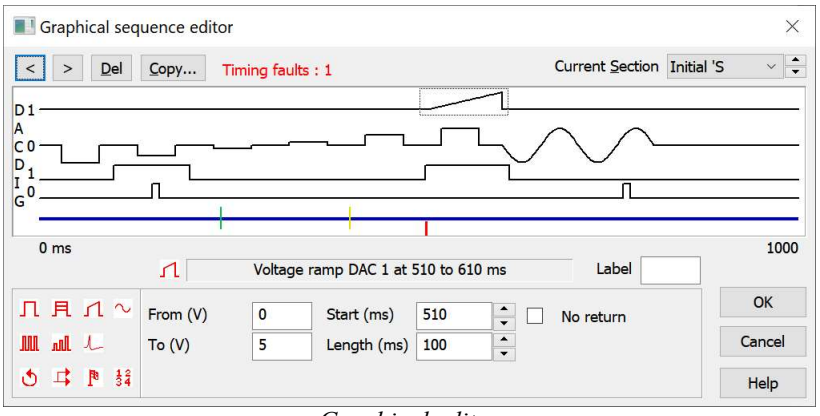
This opens the editor so you can make changes to the sequence.

Graphical editing

To open the graphical editor, select Use graphical editor and click the Graphical editor… button in the Sequencer tab of the Sampling configuration dialog. You can resize the dialog by clicking and dragging an edge. Double-click the title bar to maximise the dialog, double-click again to minimise it. The editor window has five areas:

1. a window with a graphical representation of the output sequence
2. above the graphical window are controls to iterate through and delete selected items, a message area and the Current Section selector
3. the lower left-hand corner holds a palette of items to drag into the graphical window
4. the lower right hand corner has control buttons
5. the settings for a selected graphical item lie between the palette and the buttons

The OK and Cancel buttons both close the dialog. OK accepts all changes, Cancel rejects all changes. The Help button displays the information you are reading!



Graphical editor

Sections

There are 27 sections: Initial and Section A through Section Z. The Current section field sets the section to display and edit. There are 27 possible sections, named Initial then Section A through Section Z. If a section has an associated Key, this is displayed to the right of the section name, for example Initial 'S. To make it easier to identify empty sections in the Current Section drop down list, sections without a Key or any added items have their names in brackets, for example (Section A).

The Initial section runs when the sequence starts; in some cases this may be the only section you need. The remaining sections are optional. A section displays a representation of the output for each DAC and digital output that is set for use in the Sequencer tab of the sampling configuration. There is a thicker line at the bottom for the control track, which holds all other sequencing actions.

Selecting items

Click an item in the graphical view to select it. A grey rectangle marks the selected item and the item settings appear below the display. The < and > buttons at the top left select the previous and next item on the current track; they are very useful when items overlap. If you hover the mouse over one of the digital or DAC tracks a tool tip will tell you the track name, the number of items in the track and remind you that you can click to select an item or double-click the track to zoom it. The first item in a digital or DAC track spans the entire track length and sets the initial track level. Any additional items are additions to the track, for example pulses or pulse trains.

Section settings

Click on the control track clear of any dropped items. You can now set the associated key, the section length, number of repeats, action when the repeats are done, and a comment that is displayed in the sequencer control panel to identify sections with keys. You can set a section length up to 10000 seconds and up to 1000000 repeats! If you want a section to repeat forever, set the repeat count to 0. The Then field determines what happens when the section ends. You can stop the sequencer with Halt, or select another section to run.

Comments and display strings

You can set a display string that appears in the Sequencer control panel when the instruction associated with the comment is displayed by adding it to the end of the comment preceded by a > character. For example: `Pulse outputs>Start pulsing`

Key field

In an empty graphical sequence, no section has a key assigned. If you assign a key to a section, that section can be activated by a user key press (if enabled) and the section will be listed in the sequencer control panel together with the section comment. You can set the key to any of `A-Z`, `a-z` or `0-9` (upper and lower case are different). Only one section is allowed to use any particular key.

Adding and deleting items

The graphical palette at the bottom left-hand corner of the dialog contains all the items that you can add to the display. Move the mouse over an item so see a short description. Click an item and drag it to a suitable track, then release to add it to the section. The Del button on the top line of the dialog removes the selected item. You cannot remove the control track or the lines that represent the initial state of the DACs and digital outputs.

Change track labels

From version [10.07] you can change the default track labels (0-7) to a label of your choice of up to 5 characters. This label is visible within the graphical editor only and the only purpose is to remind you of the function of each Digital or DAC output. To edit a label, click the label (next to the vertical DAC or DIG). The mouse pointer changes to show Aa when you are over editable text. You may not set a blank label.

Dragging and duplicating an item on a track

To move an item, click on it and drag to the destination and release. To duplicate an item, hold down the `Ctrl` key, click on the source item and drag it to the destination position; you must keep `Ctrl` held down when you release the mouse button or the duplicate operation will become a drag. While dragging, the positions are quantized (for example a 1 second section drags in 10 ms steps). You can hold down the `Shift` key to remove this, allowing positioning as accurately as the pixel grid allows. For fine tuning of positions you will need to set the position as text.

Timing faults

The sequencer attempts to match the timing you request. If this cannot be done, timing conflicts are marked by a red vertical line below the control track and the number of conflicts is given in the message area. You will get a conflict if you try to position any action at the start of a section. This is because the first instructions in a section set the initial digital output and DAC state. You can choose to ignore timing faults; the sequence will run with the changes as close to the requested time as possible. The sequencer time resolution field of the main sequencer page sets how close the next instruction can be to the place you asked for.

Zoom a track

If you double-click any track except the control track, it zooms in size to occupy the display area. Double-click it again to display all tracks. A Z appears at the upper right corner of the graphical window when the window is zoomed. If you double-click an arbitrary waveform item this does not zoom the track; instead it refreshes the choice of arbitrary waveform.

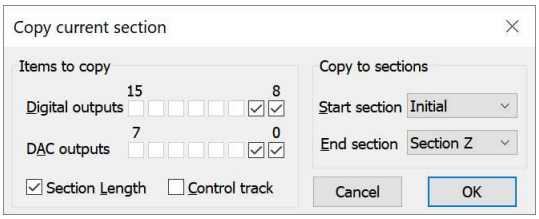
Copy section

The Copy button opens a dialog where you can copy tracks in the current section to a range of sections.

Setting initial DAC and digital levels

To set the initial DAC and digital levels, click on a track clear of any added items. The initial digital output level is 0 or 1; the initial DAC level is in the units set by the Sequencer tab of the Sampling configuration. Every section starts with the instructions that set these levels, so output changes caused by them will be as close to time 0 in a section as possible. The fact that these levels may not be set at exactly time zero is not considered a timing fault.

Copy section dialog

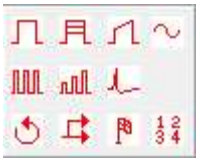


The Copy button opens a dialog where you can copy items in the current section to a range of sections. This makes it easy to create lists of similar stimuli.

The left-hand side of the dialog sets the items to copy; the right-hand side sets the destination. If you clear the Section Length check box, the length of the target sections does not change, otherwise the target sections are set to the length of the source section.

If you check the Control track box, the Section Length box is ignored and the length is always copied. For any item selected for copying, the corresponding item in the target section is first deleted, then the source item is copied.

Graphical palette



The palette contains 11 items you can drag and drop in the graphical window to generate a sequence. There are three dropping zones: a digital output, a DAC output and the control track. Items will only drop onto suitable targets. For example, you cannot drop a sinusoid on a digital track. There is no need to drop the items at exactly the right time point; you can edit the position afterwards.

The palette is replaced by an error message if you set an illegal value for a pulse; you must fix the error before you can add more items.

Dragging and duplicating an item on a track

To move an item, click on it and drag to the destination and release. To duplicate an item, hold down the `Ctrl` key, click on the source item and drag it to the destination position; you must keep `Ctrl` held down when you release the mouse button or the duplicate operation will become a drag.

Common editable fields

When you select an item in the graphical window you can edit the fields that relate to it. The following are described here to avoid repeating the descriptions.

Label

This field is normally blank. Use it to label the selected item so that you can branch to it. A label can be up to 6 alphanumeric (A-Z, 0-9) characters long and is case insensitive; `abc23` and `ABC23` are the same. Labels must be unique in each section, but you can have the same label in a different section.

You cannot set a label for the initial levels of the digital and DAC tracks or for the control track as these items all start at time 0 and can be branched to by referring to the section name. You cannot set a label for arbitrary waveform output as this would prevent an important optimisation required when the sequence is generated.

Start time/At

All digital and DAC change items have a Start time field and all items on the control track have an At field. The time is in units of seconds or milliseconds, as set by the Graphical editor settings, and is relative to the start time of the sequence section. There is a spin control to nudge the time on or back by the sequencer time resolution.

If you add items to the control track that take an unknown time to complete, for example a random delay or a wait for an input signal to achieve a set value, the At time determines where the items are drawn in the graphical editor. In this case the sequencer will maintain the time intervals between items wherever possible.

For pulses, ramps, sinusoidal and arbitrary waveform output, the sequencer attempts to produce the first output change at exactly the time you specify. For other item types, the sequencer attempts to run the first instruction of the item at the specified time.

Length
Several items have a length in seconds or milliseconds. For all the pulse types, this is the period for which the pulse changes to the amplitude or level you set before it reverts to the original level. For a ramp and a sinusoid, this is the length of the output. For arbitrary waveform output, this is set to the length of the arbitrary wave you associate with the command. You can set it to be less than the length of the wave, in which case the output is stopped after the time you have set. There is a spin control to nudge the time on or back by the sequencer time resolution.

Interval
For the pulse train items, this is the time between pulse starts in seconds or milliseconds.

Level (units)/Size (units)
These fields are used with DAC pulse outputs. The field you see depends on the state of the Absolute pulse levels check box in the Graphical editor settings. The Size field sets the amplitude of a pulse; the Level field sets the absolute level of a pulse. You set the value in the DAC units set in the Graphical editor settings.

No return
Normally the output from single pulses, ramps and sinusoidal output returns to the background level at the end of the item. If you check this box, the output change produced by this item is not removed at the end of the item. To indicate this, the grey rectangle surrounding the item extends to the end of the section. You could use this to ramp a DAC up to a value and leave it there.

Overlapping items
If digital pulses overlap, the result is the logical OR of the pulses.

If DAC items overlap on the same channel, the output depends on the state of the Absolute pulse levels check box in the Graphical editor settings. If it is clear, the result is the sum of the outputs. If it is checked, the output level is set by the last item in the overlap. There is an exception: arbitrary waveform output overrides all other items.

When adding pulses and pulse trains where the result would exceed the DAC range, the output is limited to the DAC range. However, pulses with an amplitude change on repeats can exceed the DAC range and wrap around. A value that goes off the top of the range will reappear at the bottom; a value that goes off the bottom of the DAC range will reappear at the top.

Graphical palette items

Some of the following descriptions say that a time value is 'limited to 31 bits of sampling clock ticks'. This is because the sequencer variables are 32-bit signed values. Values stored in them range from about minus 2 billion (-2,000,000,000) to plus 2 billion (2,000,000,000). At a time resolution of 1 microsecond, 2 billion ticks represents 2,000 seconds, which is usually sufficient. Other items may be limited to 31 bits of sequencer clock ticks for the same reason, but as the fastest sequencer rate is 5 microseconds per tick, this is unlikely to cause a problem.


Single pulse
You can drag this item onto either the digital or the DAC outputs. For a digital output, this sets the output to be the inverse of the initial level set for this output in this section. For a DAC output, you set the amplitude of the pulse with the Size field or the absolute level with the Level field.


Single pulse amplitude change on repeat
You can drag this item to a DAC output. It is for use in a repeated section. The Size/Level field sets the initial amplitude or absolute level of the pulse the first time the section runs. The Change field sets the amplitude change to apply on each subsequent repeat. You set the number of repeats by selecting the control track and editing the Repeats field. The changes are calculated in real time in the 1401. If the initial level plus the number of changes times the number of repeats exceeds the DAC range, the output wraps around.

Ramp

You can drag this item to a DAC output. The From and To fields set the initial and final amplitudes or levels of the ramp depending on the state of the Absolute pulse levels check box in the Graphical editor settings.

Sinusoid

You can request sinusoids on DACs 0 to 7 in the Power1401 mk II and -3. However, Micro1401s can only generate them on DACs 0 and 1 and the original Power1401 can use DACs 0 to 3.

The Size (units) field sets the sinusoid amplitude; this is not affected by the Absolute pulse levels check box. You can offset the sinusoid with the Centre (units) field. If the Absolute pulse levels check box is clear, the sinusoid and offset are added to the DAC value. If the check box is set, the DAC output is defined by the sinusoid and offset.

The Period field sets the time for one cycle of the sinusoid in seconds or in milliseconds. Alternatively, you can use the Frequency (Hz) field to set the frequency. The Phase field sets the initial phase in degrees. The output is a cosine, so a phase of 0 means start at maximum amplitude. A phase of –90 or 270 produces a sine output.

Pulse train

You can drag this to a DAC or digital output. It generates a train of pulses defined by the number of pulses (Pulses field), the length of each pulse (Length field) and the interval between each pulse and the next (Interval field) or the pulse Frequency (Hz) field. For a DAC output you can also set the amplitude with the Size/Level field.

This method is inefficient if you need to generate a large number of pulses as each pulse takes several instructions. You should consider setting up a single pulse as a section and repeating the section to create the pulse train. The maximum number of pulses you can set in one train is 1000 (was 400 before 8.03).

Pulse train with varying amplitude

You can drag this to a DAC output. It generates a train of pulses defined by the number of pulses (Pulses field), the length of each pulse (Length field) and the interval between pulse starts (Interval field) or the pulse Frequency (Hz) field. You set the amplitude of the first pulse with the Size/Level field. The pulse size changes by the value in the Change field for each repeat.

This method is inefficient if you need to generate a large number of pulses as each pulse takes several instructions (each pulses takes 4 instructions). You should consider setting up a single pulse with changing amplitude as a section and repeating the section to create the pulse train. The maximum number of pulses you can set in one train is 1000 (was 400 before 8.03).

Arbitrary waveforms

Drag this item to the control track to start playing a waveform through the 1401 DAC outputs. You cannot drag this item if no arbitrary waveforms exist. You can play arbitrary waveforms through any combination of DACs 0-3 (DACs 2 and 3 do not exist on all 1401s). The DACs used are set when you create the waveform and can be edited in the Play Waveform tab of the Sampling configuration. Create the arbitrary outputs with the Sample menu Output Waveform command or from a script.

If more than one waveform is defined in the sampling configuration, you are prompted to choose one from a list. Double-click the dropped item in the control track to redisplay the list and reset the waveform length. Each waveform is identified by a code; this is either a single alphanumeric character or two hexadecimal digits. When you set this in the Key field or by selecting a waveform, the Length field is set to the waveform length or to a lesser value if the wave is longer than the section or to 0 if there is no matching wave. Set the length shorter than the entire wave to truncate the output.

Arbitrary output takes appreciable time to set it up; times in excess of 10 milliseconds are possible. When generating the output, this set up is moved as far forward in the sequence section as possible so that the output starts at the exact time you set. If the preparation has not completed by the time you request output to start, the sequence stalls until it is ready.

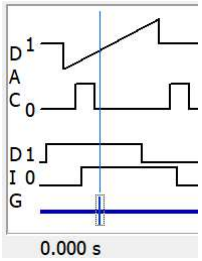
When the arbitrary output ends, the DAC outputs are returned to the background level as soon as possible. If the arbitrary waveform has more than one channel, there will be one sequencer clock period between each DAC changing to the background level. You can edit the duration of the wave, but only to be shorter than the

nominal length. The nominal length is not affected by the replay speed (you can change this in the Play Waveform tab).

Linked and repeated waves and speed changes
If you have set the wave to play more than once or to link to another wave, or you have adjusted the waveform replay speed to be less than 1.00, you must add a Wait for time, condition or variable before the end of the waveform to extend the time it has for running. If you do not, the waveform will stop at the end of the time slot allocated to it. Currently there is no way to wait for the repeats to end; if you want this functionality you must use a text sequence.

Drawing the waveform
If the arbitrary waveform is stored in memory, or (from [10.10]) the waveform is stored on disk and not too long, Spike2 draws it in position. Otherwise, the waveform is represented by a black rectangle.

Wait for time, condition or variable



You can drag this item to the control track only. It pauses the sequence until a condition is met. The duration of this item may be unknown; it is drawn as if it were of zero width and is always treated as if it had a duration of 1 tick. This means that all Wait for... instructions cause a hiatus in the time base. This is indicated by drawing a vertical blue line through all the DAC and DIG traces. A Wait also stops all other sequencer activity apart from previously started Sinusoidal output, Ramps and Arbitrary waveform output, which continue during the wait.

You can select the following delay types:

Fixed delay
Set the time to wait in the Delay field. This is useful in sequences with short active periods separated by long delays (so the active periods occupy a reasonable proportion of the visible time base). Even though we know how long the delay is, we treat it as if it were 1 tick when drawing. Another way to achieve this is to use multiple, linked sections; use a long empty section for the delay.

Random delay
This holds the sequence for a time between Min time and Max time seconds or milliseconds. All delays between the minimum and maximum have equal probability (within the capabilities of the sequencer). You will get the best result for delay ranges that are long compared to the time resolution.

Poisson delay
This generates a delay with a Poisson statistic; the delay has the same probability of ending at any time during it. The Time const field sets the average delay length.

Digital input high/low
You can wait for a nominated digital input bit in the range 0-7 to be high or low. These are not the same bits as used for the event inputs (which are digital inputs 8-15). If you want to synchronise to a bit changing you must wait for it to be in the opposite state to the one you want first. If you need the sequence to perform actions while you wait, use a branch item. You can wait for combinations of input bits with the text sequencer.

From [11.00] you can add an optional time out, after which time the sequence will branch to a nominated label or Section. If you do not want a time out, set the field to 0. If you set a time out, the code loop to do this takes two instructions, so the timing accuracy of the change detection is 2 sequencer clock ticks, rather than 1 (without the time out).

Channel above/below/outside/within
You can wait for a nominated waveform channel sampled by the 1401 to be above or below a level set by the Threshold field, or to be outside or within a pair of levels set by the Lower and Upper levels fields. The levels are set in the channel units, as stored in the sampling configuration. If you subsequently change the channel type, the results will not be harmful, but the sequence will not operate as intended!

Next event
You can wait for the number of events set by the Count field to occur on a nominated event, marker or WaveMark channel that is sampled by the 1401. When this delay item is reached, the sequence notes how many events have been sampled on the channel and waits until the number increases by the count.

Time reached
Waits until the sample time reaches the set time. There is no wait if the time has already passed the set value.

Cosine phase 0
This item waits for the next time that cosine output on the nominated DAC channel passes through phase zero.

Variable comparisons
You can wait for various conditions based on sequence variables 1 to 25. The remaining variables are used to implement the sequence sections. Variable values are 32-bit integers. You can manipulate the variables in the Variable arithmetic item. Variables are described in detail for the text sequencer.

Event burst
This item monitors an event, marker or WaveMark channel sampled by the 1401 for a group of events with a user-defined maximum separation. The Intervals field sets the number of gaps to check and the Max time field sets the maximum acceptable interval. If any interval is greater than the maximum, the sequence starts the search again. From the detection of the last event in a burst, there is a latency of 10-12 sequencer instructions until the next sequencer action can occur.

For this to work well, the maximum interval must be significantly greater than the time resolution of the sequence. The total burst duration is limited to 31 bits of sampling clock ticks.

Branch on condition, probability or variable

With this item you can break the normal flow of the sequence and branch to a different section or to a label you have defined for an item in the current section. All branches have a Branch destination field in which you can select a section to branch to, or you can type the name of a label in the current section.

When you branch, the timing to the target may not be exactly what you expect. The sequence will take one or more steps to implement the branch and the target instruction may require preparatory steps. Such effects are small unless you use arbitrary waveform output where the preparatory steps can take several milliseconds. If you need the tightest possible control over branch timing you should consider using the text sequence editor. The branches you can set are:

Probability
Percent sets the probability of the branch, 0% never branches, 100% always branches.

Digital input high/low
The Bit to test field sets the digital input bit number in the range 0-7 to test.

Channel above/below/outside/within
You can branch if a nominated waveform channel sampled by the 1401 is above or below a level set by the Threshold field, or is outside or within a pair of levels set by the Lower and Upper levels fields. The levels are set in the channel units, as stored in the sampling configuration. If you subsequently change the channel type, the result is not harmful, but the sequence will not operate as intended! It takes two (or three for the outside/within cases) sequencer instructions to do the check, so make sure that the sequencer is running fast enough to detect the changes you seek.

Variable comparisons
You can branch on the result of comparing sequence variables with constant values and other variables. Some variables have special uses. Variable values are 32-bit integers. You can manipulate the variables in the Variable arithmetic item. Variables are described in detail for the text sequencer.

Unconditional
This always branches to the destination.

Time comparisons
You can compare a variable plus a time offset with the current time and branch on the result. You can set the variable to the current time (plus a time offset) with the variable arithmetic *current time* instruction.

Response with timeout
You can wait for a new data item in an event, marker or WaveMark channel sampled by the 1401. The branch is taken if a new item is detected within the time out period. The time out period is limited to 31 bits of sampling clock ticks.

Generate digital marker channel event

This adds an event to the Marker channel (if it is enabled). You can set the marker code with the Marker code field or check the Record data box to record the state of digital input bits 0 to 7 (these are not the same bits used for event inputs). The Marker code field should be set to one character or to two hexadecimal digits.

Variable arithmetic

Although the use of variables is more commonly done with the text editor, you can perform basic variable manipulation here. You can use variables 1 to 25. Variable values are 32-bit integers. In all cases, the variable that is changed is set by the Target var field. Where operations involve time (in sampling clock ticks) you must remember that sequencer variables are 32 bits and times can now be up to 64 bits. This means that once the sample time exceeds around 2 billion clock ticks, the values stored will no longer represent the time in a useful way (though you can still use differences of times as long as the difference is less than 2 billion sampling clock ticks). Operations are:

Set to value/variable
Replace the target variable with the contents of the Value field or of a variable.

Add/subtract value/variable
Adds or subtracts the contents of the variable or value.

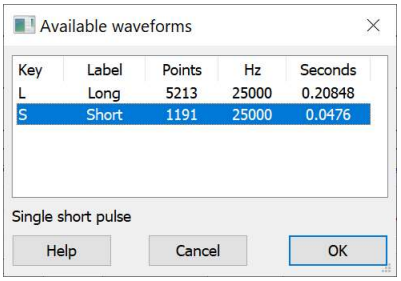
Multiply by value/variable
Multiplies the target variable by the variable or value.

Random value
This replaces the target variable by a random number that is from 1 to 30 bits long, set by the Bits field. The possible values for an `n` bit number are 0 up to 2**n minus 1. For example, if the Bits field is 4, the possible results are 0 to 15.

Current time (deprecated)
The variable is set to the current sample time plus a time offset, in Spike2 clock ticks, as set in the Resolution tab of the Sampling configuration. As the current time can occupy more than 31 bits, the value stored in the variable will, in general, only be useful for the first 2 billion sampling clock ticks. This is retained for compatibility with Spike2 version 7, but should be avoided.

Fixed time
The variable is set to a time, in Spike2 clock ticks, as set in the Resolution tab of the Sampling configuration. As variables can only store positive values up to 31 bits in length, this is only useful for a fixed time that is less than around 2 billion sampling clock ticks. This is here for compatibility with version 7, but should be avoided.

Select arbitrary waveform

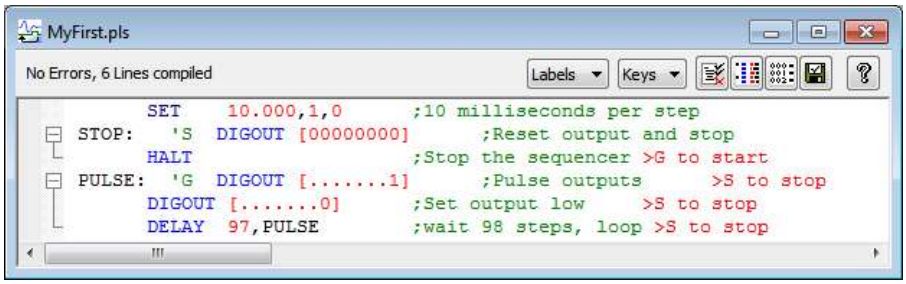


This dialog opens when you drag the arbitrary waveform output icon onto an output track and there is more than one arbitrary waveform to choose from or when you double click an arbitrary waveform output item in an output track. You can add items to this list from the Offline waveform output dialog accessed from the Sample menu or with the `PlayWaveAdd()` script command.

Select one of the waveforms in the list and click OK to add the wave into the sequencer output or click Cancel to close the dialog. You can adjust the length of the played data in the graphical editor. The line of text above the buttons displays the comment associated with the selected item.

The text editor

The output sequencer text editor gives you full control over the output sequencer. The price you pay for this is that you become responsible for calculating the timing of everything. Unlike the graphical editor, where you can deal with items like pulse trains, in the text editor you must construct pulse trains edge by edge; fortunately, this is not as difficult as it sounds. Even if you plan to use the graphical editor, some understanding of how you would program with the text editor is useful as it will help you understand why some features of the graphical editor work in the way they do (as all graphical sequences are converted to text sequences before they are used).

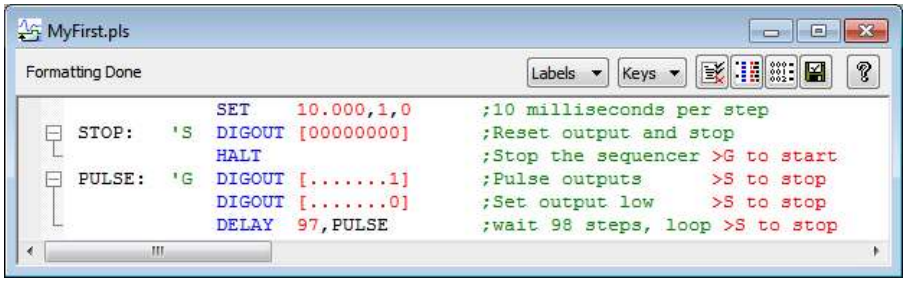


The text output sequence is stored as a text file with the extension `.pls`. From [10.15] you can also store a sequence as text in the sampling configuration. You can open existing Spike2 output sequence files or create new ones with File menu New… You can also open this window by double-clicking on a displayed sequence in the Sampling configuration dialog Sequencer tab.

If the folding margin is displayed as in this example (if not, see the View menu Folding command and select a folding style), you will see that you can fold up the window text based on the keyboard codes as folding points. The Keys and Labels drop downs show a list of all the keyboard codes and labels in the file; select one to navigate to them. If you load a sequence from a read only medium or the file is marked read only, you are not allowed to change it in the editor.

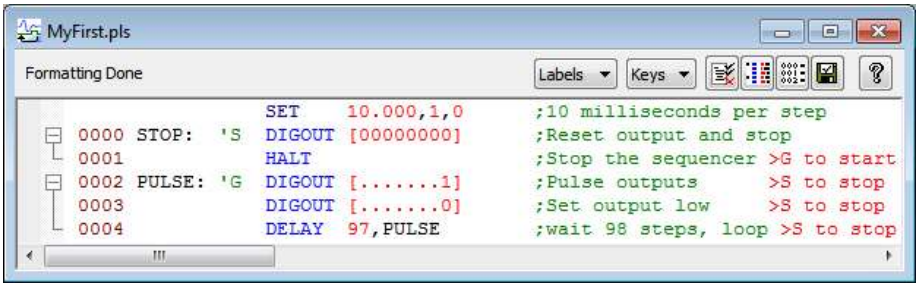
There are five buttons at the top of the window:

Format



This aligns the labels, key codes, instructions and any arguments, output text and comments and removes step numbers. It will also do basic syntax checking of the sequence; undefined labels are not flagged but there must be no other errors. If there are errors, the first offending line is flagged and formatting stops. You will normally want to have your sequences formatted as it makes them much easier to read and understand. However, it is usually much quicker to type sequences in unformatted, then use the Format command to tidy them up.

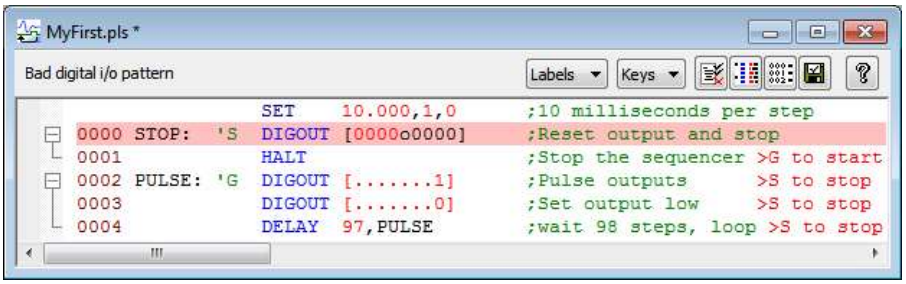
Format with step numbers



This does the same job as the Format button, and also starts each line with the step number. Step numbers can be useful as they give an indication when you are running out of space (there are a maximum of 8191 instructions allowed) and can pinpoint the line where your sequence is not behaving as you expect. When your sequence is running, the sequencer control panel displays the current sequencer step. A number at the start of a sequencer

line is completely ignored when sequences are compiled, so you need not worry about editing a sequence that has step numbers. Just format the sequence to clean up the changes.

Compile



This checks the sequence to make sure that it is correct with no labels missing or duplicated and no duplicated key codes. The picture shows a sequence with a simple error (the o in the second line should be a zero). The line in error is marked by changing the background colour and an explanatory message is shown at the top of the screen. Any edit to the sequence will remove the background colour change. The compiler stops at the first error it finds. If the compiler completes without error is display the number of sequencer lines (instructions) that were compiled. The compiler always adds a `HALT` instruction to the end of the sequence, so the count will be one more than the number of listed lines.

Current

This compiles the sequence and if it compiles without error, the sequence is saved. If there is any error, it is reported and the command ends. If there is no error, what happens next depends on whether Spike is sampling data or not:

Not sampling — The name of the sequence file is set in the sampling configuration and the sequence will be used when you next sample data. You can also set the output sequence file from the Sampling Configuration dialog accessed from the Sample menu. From version [10.15], if the file name is `S2CFGSEQ$.PLS` (this is the name given to temporary files used to edit sequences stored as text in the sampling configuration), this will update the sequence stored as text in the sampling configuration.

Sampling — Spike attempts to replace the current sequence. This will fail if there is no current sequence, or if the sequence or table size is larger than the current size or the size reserved in the Sequencer tab of the Sampling configuration dialog. If a sequence is replaced, it runs from the first instruction. Any variables that are declared with initial values are updated as are all initialised table values. The Sampling configuration is not updated with the name of this sequence file.

You can also change the current sequence file from the Sample menu and with a script.

A message indicating the outcome of the operation is displayed at the top of the window.

Help

This is the Help button. It opens a window holding a list of the sequencer topics for which help is available. You can copy and paste text from the help window into your sequence.

Loading sequencer files for sampling

The name of the output sequencer file to use during sampling is part of the sampling configuration. The file name, including the path to the folder containing it, must be less than 200 characters long. You set the file name either with the Current button, as described above, or in the Sampling Configuration dialog. When you start sampling, Spike2 searches for any output sequence file named in the sampling configuration.

From version [10.15], you also have the option of storing the output sequence text as part of the sampling configuration.

Spike2 compiles output sequence files whenever you use them. If a file contains errors you are warned and the file is ignored.

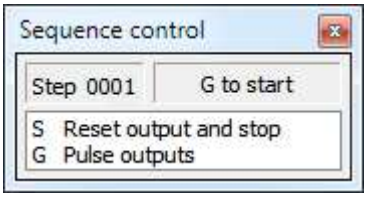
Getting started

The sequencer runs instructions in order unless told to branch. It can be re-routed during data acquisition by associating an instruction with a key on the keyboard. Each time the key is pressed or the associated sequencer control panel button is clicked or the script language `SampleKey()` command is used, the sequencer jumps to the associated instruction. Spike2 records keys pressed during sampling, so you have a record of where in the document you switched to a new portion of the sequence.

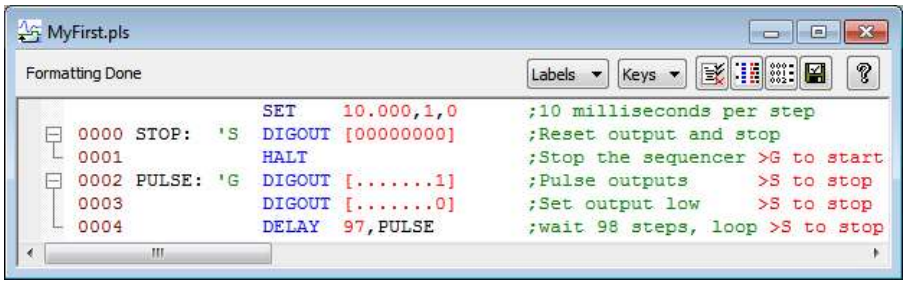
Here is a simple sequence that will pulse digital output bit 0 for 10 milliseconds once per second. You can start and stop the pulses with keys or by clicking buttons. You will find this in the `Sequencer\MyFirst.pls` file (in the User data folder) to save you typing it in.

```
          SET    10            ;10 milliseconds per step
       'S DIGOUT [00000000]    ;Reset output and Stop
          HALT                 ;Stop the sequencer    >G to start
PULSE:  'G DIGOUT [.......1]    ;Pulse outputs         >S to stop
          DIGOUT [.......0]    ;Set output low        >S to stop
          DELAY  97,PULSE      ;wait 98 steps, loop   >S to stop
```

Open `MyFirst.pls` and click the Check and make current sequence button at the upper right of the window (leave the mouse over each button for a second or so to see the descriptive text). Open the Sampling configuration dialog, and set a configuration with one event channel on port 0. To record the output pulses you must connect the digital output to the event input. This is easy with the Micro1401 and Power1401 as you can connect Digital output 0 to Event input 0 on the front panel. You do not need to make the connection to follow this description.



Click the Run now button in the sampling configuration, and then click the Start button. The sequencer control panel is now visible. It displays Step 0001 in the top left window and G to start to the right. If the control panel is docked there are two buttons labelled S and G, otherwise you can see the text S Reset output and Stop and G Pulse outputs. To start the output, click on the G or type `G` on the keyboard. While it runs, the display will change to Step 0004 (with occasional flickers) and S to stop. If you have connected the digital output to the input, you will see your pulses recorded, once per second. Click the S or type `S` on the keyboard to stop the output.



As this is our first sequence we will explain it in detail. Click the Format and add step numbers button at the top of the sequencer window. All the lines get a step number except the first. This is the step number displayed in the control panel. You can remove step numbers with the Format button. Lines that get a number are part of the sequence run by the 1401. The `SET` line tells the sequencer how fast to run, in this case 10 milliseconds per step, and is not part of the sequence run by the 1401. A `SET` or `SCLK` directive is usually the first line in the sequence.

The next line (step 0000) is the first to run when you start sampling. The `'S` means *jump to this line every time the S key is pressed during sampling* (as long as the data window is the current window). To allow you a wide choice of keyboard jumps, lower case s and upper case S are treated as different keys.

The instruction on this line is `DIGOUT [00000000]`, which sets 8 digital outputs (bits 15 to 8 of the digital output) to the low state, nominally 0 Volts. The remainder of the line is a comment. Because there is a keyboard jump set for this line, the comment is also used as a label for the sequencer control panel.

The next line, (step 0001) holds a `HALT` instruction. This stops the sequencer and nothing will happen until the sequencer is told to jump to one of the steps labelled with a key code. The sequencer control panel displays the text to the right of the `>` for the current step. When the sequencer halts, it stays at step 0001, so you see the message G to start.

Steps 0002 and 0003 set digital output bit 8 high (nominally 5 Volts) and low again. The sequencer is running at 10 milliseconds per step, so the pulse is 10 milliseconds wide. Step 0002 starts with a label, `PULSE:`, so we can branch back here in the next step.

Step 0004 has the instruction `DELAY 97,PULSE` to make the sequence wait for 97 extra step times in addition to the step time that every instruction takes, and then branch to the instruction labelled `PULSE`. This instruction takes 980 milliseconds, and together with the 20 milliseconds taken by steps 0002 and 0003, this makes 1000 milliseconds for the loop. Instead of 97 we could have written `s(1)-3`. The `s(1)` function returns the number of sequencer steps in 1 second.

Time delays can also be implemented with the `TICKS` instruction, which gives you access to the underlying tick count used by the Spike2 sampling.

The remainder of this chapter is organised as a reference manual for the sequencer instructions. You can find more information about the output sequencer in the *Spike2 Training Course* manual.

Sequencer compiler error messages

When you use the Format or Compile buttons in the sequence editor, Spike2 displays the result of the compilation or format operation in the message bar at the top of the window. The messages report either successful operation or the cause of the problem.

```
No errors, N lines compiled
```

Your sequence has been checked and is syntactically correct. This means that it will certainly run, but it is up to you to ensure that the sequence of codes produces the desired effect. If your sequence runs off the end of the defined instructions, it will stop as the compiler adds a `HALT` instruction to the end of every sequence. The number of lines compiled is a count of the number of instructions generated, which is one more than the number in the sequence text as Spike2 adds a `HALT` instruction to the end of the sequence.

Formatting done

The format operation has completed and all lines of instructions are syntactically correct in themselves. There is no check that a label referenced in an instruction exists.

Invalid label format, or same as Opcode, VAR, CONST or function

A colon was found, indicating a label, but the characters before the colon are not legal. A label starts with an alphabetic character (`A-Z`) and is followed by alphanumeric characters (`A-Z` and `0-9`) and is terminated by a colon. Case is not significant in labels. The label can be up to 8 characters long. It must not be the same as an instruction name, a variable name, a constant name or a built-in function.

Invalid key definition

A quote mark was found, but there was no acceptable character following the quote, or there was more than one character.

Unknown command used

A group of characters was found in the correct position to be an instruction, but they were not recognised. Change them to a correct instruction mnemonic. This can also be caused by a missing colon at the end of a label.

The first/second/third/fourth argument is invalid

The instruction argument is either missing, invalid or was too long to be correct. If the argument included a table reference, the offset may be too large or too small.

Internal error, contact CED

This is caused by a serious system error. If you can reproduce this problem please contact CED for advice.

Out of memory

Spike2 has run out of memory while processing the sequence. Close any non-essential windows in the Spike2 application and try again. If this does not allow compilation, close any other applications and retry. As you are most unlikely to exhaust your memory when compiling it may mean that something has gone badly wrong.

Label or table size defined twice
A label has been defined on more than one line. Remember that the labels are converted to upper case, so two labels that differ in case only will cause this error. This error is also given if you use `TABSZ` more than once in a sequence.

Unexpected characters at end of line
The most common cause of this error is a missing semicolon to introduce a comment.

Couldn't find branch destination
The label used by the instruction is not defined anywhere in the sequence. Define the label, or correct the label name.

Invalid numeric value
A numeric argument was expected, but an invalid number or one that was outside the permitted range for the argument was found.

Probability out of range
The probability field of the `BRAND` instruction must be set to a number in the range 0 to less than 1.0000000000; correct the number.

Bad digital i/o pattern
Whenever a digital i/o pattern is expected, exactly 8 characters must be present, one for each bit of the digital i/o. The characters are restricted to "`0`", "`1`" and "`.`", plus "`c`" for `DIGIN` and "`i`" for `DIGOUT`.

A label, key or display string but no instruction
Labels, key definitions and display strings may only occur on lines that have an instruction. If you really want an instruction that does nothing, use `NOP`. Blank lines and lines consisting entirely of comments are allowed and ignored.

Too many instructions
You have defined more than the maximum allowed number of instructions (8191), so you must shorten your sequence. You may be able to significantly shorted your sequence by using variables and the `SampleSeqVar()` script instruction. You can also split a sequence into sections and load these one at a time.

Input line too long; shorten and try again
Each line of the file must be no more than 100 characters long. Shorten the offending line and try again. This limit in old versions of Spike2 was less, so it is a good idea to keep to short lines wherever you can.

Unexpected end of file encountered
This shouldn't happen and probably means that something has gone badly wrong.

Variable name too long, unknown, badly formed or missing
A variable name used to replace the `V1 to V256` built-in names is incorrectly formed.

Please use a single character or 2 hexadecimal digits as wave code
The waveform codes for the `WAVEGO` command must either be a single character or two hexadecimal digits. A hexadecimal digit is one of the characters 0-9 or a-f.

Bad branch code: C=Cycles, A=Area, S=Stopped, T=Triggered, W=Wait for WAVEGO
The branch code for the `WAVEBR` command was not one of the allowed characters listed, or the branch code was more than one character.

Bad start or stop code: S=Stop now, C=one more cycle, T=Trigger now
The code given for the `WAVEST` command was not one of those listed, or there was more than one character in the code.

Bad flags: T=Triggered, W=Wait until hardware is ready
The flags for the `WAVEGO` command were not those listed, or more than two flags were used.

Variable name already defined as Opcode, label, function or CONST
Old versions of Spike2 allowed a variable name to be the same as a label or an `Opcode`. This is no longer allowed. Also, the name is not allowed to the same as a built-in function, for example `VAngle` or the same as a `CONST` item. Change the variable name.

Label already exists as Opcode, variable, function or CONST
Old versions of Spike2 allowed a label to be the same as a variable name or an instruction name. This is no longer allowed. The name may not match a built-in function or a constant. Change the label.

The DAC value exceeds the DAC output range
You have entered an expression for a DAC value that when scaled and offset by the values in the `SET` directive, produces a result that exceeds the DAC range. Check that the `SET` directive scale and offset are correct and that you have not mistyped the value.

More table entries than set by TABSZ
You have used the `TABDAT` directive to create table entries. There is either no `TABSZ` directive, or you have generated more table data than you allocated with `TABSZ`.

Compatibility with previous versions

Sequences from all previous versions still work, with the following provisos on sequences written for Spike2 version 3 and earlier:

1. Variables and labels may not share names nor be the same as instruction codes.
2. `DELAY 0` no longer hangs up for a long time. It now has the same effect as `NOP`.
3. If a version 3 sequence relied on looping back to the start after 256 steps, this no longer works as all sequences have a `HALT` instruction automatically added to the end. You must add a label at the start and a `JUMP` instruction at the end.
4. `BRAND` set the probability to an accuracy of one part in 256. It now sets the probability to a very high accuracy. This might affect your results.
5. `DACn` and `ADDACn` now insist that any value you use lies within the range of the DAC.
6. `ADDACn` previously expressed your increment as a 16-bit integer. It now generates a much more accurate 32-bit integer. However, this means that ramps generated with `ADDACn` may have a different slope (probably much closer to the intended slope).

If you want to write a sequence that will run on previous versions of Spike2, either refer to the manual for the previous version, or write the sequence using the previous version to be certain of complete compatibility. Use of any of the following features will prevent the sequence working on version 6 or earlier:

1. Use of more than 1023 sequencer instructions or more than 64 variables.
2. Use of the `ASz()` or `VSz()` expressions.
3. Use of a label in an expression
4. Use of the `JUMP (Vn)` or `JUMP LB(Vn)` instructions
5. Use of new instructions: `TABADD`, `TABSUB`, `ABS`, `AND`, `ANDI`, `OR`, `ORI`, `XOR`, `XORI`, `DIGPC`, `DIGPS`, `DIGPBR`
6. Use of `#include` to include other files.
7. Use of the = directive to define a constant.
8. Use of more than DACs 0-3 in the sinusoidal output or `RAMP` instructions.

If you use the `sTk64h()` or `sTk64l()` instructions or refers to `VTick0L` or `VTick0H`, your sequence will only run in version 8 onwards.

Instructions

These are the Spike2 output sequencer instructions. Instructions in brackets are obsolete and should be avoided in new sequences.

Digital (TTL compatible) input and output

`DIGOUT`	Write to digital output bits 15-8

DIGLOW	Write to digital output bits 7-0
DIBNE,DIBEQ	Read digital input bits 7-0, copy to V56 test and branch
DISBNE,DISBEQ	Test last read digital inputs (in V56) and branch
WAIT	Wait for a pattern on the lower 8 digital inputs
(DIGIN)	Read and test digital inputs
(BZERO,BNZERO)	Branch as result of DIGIN test

DAC (waveform/voltage) outputs

DAC	Set DAC value (for DACs 0-7)
(DACn)	Version 3 compatible, set DAC n (0-3) to a value
ADDAC	Increment DAC by a value
ADDACn	Version 3 compatible, increment DAC n (0-3) by a value
RAMP	Set automatic DAC ramping to a target value

Sinusoidal waveform output

SZ	Set the cosine output amplitude (CSZ,DSZ)
SZINC	Change the cosine output amplitude (CSZINC,DSZINC)
RATE	Set the cosine angular increment per step (CRATE,DRATE)
RATEW	As RATE, but waits for phase 0 (CRATEW,DRATEW)
ANGLE	Set cosine angle for the next step (CANGLE,DANGLE)
WAITC	Branch until cosine phase 0 (CWAIT,DWAIT)
RINC	Change the cosine angle increment per step (CRINC,DRINC)
RINCW	As RINC but waits for phase 0 (CRINCW,DRINCW)
PHASE	Defines what phase 0 means (CPHASE,DPHASE)
OFFSET	Offset for sinusoidal output (COFF,DOFF)
CLRC	Clear the new cycle flag (CWCLR,DWCLR)

General control

DELAY	Do nothing for a set number of steps
DBNZ	Decrement a variable and branch if not zero
(LDCNTn,DBNZn)	Load counter 1 to 4 (V61-V64), decrement, branch if not zero
Bxx	Compare variables and branch (xx = GT, GE, EQ, LE, LT, NE)
CALL	Branch to a label, save return position
CALLV,(CALLn)	Like CALL, but load a variable (counter 1-4) with a value
RETURN	Branch to instruction after last CALL, CALLV or CALLn
JUMP	Unconditional branch to a label
HALT	Stops the sequencer and waits to be re-routed
NOP	This does nothing for one step (No OPeration)

Variable arithmetic

ABS	Take the absolute value of a variable
ADD	Add one variable to another
ADDI,(ADDIL)	Add a constant value to a variable
MOV	Copy one variable to another
MOVI,(MOVIL)	Move a constant value into a variable
MUL,MULI	Multiply two variables, multiply by a constant
NEG	Move minus the value of a variable to another
SUB	Subtract one variable from another
DIV,RECIP	Division and reciprocal of variables

Variable logic

AND, ANDI	Bitwise AND of variables, variable and constant
OR, ORI	Bitwise OR of variables, variable and constant
XOR, XORI	Bitwise exclusive OR of variables, variable and constant

Table support

TABLD,TABST	Load a register from the table and store a register to the table
TADADD,TABSUB	Add a table value to a variable or subtract it from a variable
TABINC	Increment a register and branch while within the table

Access to data capture

REPORT,MARK	Simulate an external E1 pulse to record/set a digital marker
CHAN	Get the latest waveform value or event count from a channel

TICKS	Load a variable with the time in Spike2 time units

Randomisation

BRAND,(BRANDV)	Random branch with a probability
MOVRND	Load a variable with a random number
(LD1RAN)	Load counter 1 (V61) with a random number 1-256

Arbitrary waveform output

WAVEGO	Start or prepare arbitrary waveform output area
WAVEBR	Test arbitrary waveform output and branch on the result
WAVEST	Start or stop arbitrary waveform output

Instruction format

Blank text lines and lines with a semicolon as the first non-blank character, are ignored. Instructions are not case sensitive. Each instruction has the format:

```
num lab: 'key  code arg1,arg2,...  ;Comment >display
```

`num` An optional step number in the range 0 to 8190, for information only. A number at the start of an instruction line is ignored.

`lab:` An optional label, up to 8 characters long followed by a colon. The first character must be alphabetic (A-Z). Labels are not case sensitive. Labels may not be the same as instruction codes, variable names, constants or expression function names. Labels can be used in expressions, and have the value of the instruction number in the sequence; the first instruction is number 0. Labels are used by branching instruction to change the flow of control through a sequence. From version [11.00] we allow branches to use relative jumps by specifying `+n` or `-n` (`n` is the number of sequencer steps to jump) in place of a label.

`'key` In this optional field, `key` is one alphanumeric (`a-z`, `A-Z`, `0-9`) character. When this character is recorded as a keyboard marker during data capture, the sequencer jumps to this instruction. Each `key` can occur once. Upper and lower case are distinct. The `key` appears in the sequencer control panel together with any `comment`. The key must be followed by white space and a `code`. For reasons of backwards compatibility, we also accept `:key` and `lab:key`, however this syntax is deprecated and converted to `'key` when the line is formatted.

`code` This field defines the instruction to be executed. It is not case sensitive.

`arg1,…` Instructions have up to 4 arguments that are separated by commas or spaces. These are described with the instructions. If an argument can be represented in different ways, they are separated by vertical bars (read as "or"), for example: `expr|Vn|[Vn+off]`. In this case, the argument can be an expression, a variable or a table reference.

`comment` The text after the semicolon is to remind you of the reason for the instruction. If a `key` is set, this comment also appears in the sequencer control panel.

`>display` When a sequence runs, text following a `>` in a comment is displayed in the sequencer control panel to indicate the current instruction. Spike2 is notified every few milliseconds of the current instruction and the control panel is updated whenever there is free time, so the display is only a sampling of what is going on. The special combination `>"` means use the same display text as for the previous instruction. This can save you some typing when a group of instructions in a loop need the same display. If there are several instructions in a row that use this, Spike2 searches backwards until it finds an instruction with a display string that is not `>"`. The special combination `>=` means make no change to the display string. This might be used in a code section that is called as a subroutine from several different places.

Expressions

Many instructions allow the use of an expression in place of a constant value, indicated by `expr`. An expression is formed from constants, numbers, labels, round brackets `(` and `)`, the operators `+`, `-`, `*` and `/`, and sequencer expression functions. Labels have the value of the instruction number in the sequence; the first instruction is numbered 0.

The operators * and / (multiply and divide) have higher priority than + and − (add and subtract). This means that 1+2*3 is interpreted as 1+(2*3) and not as (1+2)*3. Apart from this, evaluation is from left to right unless modified by brackets.

The sequence compiler evaluates expressions as real numbers, so 3/2 has the value 1.5. If expr is used as an integer, for example DELAY expr, it is rounded to the nearest integer. Values in the range 3.5 to 4.49999… are treated as 4. Before version 4.06 the result was truncated, so 3.0 to 3.9999… was treated as 3. From [11.00] there are new expression functions Ceil(), Floor(), Round() and Trunc() to force how fractional numbers are treated.

You cannot use sequencer variables (Vnn) in an expression. Variables exist when a sequence runs within the 1401 interface. Expressions are evaluated when a sequence is compiled inside your computer.

Sequencer expression functions

These functions can be used as part of expressions to give you access to Spike2 clock and sequencer step timing and to convert between user units and DAC and ADC values and to calculate table indices.

s(expr)
ms(expr)
us(expr)

The number of sequencer steps in expr seconds, milliseconds or microseconds. For example, with a step size of 200 milliseconds, s(1.1) returns 5.5. This is often used with the DELAY instruction. Each instruction uses 1 step, so use DELAY s(1)-1 for a delay of 1 second. These expressions assume that the sequencer runs at the rate set by the SET or SCLK directives at the start of the sequence.

sTick(expr)
msTick(expr)
usTick(expr)

The number of Spike2 sample clock ticks in expr seconds, milliseconds and microseconds. The result is in the same units as returned by the TICKS instruction, so TICKS V1,sTick(1) sets V1 to the value that TICKS will return 1 second later. The result is a signed 32-bit value, so is limited in magnitude to 2 billion (2000000000) sample clock ticks. If you run sampling at a resolution of 1 microsecond, this is a range of -31 minutes to 31 minutes. At 2 microsecond resolution it is plus or minus an hour, and so on. These expressions assume that the sequencer runs at the rate set by the SET or SCLK directives at the start of the sequence.

sTk64h(expr)
sTk64l(expr)

[8.00]. These converts a time in seconds into Spike2 sample clock ticks as a 64-bit number, then returns either the upper 32 bits (sTk64h) or the lower 32 bits (sTk64l). If you need to do 64-bit arithmetic with these values you should be aware that the lower 32 bits is effectively an unsigned number, so if you add a 32-bit value to it you may need to detect if the result overflowed, and if it did overflow add 1 to the upper 32 bits. There are currently no 64-bit arithmetic instructions in the sequencer instruction set, but you can emulate them (with some effort) by looking at the signs of the values before and after addition.

Hz(expr)

The angle change in degrees per step for a cosine output of expr Hz. For a 2 Hz cosine on DAC n, use RATE n,Hz(2). To slow the current rate down by 0.1 degrees per step use RINC n,Hz(-0.1). Use in RATE, RATEW, RINC and RINCW instructions.

VHz(expr)

The same as Hz(), but the result is scaled into the 32-bit integer units used when a variable sets the rate. MOVI V1,VHz(2) followed by RATE n,V1 will set a 2 Hz rate.

VAngle(expr)

Converts an angle in degrees into the internal angle format. The 32-bit integer range is 360 degrees. The result is expr * 11930464.71. For use with ANGLE and PHASE.

VDAC16(expr)

Converts expr user DAC units so that the full DAC range spans the full range of a 16-bit integer. Use with variables for the obsolete DACn and ADDACn. For example, MOVI V1,VDAC16(2.5) then DAC0 V1 has the same effect as DAC0 2.5.

VDAC32(expr)

Converts expr user DAC units into a 32-bit integer value such that the full DAC range spans the 32-bit integer range. Use this to load variables for the DAC and ADDAC instructions. For example, MOVI V1,VDAC32(2.5) then DAC 0,V1 has the same effect as DAC 0,2.5 (setting the DAC 0 output to 2.5 user DAC units).

ASz(expr)

Converts expr user DAC units into a value suitable for use with the SZ and SZINC commands. That is, it converts an amplitude or amplitude change in user DAC units to a value in the range 0 to 1. It does this by dividing the value you supply by the (DAC scale factor * 5).

VSz(expr)

Converts expr user DAC units into a variable value suitable for use with the SZ and SZINC commands. That is, it converts an amplitude or amplitude change in user DAC units to a

value in the range 0 to 32768. It does this by multiplying the value you supply by 32768/ (DAC scale factor * 5).

`TabPos()` The number of table data items defined at this point in the sequence by `TABDAT`. This is also the index of the next table data item to use. You can use this to define constants that reference table indices.

`DRange()` The DAC output range (usually 5.0 for a ±5 Volt system or 10.0 for a ±10 Volt system). When the sequence is built for use during sampling, this value is taken from the DAC output range of the attached 1401, otherwise, this is taken from the Voltage range set in the Edit menu Preferences option.

`Ceil(expr)` Integral values are unchanged, but all non-integral values are set to the next higher integral value. Think *ceiling*.

`Floor(expr)` Integral values are unchanged, but all non-integral values are set to the next lower integral value.

`Round(expr)` Round `expr` up or down to the nearest integral value.

`Trunc(expr)` Remove the fractional part of `expr`.

Used with care, the built-in functions allow you to write sequences that operate in the same way regardless of the sequencer step time or DAC scaling values. Note that `Ceil()`, `Floor()`, `Round()` and `Trunc()` were added at version [11.00].

Variables

You can use the 256 variables, `V1` to `V256`, in place of fixed values in many instructions. In the sequencer command descriptions, `Vn` indicates the use of a variable. Where a variable is an alternative to a fixed value expression we use `expr|Vn`. Variables hold 32-bit signed integer numbers that you can set and read with the `SampleSeqVar()` script command. There are instructions that can be used to perform arithmetic on variables. If you need more than 256 variables, you could consider using a Table.

Variables exist in memory in the 1401. Scripts can set their values and read them back, which involves the transfer of data between the 1401 and the host computer, typically taking a few milliseconds to do. Variables allow you to perform calculations and store and recall information with a running sequence. You can also use them to control a running sequence.

When an instruction includes a shift of a variable (described as `>>` **`shift`**), the shift is arithmetic (that is it preserves the sign).

Variables with pre-defined uses

Some variables have specific uses and pre-defined symbolic names (which are easier to remember than `V1` to `V256`). If you use these variables for other purposes you can rename them (see below), but once renamed, the pre-defined name is no longer available.

Variable	Pre-defined name	Comment
`V56`	`VDigIn`	This holds the last bit pattern read from the digital inputs with the `DIBxx` or `DIGIN` instructions.
`V57-V64`	`VDAC0-VDAC7`	These hold the last value written by the sequencer to DACs 0-7. These variables are not updated by arbitrary waveform output so if you use it, you cannot rely on `VDACn` for the current DAC value.
`V61-V64`		These are used to emulate the counters for the obsolete `LDCNDn and DBNZn` instructions (which you should NOT use). If you avoid these instructions, the variables are free for use. This use overlaps with the `VDAC4-VDAC7` use.
`V255`	`VTick0L`	This holds the first (lower) 32-bits of the zero time used by the `TICKS` instruction and set by the `TICK0` instruction. This is an unsigned 32-bit number, which makes it tricky to use in comparisons.

V256	VTick0H	This holds the second (upper) 32-bits of the zero time used by TICKS and set by TICK0.

Variables V57 through V64 hold the last value written by the sequencer to DACs 0 through 7 (and can also be referred to as VDAC0 through VDAC7), V56 holds the last bit pattern read from the digital inputs with the DIBxx or DIGIN instructions, V61 through V64 are also used to emulate the V255 and V256 store the zero time used by the TICKS instruction and are set by the TICK0 instruction.

VAR directive

You can assign each variable a name and an initial value with the VAR directive. Names must be assigned before they are used, usually at the start of the sequence. Each variable can be assigned a name once only, and the name must be unique and not the same as a label, a constant, an expression or an instruction. The syntax is:

```
VAR    Vn,name=expr          ;comment
```

VAR does not generate any instructions. It makes the symbol name equivalent to variable Vn and sets the initial value when the sequence is loaded. Anywhere in the remainder of the sequence where Vn is acceptable, name can be used. name can be up to 8 characters, must start with an alphabetic character and can contain alphabetic characters and the digits 0 to 9. Variable names are not case sensitive. A variable name must not be the same as an instruction code, a constant or a label.

There is no need to specify a name or an initial value. If no initial value is set, a variable is initialised to 0 even if not included in a VAR statement. Spike2 automatically assigns V56 the name VDigIn and variables V57 through V64 the names VDAC0 through VDAC7. If no symbolic name is set, any pre-defined name for the variable is cleared. The following are all acceptable examples:

```
VAR    V1,Wait1=ms(100)    ;Set name and initial value
VAR    V2,UseMe            ;Set name only, so value is 0
VAR    V3=200              ;No name, initialise to a value
VAR    V4                  ;No name, initialised to 0
VAR    V56                 ;No name, the pre-defined VDigIn is cleared
```

When a variable is used in place of a bit pattern in a digital input or output instruction, bits 15 to 8 and bits 7 to 0 have different uses. In the expressions that describe these operations we write Vn(7-0) and Vn(15-8) to describe which bits are used. BAND means bitwise binary AND (if both bits are 1, the output is 1, otherwise 0), BXOR means bitwise exclusive OR (if both bits are different the output is 1, otherwise 0)

When a variable is used in place of a voltage value in a DAC or ADDAC output instruction, the full 32-bit range of the sequencer variable value corresponds to the full range of voltages that can be generated by the DAC, so -2147483648 corresponds to the lowest possible output voltage (-5 or -10 volts), 0 corresponds to a 0 output and 2147483647 corresponds to the highest possible output voltage.

When used in the DACn or ADDACn instructions, only the lower 16 bits of each value are used, so the value -32768 (0x8000) corresponds to the lowest possible output, 0 corresponds to a 0 output and 32767 (0x7fff) corresponds to the highest output. Adding multiples of 65536 (0x10000) to the variable value has no effect on the output.

When used in one of the cosine output angle instructions, the 32-bit variable range from 2147483648 to 2147483647 represents -180 up to +180 degrees. The VarValue script in the Scripts folder calculates variable values for the digital and the cosine instructions.

Prior to version 7, you could change the name of a variable part-way through a sequence. From the point of the change, the old name was hidden. This now generates a multiply defined error.

Script access to variables

Scripts can set and read the variable values with the SampleSeqVar() script command. You can set initial values from the script as long as you set the values after you create the new data file, but before you start sampling. Values set in this way take precedence over values set by the VAR directive.

Constants

It is often useful to define a numeric value as a named constant. For example, when referencing a table value [V1+Slope] is easier to understand than [V1+23]. It also helps to maintain code; if you need to change a numeric value that is used often make it a named constant and just change the constant once. The = directive

does not generate any instructions, it just makes a name equivalent to a number, and the name can be used anywhere a numeric expression is valid. The syntax is:

```
name = expr             ;comment
```

The `name` must start with an alphabetic character and can contain alphabetic characters and the digits 0 to 9. The name can be up to 8 characters long. Numeric constant names are not case sensitive and must not clash with label, instruction, variable or built-in function names. Examples of use:

```
Wait1  = ms(100)
Wait2  = Wait1*1.3
       DELAY  Wait1-1  ;Use constant in an instruction
```

The expression values are calculated and stored as floating point numbers. If they are used in a context that requires an integer value, the fractional part of the number is ignored.

The = directive was added at version 7.00.

The SET SCLK and SDAC directives

These directives control the interval between sequencer clock ticks and the DAC output units. Directives are not part of the sequence and do not occupy a step. These directives should occur before any step-generating code. The `SCLK` and `SDAC` directives were added at version [6.03] to separate setting the tick rate from setting the DAC output units.

```
       SET    msPerStep, DACscale, DACoffset
       SCLK   msPerStep
       SDAC   DACscale, DACoffset
```

The `msPerStep` field sets the milliseconds per step in the range 0.004 to 3000. The table shows the minimum step times and timing resolution for each type of 1401. To ensure accurate timing, `msPerStep` must be an integral multiple of Resolution or Spike2 will not sample. This is checked each time Spike2 samples data.

	Power3	Power1-2	Micro2-3	Micro4
Minimum step (ms)	0.004	.010	.010	0.004
Resolution (ms)	.001	.001	.001	.001

If there is no `SET` or `SCLK` directive, the sequence runs at the default rate, which is 10 milliseconds per step unless the script command `SampleSeqClock()` has changed it. If there is no `SET` or `SDAC` directive, `DACScale` is 1 and `DACOffset` is 0 for a ±5 Volt system and `DACScale` is 2 for a ±10 Volts system. The sequencer expression functions `s()`, `ms()`, `us()`, `Hz()` and `VHz()` use the `msPerStep` value. If you use these functions, the `SET` or `SCLK` directive must occur first in the sequence.

DAC scaling

The DACs in the 1401 are implemented so that the full range maps onto the full range of 16-bit signed integer numbers (-32768 to +32767). If you have 16-bit DACs a change of 1 changes the output by the smallest step possible. With 12-bit DACs (Micro1401 mk II), the smallest step is a change of 16. For most purposes, it is easier to work in units such as Volts or millivolts rather than these DAC units. However, the 1401 works in DAC units and if you set DAC values using variables, the variable values are based on DAC units. In the `SET` and `SDAC` directives, the `DACscale` and `DACoffset` fields define the conversion from user units into DAC units. The standard values of 1.0 for the scale and 0.0 for the offset make the full scale DAC values run from –5 to +4.99985. As most systems have ±5 Volt analogue systems, the standard scale and offset let you work with the DACs in Volts.

`DACScale` The number of user units that correspond to an output change of 6553.6 DAC units (1 Volt for a ±5 Volt system, 2 Volts for a ±10 Volt system). The standard value 1.0 is used if you omit `DACScale`. To work in millivolts, set the scale to 1000.

`DACOffset` The user units that correspond to 0 DAC units (0 Volt output). The standard value 0.0 is used if you omit `DACOffset`.

Please remember that `DACScale` and `DACOffset` do not change the DAC outputs in any way. They are a convenience to allow you to enter values in units that are appropriate to your application. The sequencer expression functions `VDAC16()` and `VDAC32()` use `DACScale` and `DACOffset`, so they must come after the `SET` or `SDAC` directives.

For example, consider the case where the DACs drive a patch clamp amplifier where a change of 2.5 Volts into the amplifier causes a 200 mV potential at the cell and 0 Volts into the amplifier is 0 mV at the cell. For a ±5 Volt system, 2.5 Volts is 16384 DAC units, so `DACScale` is `200 * 6553.6/16384`, which is 80. `DACOffset` is 0, as an output of 0 produces 0 mV at the cell.

If you have both ±5 and ±10 Volt systems, and you want to specify the output in Volts, you can use:

```
SDAC   DRange()/5, 0
```

which will set the `DACScale` value to 1.0 on a ±5 Volt system and to 2.0 on a ±10 Volts system.

Table of values

From Spike2 version [5.06] onwards the sequencer supports a table of 32-bit values.

Declaring the table

You declare that a table exists with the `TABSZ` directive, which normally occurs at the start of your sequence:

```
TABSZ  expr
```

Where `expr` is an expression that sets the number of items in the table. The table size must evaluate to a number in the range 1 to 1000000. Each table item is a 32-bit integer and uses 4 bytes of 1401 memory. The maximum size in a 1401 with 1MB of memory, and assuming that there is no arbitrary waveform output, is around 150000 items. If your 1401 has more memory, you have potentially more table space. The first table item has an index number of 0, the second item is index 1, and so on.

Setting table data

From the script language you can move data between an integer array and the table with the `SampleSeqTable()` function. You can also preset table data from the sequence with the `TABDAT` directive, which must come after the `TABSZ` directive:

```
TABDAT expr
TABDAT expr,expr,expr...
```

Where `expr` is an expression that evaluates to a 32-bit integer. Each `TABDAT` directive adds data to the table, starting at the beginning. The sequencer compiler will flag an error if you define more data that will fit in the table. Table data declared in this way is stored separately from the sequence and is transferred to the 1401 when you create a new data file to sample. If you do not set the table data with the `TABDAT` directive or from a script, the values in the table are undefined.. The `TabPos()` expression has the value of the number of data items that have been defined at the point where it is used. For example:

```
TABDAT TabPos() ;a table item that holds its own index.
```

Accessing table data

Although you can move data between one of the variables and the table with the `TABLD` and `TABST` instructions, many instructions access the table directly. It takes more time to use a table than to use a variable (this is not usually significant).

All references to the table use the contents of one of the variables as an index into the table plus an optional offset as: `[Vn]` or `[Vn+off]` or `[Vn-off]`. The offset `off` is an expression that evaluates to a number in the range –1000000 to 1000000. For example, if `V1` holds 10, `[V1]` refers to the contents of index 10, `[V1-10]` refers to index 0 and `[V1+10]` refers to index 20. Out of range table indices read 0 and are non-destructive.

The `TABINC` instruction makes it easy to increment a variable used as a table index and branch until the increment generates an index outside the table. The following example generates five DAC outputs at 5 different intervals:

```
            SET    1,1,0               ; 1ms per step, normal scales
            TABSZ  10                  ; table of 10 items
oMs         =      TabPos()            ; offset to milliseconds
            TABDAT ms(1000)-3          ; 1000 ms in sequencer steps-3
oVolt       =      TabPos()            ; offset to Volts
            TABDAT VDac32(1)           ; 1 Volt
oNext       =      TabPos()            ; offset to next set of entries
            TABDAT ms(100)-3,VDac32(2)  ;  100 ms, 2 Volts
```

```
            TABDAT ms(50)-3,VDac32(3)   ;   50 ms  3 Volts
            TABDAT ms(500)-3,VDac32(-1) ;  500 ms -1 Volt
            TABDAT ms(200)-3,VDac32(0)  ;  200 ms  0 Volts

            MOVI   V1,0                 ; use V1 as table index, set 0
TLOOP:      DELAY  [V1+oMs]             ; programmed delay
            DAC    0,[V1+oVolt]         ; set DAC 0 to the value
            TABINC V1,oNext,TLOOP       ; add 2 to V1, branch if in table
```

In this case, we could just as easily have set `oMs` to 0, `oVolt` to 1 and `oNext` to 2, or used the values 0, 1 and 2 in the code. However, by using constants, we make it easy to extend the number of items per step. For example, if we decided to insert a new value in the table, making three items per step, we could do so without having to work through our code to check if each occurrence of 0, 1 or 2 was correct or needed changing.

Long data sequences

To output a very long data sequence, you can use the table as a buffer. The basic idea is to divide the table into two halves and use the `SampleSeqTable()` script command to transfer new data into the half of the table that the sequence is not using. To find out where the sequence has reached, look at the value of the variable used as an index with `SampleSeqVar()`. Set a large enough table size so that the time taken to use half the table is several seconds.

Including files

There are times when you will want to reuse definitions or code in multiple projects. You could do this with copy and paste, but it can be more convenient to use `#include` to include files into a sequence. A file that is included can also include further files. We call these nested include files. Only the first `#include` of a file has any effect. A subsequent `#include` of the same file is ignored. This prevents problems with files that include each other and stops multiple definitions when two files include a common file. The `#include` command must be the first non-white space item on a line. There are two forms of the command:

```
#include "filename" ;optional comment
#include <filename> ;optional comment
```

where `filename` is either an absolute path name (starting with a `\` or `/` or containing a `:`), for example `C:\Sequences\MyInclude.pls`, or is a relative path name, for example `include.pls`. The difference between the two command forms lies in how relative path names are treated. The search order for the first form starts at item 1 in the following list. The search for the second form starts at item 3.

1. The folder where the file with the `#include` command lives. If this fails...
2. The folder of the file that included that file up to the top of the list of nested include files. If this fails...
3. Any `\include` folder in the `Spike11` folder inside your `My Documents` folder . If this fails...
4. Any `\include` folder in `\Users\Public\Public Documents\Spike11Shared`. If this fails...
5. Any `\include` folder in the folder in which Spike2 is installed. If this fails...
6. Search the current folder.

Included files are always read from disk, even if they are already open. If you have set the Edit menu Preferences option to save modified scripts and sequences before running, modified include files are automatically saved when you compile. If this option is not set, the output sequencer compiler will stop with an error if it finds a modified include file. You must save the included file to compile your sequence.

There are no restrictions on what can be in an included file. However, they normally contain constant and variable definitions and possibly user-defined code. It is usually a good idea to put all `#include` commands at the start of a sequence file so that anyone reading the source is aware of the scope of the sequence.

The `#include` command for output sequences was added to Spike2 at version 7.00 and is not recognised by any version before this. A typical file using `#include` might start with:

```
#include <sysinc.pls>          ;my system specific includes
#include "include/proginc.pls" ;search relative to source folder
  set 1,1,0                    ;start of my code...
```

Opening included files

If you right-click on a `#include` command line, and Spike2 can locate the included file, the context menu will hold an entry to open it. The search for the file follows that described above, except that it omits step 2.

Errors in included files

If an error is detected during the compilation of an included file, an error message is displayed at the top of the original window indicating which included file has a problem, and the included file is opened (if it can be found) and the offending line is highlighted.

Changing the sequence during sampling

As long as sampling starts with a text or graphical sequence running, it can be replaced during sampling. The Sampling menu Change Output Sequence command lets you choose a text sequence file on disk and the Current button in the Sequence editor sets the current document as the sequence. When sampling starts, Spike2 allocates sufficient space in the 1401 for the first sequence and for any table space it uses. If you want to load sequences during sampling that are larger or needs more table space you must reserve space using the Sequencer tab of the Sampling configuration dialog.

When a new sequence loads, all DAC values, digital output values, variable and table values are preserved except variables that are given initial values by the `VAR` directive and table values that are given values with the `TABDAT` directive.

Sequencer instruction reference

Each instruction is followed by an example. The examples show a preferred format, however the system is flexible. For example, a comma should separate arguments, but a space is also accepted. The patterns used for digital ports should be enclosed by square brackets, however you may omit the brackets if you wish.

Many of these instructions allow you to use a variable or a table entry in place of an argument. In this case, the alternatives are separate by a vertical bar, for example:

```
        DELAY   expr|Vn|[Vn+off],OptLB
```

This means that the first argument can be an expression, a variable or a table entry. There is no explicit documentation for the use of the table, except in `TABLD and TABST`. Where table use is allowed it is written as `[Vn+off]`. If you use a table value in an instruction, the effect is exactly the same as using a variable with the same value as the table entry.

Changes at [11.00]

You can now branch and jump relative to the current instruction `+n` or `-n` (`n` is the number of steps) in place of a label name to branch to n steps after or before the current step. To loop on the current instruction you can use `+0` or `-0`. This can be used anywhere that an instruction description uses `LB` or `OptLB`.

There are new expression functions `Ceil()`, `Floor()`, `Round()` and `Trunc()`.

Digital I O

These instructions give you control over the digital output bits and allow you to read and test the state of digital input bits 7-0.

`DIGOUT`	Write to digital output bits 15-8
`DIGLOW`	Write to digital output bits 7-0
`DIBNE,DIBEQ`	Read digital input bits 7-0, copy to `V56` test and branch
`DISBNE,DISBEQ`	Test last read digital inputs (in `V56`) and branch
`WAIT`	Wait for a pattern on the lower 8 digital inputs
`(DIGIN)`	Read and test digital inputs
`(BZERO,BNZERO)`	Branch as result of DIGIN test
`DIGPS`	Play a digital pulse sequence independently of the sequencer

DIGPC	Control the sequence started by DIGPS.
DIGPBR	Branch on the state of the DIGPS sequence

DIGOUT

The DIGOUT instruction changes the state of digital output bits 15-8. The output changes occur at the next tick of the output sequencer clock, that is, they are precisely timed (unlike DIGLOW).

```
DIGOUT  [pattern]|Vn|[Vn±off],OptLab
```

pattern This determines the new output state. You can set, reset or invert each output bit, or leave a bit in the previous state. The pattern is 8 characters long, one for each bit, with bit 15 at the left and bit 8 at the right. The characters can be "0", "1", "i" or "." standing for *clear*, *set*, *invert* or *leave alone*. You may omit the square brackets, however the Format command will insert them.

```
DIGOUT  [....001i]  ;clear bits 3 and 2, set 1, invert 0
DIGOUT  [.......i]  ;invert 0 again to produce a pulse
DIGOUT  V10         ;use variable V10 to set the pattern
```

Vn With a variable the new output is: (old output BAND Vn(7-0)) BXOR Vn(15-8). The variable equivalent of [....001i] is 241+256*3, and of [.......i] is 255+256*1. If you use a table value, set the same value in the table that you would use for a variable. You can use the VarValue script in the Scripts folder to calculate variable or table values.

OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example produces ten 1 millisecond pulses 100 milliseconds apart.

```
        SET     1           ;run at 1 ms per step
        MOVI    V1,10       ;V1 holds the number of pulses
LOOP:   DIGOUT  [.......1]  ;bit 0 high         >Pulsing
        DIGOUT  [.......0]  ;bit 0 low          >Pulsing
        DELAY   ms(100)-4   ;4 inst in the loop >Pulsing
        DBNZ    V1,LOOP     ;count down         >Pulsing
        HALT                ;finished           >Done
```

DIGLOW

DIGLOW changes the state of the 1401 digital output bits 7-0. Unlike DIGOUT, the output changes occur immediately, they do not wait for the next sequencer clock tick. You can take advantage of this to change all 16 digital outputs almost simultaneously (within a few microseconds) by using DIGOUT followed by DIGLOW.

```
DIGLOW  [pattern]|Vn|[Vn+off],OptLB
```

pattern This determines the new output state. The pattern is 8 characters long, one for each bit, with bit 7 at the left and bit 0 at the right. The characters can be "0", "1", "i" or "." standing for *clear*, *set*, *invert* or *leave alone*. You may omit the square brackets, however Format will insert them.

```
DIGLOW  [....001i]  ;clear bits 3 and 2, set 1, invert 0
DIGLOW  [.......i]  ;invert 0 again to produce a pulse
DIGLOW  V10         ;use variable V10 to set the pattern
```

Vn With a variable the new output is: (old output BAND Vn(7-0)) BXOR Vn(15-8). The variable equivalent of [....001i] is 241+256*3, and of [.......i] is 255+256*1. If you use a table value, set the same value in the table that you would use for a variable. You can use the VarValue script in the Scripts folder to calculate variable or table values.

OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

DIBEQ DIBNE

These instructions test digital input bits 7-0 against a pattern. DIBEQ branches on a match. DIBNE branches on no match. Both instructions copy digital input bits 7-0 to V56 (VDigIn), for use by DISBEQ and DISBNE.

```
DIBNE   [pattern]|Vn|[Vn+off],LB
DIBEQ   [pattern]|Vn|[Vn+off],LB
```

pattern This is 8 characters, one for each input bit. The characters can be "0", "1" and "." meaning match 0 (TTL low), match 1 (TTL high) or match anything. The bit order in the pattern is [76543210]. You may omit the square brackets, however the Format command inserts them.

Vn With a variable (input BAND Vn(7-0)) BXOR Vn(15-8) is 0 for a match, else is not a match.

LB The branch destination label if the input is a match (DIBEQ) or is not a match (DIBNE). From [11.00] you can use +n or -n (n is the number of steps) to branch to n steps after or before the current step.

This example waits for a pulse sequence in which the falling edges of two consecutive pulses are less than 2*V1+2 sequencer clock ticks apart. It waits for a falling edge, waits for a rising edge with a time out and then waits for the next falling edge with a time out. If timed out, we start again. If the input signal has high states less than three ticks wide, or low states less than 2 ticks wide, this example may miss them.

```
WHI:    DIBNE   [.......1],WHI  ;wait until high       >Wait high
SETTO:  MOVI    V1,24           ;set 50 step timeout   >Wait low
WLO:    DIBNE   [.......0],+0   ;wait for falling      >Wait low
TOHI:   DIBEQ   [.......1],TOLO ;wait for high         >Wait high
        DBNZ    V1,TOHI         ;loop if not timed out >Wait high
        JUMP    WHI             ;timed out, restart    >Restart
TOLO:   DIBEQ   [.......0],GOTIT;jump if found events  >Wait low
        DBNZ    V1,TOLO         ;loop if not timed out >Wait low
        JUMP    SETTO           ;timed out, restart    >Restart
GOTIT:  ...                     ;here for 2 close pulses
```

DISBEQ DISBNE

These instructions test digital input bits 7-0 read by the last DIBEQ, DIBNE or WAIT against a pattern. DISBEQ branches on a match. DISBNE branches if it does not match.

```
DISBNE  [pattern]|Vn|[Vn+off],LB
DISBEQ  [pattern]|Vn|[Vn+off],LB
```

pattern This is 8 characters, one for each input bit. The characters can be "0", "1" and "." meaning match 0 (TTL low), match 1 (TTL high) or match anything. The bit order in the pattern is [76543210]. You may omit the square brackets, however the Format command inserts them.

Vn With a variable (input BAND Vn(7-0)) BXOR Vn(15-8) is 0 for a match, else is not a match.

LB The branch destination label if the input is a match (DISBEQ) or not a match (DISBNE). From [11.00] you can use +n or -n (n is the number of steps) to branch to n steps after or before the current step.

This example shows a typical use of this instruction. We run trials signalled by external equipment that writes the trial type to digital input bits 1 and 0; 00 means no trial, 01, 10 and 11 select trial types 1, 2 and 3.

```
TRWAIT:'W DIBEQ   [......00],TRWAIT ;Wait for trial >Wait...
          DISBEQ  [......01],TRIAL1
          DISBEQ  [......10],TRIAL2
          DISBEQ  [......11],TRIAL3
```

WAIT

The WAIT instruction causes the sequence to wait until bits 7-0 of the 1401 digital input match a pattern. The digital input port is sampled once every sequencer clock tick until the pattern is found, or until the sequence is sent elsewhere by a keyboard command. WAIT copies digital input bits 7-0 to V56 (VDigIn) for use by DISBEQ and DISBNE. It is usually a good idea to have a display message explaining what you are waiting for.

```
        WAIT    [pattern]|Vn|[Vn+off]
```

pattern This is the input condition to match before the sequence can continue. It is 8 characters long, one for each input bit. The characters can be "1", "0" or "." indicating that the input bit in that position must be a one, a zero or don't care. The bits are given in the order [76543210]. You may omit the square brackets round the pattern if you wish; the Format command will insert them.

```
        WAIT    [.......1]  ;wait for bit 0 set>Wait for bit 0
        WAIT    [0......0]  ;wait for 7 and 0 clear>Wait 7&0 low
```

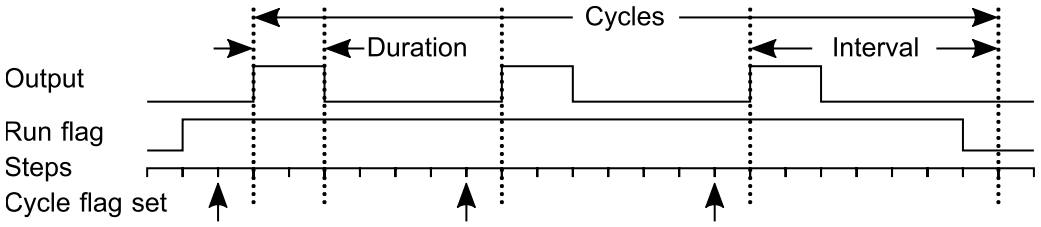
Vn With a variable, (input BAND Vn(7-0)) BXOR Vn(15-8) must be 0 to continue.

This instruction is shorthand for:

```
HERE:   DIBNE   [pattern]|Vn|[Vn+off],HERE
```

DIGPS

This command together with DIGPC (to start and stop output and clear the pulse cycle flag) and DIGPBR (to test for start of pulses and output done) were added at [10.06] to generate trains of digital output pulses that run independently of other sequencer operations. Pulses are on digital output bits 15-8 (9-8 for Micro2 and Micro3); these are the same bits that are controlled by the DIGOUT instruction. The output is defined in pulse cycles. The digital output sets at the start of each pulse cycle and clears after a duration. The cycle length and pulse duration are set in sequencer steps. By default, set means high output and clear means low, but you can choose to invert the output state. Output changes are calculated at the start of each sequencer step and occur on the sequencer clock tick after the step that changes them. If you HALT the sequencer, the output stops.



DIGPS prepares the digital pulses for running and adjusts the state of the pulse output. Pulses are defined by four parameters: the interval between repeats of the pulses, the duration of each pulse, the direction of the pulses and the number of pulses.

```
        DIGPS  n,op,value[Vn|[Vn+off],OptLB
```

n The output channel to use in the range 0-7 corresponding to the digital output bits 8-15. 0 and 1 will appear on the 1401 front panel unless routed to the rear panel. The Micro2 and 3 support only channels 0 and 1.

op From 1-2 option characters. One of these must be S, P, I, D or C. There is an optional flag: R intended to qualify the S and P options but can be used with the other options.

value The number of sequencer steps to set for the S, P, I or D options or the number of cycles to run for in the C option. You can also use a variable or a table entry to provide this parameter.

OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

Options characters

You must include one of the following options:

op	value	Purpose

S	Interval	Set up a square wave. value sets the repeat interval in sequencer steps. The pulse duration is set to value/2 steps. This resets the command so that a pulse starts immediately when told to go (or on the next step if already running). This does not change the cycle count (use the R flag to reset it so that the pulse train runs for 2^32 cycles).
P	Interval	Sets up a pulsed output. value sets the repeat interval in sequencer steps. The pulse duration is set to 1 step. This resets the command so that a pulse starts immediately when told to go (or on the next step if already running). This does not change the cycle count (use the R flag to reset it so that the pulse train runs for 2^32 cycles).
I	Interval	Changes the interval between pulses and has no other effect (does not change the duration). If used while the output is running it has no effect until the end of the current pulse cycle. If you set the interval to the same or less than the duration the output will not clear. If you intend to change the interval dynamically you should probably set a pulse duration of 1 (P option).
D	Duration	Changes the width of each pulse and has no other effect. Setting the width greater than or equal to the interval will not work. Changing the width while running is not recommended as it can cause an extra long pulse.
C	Cycle count	Change the number of cycles to be done. Setting a cycle count of -1 sets 2^32 cycles, which will run for a long time (this is what the R flag does). Setting the count to n while running requests n more cycles after the current cycle ends. Setting the count to 0 while running stops output at the end of the current cycle.

The flag character is:

R	Reset the cycle count to the maximum value (more than 4 billion cycles, effectively run forever).

Use of the commands

We expect you to use one the S or P option first to prepare to output a pulse train. This is likely followed by the C option to set the number of cycles unless you want to run until stopped. You start the train with the DIGPC n,G command. You can test the state of the output with the DIGPBR command. To run continuously until stopped you might use:

```
       DIGPS  0,SR,s(1)    ;Square wave, 1 Hz, run continuously
       DIGPC  0,G          ;Start (output changes 2 steps later)
       ...
       DIGPS  0,C,0        ;Request stop at end of current cycle
WEND:  DIGPBR 0,S,WEND     ;Wait for stopped > Wait for stop
```

To run a fixed number of cycles you could use:

```
       DIGPS   0,P,ms(50)  ;1 step pulse every 50 milliseconds
       DIGPS   0,C,4       ;Run 4 cycles
       DIGPC   0,G         ;Start
```

Performance considerations

Each channel of output consumes 1401 time at the start of each sequencer step (in the same way as sinusoidal output and RAMP output). The overhead per channel per step is similar to the RAMP instruction and less than the sinusoidal output.

DIGPC

The DIGPC command (DIGital Pulse Control) enables and disables the pulse train output and clears the pulse cycle start flag. This command together with DIGPS (to set pulse train parameters) and DIGPBR (to test for start of cycles and output done) were added at [10.06] to generate trains of digital i/o pulses.

```
       DIGPC  n,op,value,OptLB
```

n	The output channel to use in the range 0-7 corresponding to the digital output bits 8-15. 0 and 1 will appear on the 1401 front panel unless routed to the rear panel. The Micro2 and 3 support only channels 0 and 1.

`op` From 1-2 case insensitive option characters that must include one of: `G` to start the pulse train, `S` to stop the pulse train or `C` to clear the pulse cycle flag. There is an optional flag character: `I` to invert the output, used with the `G` option.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

G Enable train
The `G` option enables the pulse train for the output channel (if not already enabled). When used after the `DIGPS S` or `P` options, the next sequencer step will set the output, but the effect of this is synchronized to the following sequencer clock tick, so the output changes 2 sequencer steps after the `G` option is used. When used after the `DIGPC S` option the sequencer resumes from wherever it had just reached.

S Suspend train
The S option suspends the pulse train. You can resume it from the same state with the `G` option.

C Clear pulse cycle flag
The cycle flag is set at the start of each output cycle, one sequencer step before the output changes. The `C` option clears the cycle flag (it is also cleared by testing it with the `DIGPBR n,C` command).

I Invert output
Normally, the designated output goes high at the start of each cycle and low after the pulse duration. If you include this flag with the `G` option, the output goes low at the start of each cycle and high after the duration. The flag is ignored with the `S` and `C` options.

DIGPBR

The `DIGPBR` command (`DIG`ital `P`ulse `BR`anch) tests if the pulse train is running and the state of the cycle flag. This command together with `DIGPS` (to set pulse train parameters) and `DIGPC` (to control the pulse train) were added at [10.06] to generate trains of digital i/o pulses.

```
        DIGPBR  n,what,Label
```

`n` The output channel to use in the range 0-7 corresponding to the digital output bits 8-15. 0 and 1 will appear on the 1401 front panel unless routed to the rear panel. The Micro2 and 3 support only channels 0 and 1.

`what` A single character, one of `C` to test the cycle flag or `S` to test for stopped.

`Label` Where to branch to if the condition tested is not true.

C flag
If you test the new cycle flag, the branch is taken if the pulse train is running and the cycle flag is not set. The sequence continues to the next instruction if the pulse train is not active or if the new cycle flag is set. This instruction also clears the new cycle flag. Note that the cycle flag sets one sequencer step before the output changes.

S flag
If you test the S (Suspended) flag, the branch is taken if the pulse train is running and runs the next instruction if the pulse train is not running.

Example
The following sequence runs 4 cycles of pulses at 1 second intervals, waiting for the first 3 pulses, then waiting for the sequence to end:

```
          SET    1,1,0
          DIGOUT [.......0]      ;Set initial state
       'G DELAY  s(2)            ;Delay 2 seconds
          DIGPS  0,P,s(1)        ;Pulse every second
```

```
            DIGPS  0,C,4          ;Run 4 cycles
            DIGPC  0,G            ;Go
W0:         DIGPBR 0,C,W0         ;C flag will be set so no wait
W1:         DIGPBR 0,C,W1         ;Wait 1 second here
W2:         DIGPBR 0,C,W2         ;Wait 1 second here
WE:         DIGPBR 0,S,WE         ;Wait 2 seconds here
```

DIGIN BZERO BNZERO

These obsolete instructions are provided for backwards compatibility. DIBNE and DIBEQ are more efficient. DIGIN reads digital input bits 7-0, compares them with a pattern and saves the result. The result can be tested with the branching instructions BZERO and BNZERO. The comparison result is saved until the next DIGIN.

```
        DIGIN  [pattern]|Vn|[Vn+off]     ;Test input state
        BZERO  LB               ;branch to LB if result is zero
        BNZERO LB               ;branch to LB if result non-zero
```

pattern This is 8 characters, one for each input bit in the order [76543210]. The characters can be ".", "0", "1" and "c". For "0" or "1", the result for that bit is 0 if the input bit was the same, or 1 if it was different. "c" means copy the input bit to the result (this is the same as "0"). The result for "." is always zero. You may omit the square brackets round the pattern, however the Format command will insert them.

Vn With a variable the result is (input BAND Vn(7-0)) BXOR Vn(15-8). The variable equivalent of [0.c0110.] is 190 + 256*12. The VarValue script calculates values equivalent to patterns.

LB The branch destination if the last DIGIN produced a zero (BZERO) or non-zero (BNZERO) result. From [11.00] you can use +n or -n (n is the number of steps) to branch to n steps after or before the current step.

```
Loop:   DIGIN  [0.c0110.]       ;assume input is 01101011
        BZERO  GoOn             ;result is 00100110 so no branch
        BNZERO Loop             ;This will branch
GoOn:   ...
```

DAC outputs

The output sequencer supports up to 8 DAC (Digital to Analogue Converter) outputs. The Power1401 has four DACs and the Micro1401 has two. However, the Power1401 can be expanded with up to 8 DAC outputs and the Micro4 with up to 4. The last value written to DACs 0-7 is stored in variables V57-V64 (you can also refer to these variables as VDAC0-VDAC7). The values are stored as 32-bit numbers with the full 32-bit range corresponding to the full range of the DAC. This high resolution allows us to ramp the DACs smoothly. If you write to a DAC that does not exist in your 1401, the variable associated with the DAC is set as if the DAC were present.

Constant values written to the DACs are expressed in units of your choice. The SET or SDAC directive determines the conversion between the numbers you supply and the DAC outputs. The standard settings for a system with ±5 Volts DACs is to set the DAC outputs in Volts. If you use sequencer variables to update the DACs, you must scale the user units into suitable values; see the DAC and ADDAC instructions for details.

The Power1401 DAC 2 and 3 outputs are on the rear panel 37-way Cannon D type Analogue Expansion connector. DAC 2 is pin 36 and DAC 3 is pin 37. Suitable grounds are on adjacent pins 18 and 19. If you have a Power1401 top-box with additional front panel DACs, the two rear DACs are mapped to the two highest numbered DACs. For example, with a 2709 Spike2 Top Box, DACs 2 and 3 are available as front panel BNC connections, and the rear panel DACs become DAC 4 on pin 36 and DAC 5 on pin 37.

DAC	Set DAC value (for DACs 0-7)
(DACn)	Version 3 compatible, set DAC n (0-3) to a value
ADDAC	Increment DAC by a value
ADDACn	Version 3 compatible, increment DAC n (0-3) by a value
RAMP	Set automatic DAC ramping to a target value

DAC ADDAC

The DAC instruction can write a value to any of the 8 possible DAC outputs. The ADDAC instruction adds a value to the DAC output. The output value changes immediately unless the DAC is in use by the arbitrary waveform output, in which case the result is undefined.

```
        DAC     n,expr|Vn|[Vn+off],OptLB
        ADDAC   n,expr|Vn|[Vn+off],OptLb
```

n — The DAC number, in the range 0-7. Variable 57+n is set to the new DAC value such that the full DAC range spans the full range of the 32-bit variable.

expr — The DAC value to write or to add. The value units depend on the SET or SDAC directive; the standard units are Volts. It is an error to give a value that exceeds the DAC output range.

Vn — When a variable is used, the full range of the 32-bit variable corresponds to the full range of the DAC. You can use the VDAC32() function to load a variable with a constant value using user-defined DAC units. See the script example, below, to calculate non-constant values.

OptLB — If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example sets DAC 2 to 0 Volts, then ramps it to 4.99 Volts in 1 second using steps of 0.01 Volts. The example also shows how to do the same ramp using variables. You can also use RAMP to ramp a DAC.

```
        SET   1,1,0           ;1 ms per step, DAC scaled to Volts
      'R DAC   2,0            ;Ramp 0 to 5      >Ramping
        MOVI  V1,499          ;499 steps        >Ramping
RAMP1:  ADDAC 2,0.01          ;0.01V increment  >Ramping
        DBNZ  V1,RAMP1        ;count increments >Ramping
        HALT                  ;task finished    >Done
      'V MOVI  V3,VDAC32(0)   ;Use variables    >Ramping
        MOVI  V2,VDAC32(0.01) ;increment in V2  >Ramping
        MOVI  V1,499          ;499 steps        >Ramping
        DAC   2,V3            ;set initial value>Ramping
RAMP2:  ADDAC 2,V2            ;add increment    >Ramping
        DBNZ  V1,RAMP2        ;count increments >Ramping
        HALT                  ;task finished    >Done
```

It is a property of the signed integer numbers we use that if you add 1 to the maximum possible positive number, the result is the minimum possible negative number. If you use ADDAC repeatedly to add the same value, eventually you will run off the end of the DAC range and come back in at the other end.

Physical DAC units run from -32768 to +32767. In a ±5 Volt system with 16-bit DACs, this is -5.0000 to +4.99985 Volts. The DAC unit value for +5 Volts is +32768, but this number does not exist in 16-bit signed integers and wraps around to -32878. Because it often happens that users want to set the DAC to full scale, for the DAC command used with expr (not with Vn), we change requests to set +32768 units to set -32767 units. When used with a variable (Vn), the top 16 bits of the 32-bit variable are physical DAC units, the lower 16 bits provide extra accuracy when ramping the DACs.

This script shows how to convert a user value into a value that can be assigned to a sequencer variable for use in the DAC and ADDAC sequencer commands.

```
const DACScale := 1.0;       'Values for user DAC units in Volts
const DACOffset := 0.0;      'on a 5V 1401

'Convert a value in user units to a 32-bit DAC value
Func Dac32%(user)
const dMin := -32768*65536, dMax := 32767*65536;
var d32 := (user - DACOffset)*6553.6*65536/DACScale;    'DAC value as 32-bits
if _Version >= 1017 then
    d32 := Clamp(d32, dMin, dMax);    ' If using Spike2 10.17 or later
else
    d32 := (d32 > dMax) ? dMax : ((d32 < dMin) ? dMin : d32); ' older
endif
return round(d32);
end;
```

Unlike the digital outputs, the DAC output changes when the instruction runs, not at the next sequencer clock tick. This means that the changes may have a time jitter of a few microseconds.

DACn ADDACn

These commands provide backwards compatibility with old Spike2 versions and should be avoided in new sequences. The DACn and ADDACn commands (with n = 0, 1, 2 or 3) set and change the 1401 DAC outputs. expr is the new DAC output level, or change in level. Variable 57+n is set to the new DAC value such that the full DAC range spans the full range of the 32-bit variable. These commands do not support table use.

```
DACn     expr|Vn,OptLB
ADDACn   expr|Vn,OptLB
```

expr — The value to assign to DAC n (DACn instruction) or to add to the output level (ADDACn instruction). The units of the DAC values are usually Volts, but can be changed by setting a scale factor with SET or SDAC.

Vn — Variable values -32768 to 32767 correspond to the full DAC range. You can use VDAC16() to load a variable using user-defined DAC units. See the script example, below, to calculate non-constant values or the VarValue script. The value saved in V57+n is Vn*65536.

OptLB — If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

The important difference between these commands and DAC and ADDAC is where a variable is used. The bottom 16-bits of the variable are written to the DAC. In the case of the DAC and ADDAC commands, the upper 16 bits of the variable are written to the DAC. This command is sometimes useful with the CHAN command when reading a waveform or a DAC channel, as it reads values in the range -32868 to 32767.

This script converts a value in user units to a value that you can assign to a script variable for use in the DACn and ADDACn sequencer commands.

```
const DACScale := 1.0;      'Values for user DAC units in Volts
const DACOffset := 0.0;     'on a 5V 1401

'Convert a value in user units to a 16-bit DAC value
Func Dac16%(user)
var d16 := (user - DACOffset)*6553.6/DACScale;    'DAC value as 16-bits
if _Version >= 1017 then
    d16 := Clamp(d16, -32768, 32767);   ' If using Spike2 10.17 or later
else
    d16 := (d16 > 32767) ? 32767 : ((d16<-32768) ? -32768 : d16); ' older
endif
return round(d16);
end;
```

RAMP

This command starts a DAC ramping, with automatic updates on every sequencer step. If the DAC was generating cosine output, the cosine output stops. The DAC ramps from the current value until it reaches a target value, when the DAC cycle flag sets. You can use WAITC to test for the end of the ramp. The RATE instruction stops a ramp before it reaches the target value.

```
RAMP     n,target|Vn,slope|Vs|[Vs+off]
```

n — The DAC number in the range 0-7 for the Power1401 or 0-1 for the Micro1401.

target — This is the DAC value at which to end the ramp. The units of the DAC values are usually Volts, but can be changed by setting a scale factor with SET or SDAC.

Vn — When a variable is used for the target, the full range of the 32-bit variable corresponds to the full range of the DAC. You can use the VDAC32() function to load a variable using user-defined DAC units.

slope — This expression sets the DAC increment per sequencer step. The sign of the value you set here is ignored as the sequencer works out if it must ramp upwards or downwards to achieve the desired

target value. If your DAC is calibrated in Volts, to achieve a slope of 1 Volt per second, use `1.0/s(1.0)` for the slope.

`Vs` You can also set the slope from a variable or by reading it from the table. In this case, the full range of the 32-bit value represents the full range of the DAC. The absolute value of the 32-bit value is used to change the DAC on each step. To get a slope of 1 user unit per second, us `VDAC32(1.0)/s(1.0)` as the value.

This example ramps the DAC 1 from its current level to 1 Volt over 3 seconds, waits 1 second, then ramps it down to 0 Volts over 5 seconds.

```
        RAMP    1,1.0,1.0/S(3) ;start with zero size
        ...                    ;other instructions during ramp
WT1:    WAITC   1,WT1          ;wait for ramp to end >Ramp to 1
        DELAY   S(1)-1         ;wait for a second    >Wait 1 sec
        RAMP    1,0,1.0/S(5)   ;ramp down
WT2:    WAITC   1,WT2          ;wait for ramp        >Ramp to 0
```

The `OFFSET` command has an example that uses `RAMP` with cosine waves.

Arbitrary waveform output

You cannot use this command to ramp the output after arbitrary waveform output as it does not save the last value output to each DAC for performance reasons.

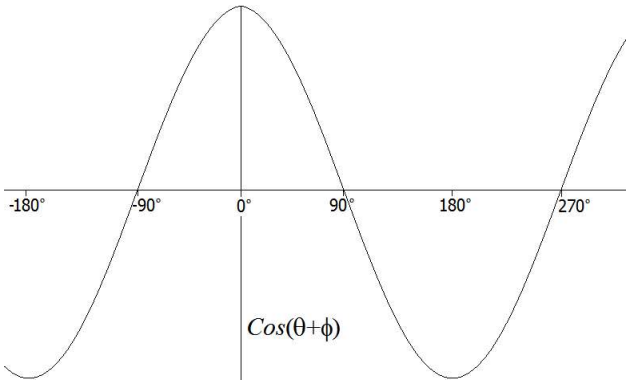
Cosine output control instructions

The sequencer can output cosine waveforms of variable amplitude and frequency through Power1401 DACs 0 to 7 and Micro1401 DACs 0 and 1 (and 2 and 3, if fitted in a Micro4). Attempts to use unimplemented Mirco1401 DACs 2 and 3 are treated as `NOP`; attempts to use Micro1401 DACs 4 to 7 are flagged as errors. When enabled, the cosine value is computed and output every step; if you `HALT` the sequencer, the cosine output will stop. The output (in a 5 Volt 1401) is:

output in Volts = $5\ A\ Cos(\theta+\phi)$ + offset

where: A is an amplitude scaling factor in the range 0 to 1
θ an angle in the range 0° to 360° that changes each step (set by `ANGLE`)
ϕ is a fixed phase in the range -360° to 360° (set by `PHASE`)
offset a voltage offset set by the `OFFSET` instruction

θ changes every step by $\delta\theta$. A cycle of the cosine takes $360/\delta\theta$ steps. You can change the angle increment immediately, or you can delay the change until the next time θ passes through 0°. You can set $\delta\theta$ in the range 0° up to 360° to an accuracy of about 0.0000001°. With the sequencer running at 1 kHz, you can output frequencies up to 500 Hz with a frequency resolution of around 0.00012 Hz. Ideally the output would be passed through a low pass filter with a corner frequency at one half or less of the sequencer step rate to smooth out the steps in the cosine wave.



By adjusting ϕ you control the output cosine phase where θ passes through zero. Unless you set the value (`PHASE`), it is zero and the zero crossing occurs at the peak of the sinusoid. To have the output rising through 0, set the phase to –90.

Each time θ passes through zero a *new cycle* flag sets. The `RAMP`, `RATEW`, `RINCW`, `WAITC` and `CLRC` instructions clear the flag.

`SZ (CSZ,DSZ)`	Set the cosine output amplitude
`SZINC (CSZINC,DSZINC)`	Change the cosine output amplitude
`RATE(CRATE,DRATE)`	Set the cosine angular increment per step
`RATEW(CRATEW,DRATEW)`	As `RATE`, but waits for phase 0
`ANGLE (CANGLE,DANGLE)`	Set cosine angle for the next step
`WAITC(CWAIT,DWAIT)`	Branch until cosine phase 0

`RINC (CRINC,DRINC)`	Change the cosine angle increment per step
`RINCW (CRINCW,DRINCW)`	As `RINC` but waits for phase 0
`PHASE (CPHASE,DPHASE)`	Defines what phase 0 means
`OFFSET (COFF,DOFF)`	Offset for sinusoidal output
`CLRC (CWCLR,DWCLR)`	Clear the new cycle flag

Obsolete commands

Before Spike2 version 5.06, the cosine output instructions supported 2 DACs through the `Cxxxx` and `Dxxxx` instructions. The `Cxxxx` family used DAC 1 on the Power1401 and Micro1401 and DAC 3 on the 1401*plus*. The `Dxxxx` family used DAC 0 on the Power1401 and Micro1401 and DAC 2 on the 1401*plus*. The descriptions include the old forms of the commands, marked `Obsolete`. The DAC numbers next to obsolete commands show the 1401*plus* DAC number in brackets. There are no plans to remove the old commands, but new sequences should avoid them.

SZ

This instruction sets the waveform amplitude. If a wave is playing, the amplitude changes at the next sequencer step. The amplitude is set to 1.0 when sampling starts.

```
        SZ      n,expr|Vn|[Vn+off],OptLB    ;DAC n
        CSZ     expr|Vn|[Vn+off],OptLB      ;DAC 1(3) - Obsolete
        DSZ     expr|Vn|[Vn+off],OptLB      ;DAC 0(2) - Obsolete
```

`n` The DAC number in the range 0-7 (available DACs depend on the 1401 type).

`expr` The cosine amplitude in the range 0 to 1. A cosine with amplitude 1.0 uses the full DAC range.

`Vn` Variable values 0 to 32768 correspond to amplitudes of 0.0 to 1.0; values outside the range 0 to 32768 cause undefined results. **If you are getting strange results, check the range of variable values you are setting.**

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

SZINC

These instructions change the waveform amplitude. The change is added to the current amplitude. If the result exceeds 1.0, it is set to 1.0. If it is less than 0, the result is 0.

```
        SZINC  n,expr|Vn|[Vn+off],OptLB     ;DAC n
        CSZINC expr|Vn|[Vn+off],OptLB       ;DAC 1(3) - Obsolete
        DSZINC expr|Vn|[Vn+off],OptLB       ;DAC 0(2) - Obsolete
```

`n` The DAC number in the range 0-7 (available DACs depend on the 1401 type).

`expr` The change in the waveform scale in the range -1 to 1.

`Vn` A variable value of 32768 is a scale change of 1.0, -16384 is -0.5 and so on.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

You can gradually increase or decrease the wave amplitude. For example, the following increases the amplitude from zero to full scale (we assume that the waveform is playing):

```
        SZ      0,0.0     ;start with zero size
        MOVI    V1,100    ;proceed in 1% increments
loop:   SZINC   0,0.01    ;a 1% increase
        DELAY   ms(100)-2 ;show some of the waveform at this size
        DBNZ    V1,loop   ;loop 100 times
```

RATE

This sets the angle increment in degrees per step, which sets the cosine frequency. If the nominated DAC was ramping, this cancels the ramp. You can stop the cosine output with a rate of 0. Any non-zero value starts the cosine output.

```
        RATE    n,expr|Vn|[Vn+off],OptLB    ;DAC n
        CRATE   expr|Vn|[Vn+off],OptLB      ;DAC 1(3) - Obsolete
        DRATE   expr|Vn|[Vn+off],OptLB      ;DAC 0(2) - Obsolete
```

`n` The DAC number in the range 0-7 (available DACs depend on the 1401 type).

`expr` The angle increment per step in the range 0.000 up to 180 degrees. The `Hz()` function calculates the increment required for a frequency.

`Vn` For a variable, the value 11930465 is an increment of 1 degree. The `VHz()` function can be used to set a variable value equivalent to an angle in degrees.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example starts cosine output at 10 Hz, runs for 10 seconds, and then stops it. This is then repeated using a variable to produce the same effect:

```
        SET 1,1,0          ;1 ms per step
     'C RATE    0,Hz(10)   ;start output at 10 Hz
        DELAY   S(10)-1    ;delay for 10 seconds >Sine wave
X:   'S RATE    0,0        ;stop output
        HALT                                  >Stopped
     'V MOVI    V1,VHz(10) ;set V1 equivalent of 10 Hz
        RATE    0,V1       ;start at 10 Hz
        DELAY   S(10)-1,X  ;delay then goto exit >Sine wave
```

RATEW

This instruction performs the same function as `RATE`, except that the change is postponed until the next time Theta passes through 0 degrees. `RATEW` cannot start output; a sinusoid must already be running to pass phase 0. It can stop output, but does not remove the overhead for using cosine output. This instruction clears the new cycle flag (see `WAITC`).

```
        RATEW   n,expr|Vn|[Vn+off]          ;DAC n
        CRATEW  expr|Vn|[Vn+off]            ;DAC 1(3) - Obsolete
        DRATEW  expr|Vn|[Vn+off]            ;DAC 0(2) - Obsolete
```

`n` The DAC number in the range 0-7 (available DACs depend on the 1401 type).

`expr` The angle increment in the range 0.000 to 180 degrees. The `Hz()` built-in function calculates the increment required for a frequency.

`Vn` For a variable, the value 11930465 is an increment of 1 degree. The `VHz()` function can be used to set a variable value equivalent to an angle in degrees.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example starts cosine output at 10 Hz, runs for 1 cycle, changes to 11 Hz for one cycle, then stops:

```
        SET 1,1,0          ;1 ms per step
        ANGLE   0,0        ;make sure we are at phase 0
        RATE    0,Hz(10)   ;start output at 10 Hz
        RATEW   0,Hz(11)   ;request 11 Hz next time around
CYCLE10: WAITC  0,CYCLE10  ;wait for the cycle>10Hz
CYCLE11: WAITC  0,CYCLE11  ;wait for the cycle>11Hz
        RATE    0,0        ;stop output
```

ANGLE

This changes the cosine angular position. It takes effect on the next instruction when the angle increment is added to the value set by this instruction and the result is output.

```
        ANGLE   n,expr|Vn|[Vn+off],OptLB    ;DAC n
        CANGLE  expr|Vn|[Vn+off],OptLB      ;DAC 1(3) - Obsolete
        DANGLE  expr|Vn|[Vn+off],OptLB      ;DAC 0(2) - Obsolete
```

`n` The DAC number in the range 0-7 (available DACs depend on the 1401 type).

expr The phase angle to set in the range -360 up to +360.

Vn For a variable, the value 11930465 is a phase of 1 degree (to be precise, 4294967296/360 is a phase of 1 degree). The VAngle() function converts degrees into a suitable value for a variable.

OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example sets the phase angle to –90 degrees directly, and by using a variable. There is no need to use the VAngle() function; we could have set V1 to –1073741824. However, VAngle(-90) is much easier to understand.

```
        ANGLE   1,-90       ;set the DAC 1 cosine angle directly
        MOVI    V1,VAngle(-90)
        ANGLE   1,V1        ;set using a variable
```

PHASE

This changes the relative phase of the cosine output for the next cosine output. A common use is to change the output from a cosine (maximum value at phase zero) to sine (rising through zero at phase zero).

```
        PHASE   n,expr|Vn|[Vn+off],OptLB    ;DAC n
        CPHASE  expr|Vn|[Vn+off],OptLB      ;DAC 1(3) - Obsolete
        DPHASE  expr|Vn|[Vn+off],OptLB      ;DAC 0(2) - Obsolete
```

n The DAC number in the range 0-7 (available DACs depend on the 1401 type).

expr The relative phase angle to set in the range -360 up to +360. The relative phase is set to 0 when sampling starts. Set –90 for sinusoidal output.

Vn For a variable, the value 11930465 is a phase of 1 degree (to be precise, 4294967296/360 is a phase of 1 degree). The VAngle() function converts degrees into a suitable value for a variable.

OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example plays a 1 Hz sinusoidal output (assuming that the output is not running).

```
        PHASE   2,-90     ;set the DAC 2 phase angle directly
        ANGLE   2,0       ;prepare to start as a sine wave
        RATE    2,Hz(1)   ;start the sinusoid
```

OFFSET

This changes the cosine output voltage offset for the next cosine output.

```
        OFFSET  n,expr|Vn|[Vn+off],OptLB    ;DAC n
        COFF    expr|Vn|[Vn+off],OptLB      ;DAC 1(3) - Obsolete
        DOFF    expr|Vn|[Vn+off],OptLB      ;DAC 0(2) - Obsolete
```

n The DAC number in the range 0-7 (available DACs depend on the 1401 type).

expr The offset value for sinusoidal output. The units of this value depend on the SET or SDAC directives; the standard units are Volts. It is an error to give a value that exceeds the DAC output range.

Vn When a variable is used, the full range of the 32-bit variable corresponds to the full range of the DAC. You can use the VDAC32() function to load a variable using user-defined DAC units.

OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example ramps DAC 0 from 0 to 1 Volt, the runs 5 cycles of a sine wave at 1 Hz, and finally ramps the data back to 0 Volts.

```
            SET    1 1 0          ;1 millisecond per step
            DAC    0,0            ;use DAC 0 for all output
            OFFSET 0,1.0          ;set DAC 0 offset
            SZ     0,0.2          ;1 V sinusoid
            PHASE  0,-90          ;Prepare sinusoid
            ANGLE  0,0            ;set start point
            RAMP   0,1.0,1.0/s(1) ;ramp to 1 Volt in 1 sec
RAMPUP:     WAITC  0,RAMPUP       ;wait for ramp      >Ramp up
```

```
            RATE   0,HZ(1)         ;start sinusoid
            DELAY  S(4.9)          ;Sinusoid           >Sine
            RATEW  0,0             ;stop at cycle end
END:        WAITC  0,END           ;wait for end       >Wait end
            RATE   0,0             ;stop now
            RAMP   0,0.0,1.0/S(1)  ;ramp to 0 Volt in 1 sec
RAMPDN:     WAITC  0,RAMPDN        ;wait               >Ramp down
            HALT
```

WAITC

Each time the phase angle of a cosine passes through 0°, or a ramp terminates, a new cycle flag sets. There is a separate flag for each DAC. This flag is cleared by `CLRC`, `RATEW`, `RINCW` and when tested by `WAITC`.

```
        WAITC   n,LB                        ;DAC n
        CWAIT   LB                          ;DAC 1(3) - Obsolete
        DWAIT   LB                          ;DAC 0(2) - Obsolete
```

`n` The DAC number in the range 0-7 (available DACs depend on the 1401 type).

`LB` A label to branch to if the new cycle flag is clear, otherwise the sequencer clears it and continues.

This instruction can produce a pulse one step after the start of each waveform cycle. The following sequence outputs 4 cycles of waveform at different rates on DAC 1, and changes the digital outputs for each cycle.

```
        SZ      1,1.0      ;make sure full size
        ANGLE   1,0.0      ;make sure we start at phase 0
        RATE    1,1.0      ;1 degree per step to start with
        DIGOUT  [00000001] ;so outside world knows
        RATEW   1,1.2      ;next cycle faster, clear cycle flag
w1:     WAITC   1,w1       ;wait for cycle >1 degree cycle
        DIGOUT  [00000010] ;announce another cycle
        RATEW   1,1.4      ;next cycle a bit faster
w2:     WAITC   1,w2       ;wait for cycle >1.2 degree cycle
        DIGOUT  [00000011] ;yet another one
        RATEW   1,1.6      ;last cycle a bit faster
w3:     WAITC   1,w3       ;wait for cycle >1.4 degree cycle
        DIGOUT  [00000100] ;last cycle number
w4:     WAITC   1,w4       ;wait for end   >1.6 degree cycle
        RATE    1,0.0      ;stop waveform
```

RINC RINCW

These instructions behave like `RATE` and `RATEW` except that they *change* the output rate (angle increment per step) by their argument rather than set it. `RINCW` clears the new cycle flag.

```
        RINC    n,expr|Vn|[Vn+off],OptLB    ;DAC n
        RINCW   n,expr|Vn|[Vn+off],OptLB    ;DAC n
        CRINC   expr|Vn|[Vn+off],OptLB      ;DAC 1(3) - Obsolete
        CRINCW  expr|Vn|[Vn+off],OptLB      ;DAC 1(3) - Obsolete
        DRINC   expr|Vn|[Vn+off],OptLB      ;DAC 0(2) - Obsolete
        DRINCW  expr|Vn|[Vn+off],OptLB      ;DAC 0(2) - Obsolete
```

`n` The DAC number in the range 0-7 (available DACs depend on the 1401 type).

`expr` The change in the angle increment per step. You can use the built-in `Hz()` function to express the change as a frequency.

`Vn` For a variable, the value 11930465 is a change of 1 degree. You can use the `VarValue` script in the `Scripts` folder to calculate variable values.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example starts cosine output at 10 Hz and lets you adjust it from the keyboard.

```
        SET    1,1,0     ;1 ms per step
        RATE   1,Hz(10)  ;start output at 10 Hz
```

```
wt:      JUMP   wt        ;HALT stops all output>P=+1Hz, M=-1Hz
      'P RINC   1,Hz(1),wt;1 Hz faster
      'M RINC   1,Hz(-1),wt;1 Hz slower
```

These instructions can be used to produce waveforms that change gradually in frequency. The following code generates a linear speed increase every two steps on DAC 1:

```
         SZ      1,1.0    ;make sure full size
         ANGLE   1,0.0    ;make sure we start at phase 0
         RATE    1,1.0    ;1 degree per step to start with
         MOVI    V1,900   ;in 900 steps of...
loop:    RINC    1,0.01   ;...1/100 degrees to...
         DBNZ    V1,loop  ;...10 degrees per step
```

The next example produces 90 cycles using V10 as a counter, increasing by 0.1 degrees per step per cycle.

```
         SZ      1,1.0    ;make sure full size
         ANGLE   1,0.0    ;make sure we start at phase 0
         RATE    1,1.0    ;1 degree per step to start with
         MOVI    V10,90   ;in 90 steps of...
loop:    RINCW   1,0.1    ;...1/10 degrees to...
wait:    WAITC   1,wait   ;...(wait for next cycle)...
         DBNZ    V10,loop ;...10 degrees per step
```

CLRC

This instruction clears the cosine output new cycle flag. If you have been running for several cycles and you want to stop the next time phase 0 is crossed use this instruction immediately before using WAITC.

```
         CLRC   n,OptLB                    ;DAC n
         CWCLR  OptLB                      ;DAC 1(3) - Obsolete
         DWCLR  OptLB                      ;DAC 0(2) - Obsolete
```

n The DAC number in the range 0-7 (available DACs depend on the 1401 type).

OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example starts a sinusoid and then stops at the next phase 0 crossing after the user requests a stop. Because the sinusoid passes phase 0 in the WAITC instruction and does another step in the RATE 1,0 instruction, we offset the phase by 2 steps. However, this would cause the start of the sinusoid to be 2 steps wrong, so we change the start angle to match.

```
     'G PHASE   1,-2*Hz(2)    ; compenstate for ending
        ANGLE   1,2*Hz(2)     ; so we start in correct place
        RATE    1,Hz(2)       ; 2 Hz output
HERE:   JUMP    HERE          ; output is running >Running
     'S CLRC    1             ; Stop output
WT:     WAITC   1,WT          ; wait for cycle end>Waiting
        RATE    1,0,HERE      ; stop and then idle
```

General control

These instructions do not change any outputs or read data from any inputs. They provide the framework of loops, branches and delays used by the other instructions.

DELAY	Do nothing for a set number of steps
DBNZ	Decrement a variable and branch if not zero
(LDCNTn,DBNZn)	Load counter 1 to 4 (V61-V64), decrement, branch if not zero
Bxx	Compare variables and branch (xx = GT, GE, EQ, LE, LT, NE)
CALL	Branch to a label, save return position
CALLV,(CALLn)	Like CALL, but load a variable (counter 1-4) with a value
RETURN	Branch to instruction after last CALL, CALLV or CALLn
JUMP	Unconditional branch to a label
HALT	Stops the sequencer and waits to be re-routed
NOP	This does nothing for one step (No OPeration)

DELAY

The DELAY instruction occupies one clock tick plus the number of extra ticks set by the argument. It produces simple delays of 1 to more than 4,000,000,000 sequencer steps.

```
        DELAY   expr|Vn|[Vn+off],OptLB
```

expr — The extra sequencer clock ticks to delay in the range 0 to 4294967295. The s(), ms() and us() built-in functions convert a delay in seconds, milliseconds or microseconds into sequencer steps.

Vn — Variable or table index from which to read the number of extra clock ticks.

OptLB — If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example uses display messages to tell the user what the sequence is doing.

```
        SET     1.00,1,0 ;run with 1 millisecond clock ticks
        DELAY   2999     ;wait 2999+1 milliseconds>3 second delay
        DELAY   s(3)-1   ;3 seconds -1 tick delay >3 second delay
        DELAY   V1,LB    ;wait V1+1 ms, branch    >variable delay
        DELAY   [V1+9]   ;V1+9 is table index     >table delay
```

DBNZ

DBNZ (Decrement and Branch if Not Zero) subtracts 1 from a variable and branches to a label unless the counter is zero. It is used for building loops.

```
        DBNZ   Vn,LB
```

Vn — The variable to decrement and test for zero.

LB — Instruction to go to next if the result of the decrement is not zero. From [11.00] you can use +n or -n (n is the number of steps) to branch to n steps after or before the current step.

DBNZ is often used with MOVI to set up loops, for example:

```
        MOVI    V2,1000    ;set times to loop
WT:     DIGOUT  [00000000] ;set all digital outputs low
        DIGOUT  [11111111] ;set them all high
        DBNZ    V2,WT      ;loop 1000 times
```

LDCNTn DBNZn

These obsolete instructions use variables V61 to V64 as counters 1 to 4. These instructions are preserved so that ancient sequences can still be used (but we urge you to recode them). New sequences should use MOVI and DBNZ, which can use any variable. LDCNTn loads a counter. DBNZn (Decrement and Branch if Not Zero) decrements a counter and branches while the counter is not zero. V61-V64 also store values for DACs 4-7.

```
        LDCNTn  expr|Vn
        DBNZn   LB
```

n — The counter to load or decrement in the range 1 to 4, using variables V61 to V64.

expr — The 32-bit integer value to load into the counter.

Vn — When a variable is used the counter is set to the 32-bit variable.

LB — Instruction to go to next if the result of the decrement is not zero. From [11.00] you can use +n or -n (n is the number of steps) to branch to n steps after or before the current step.

CALL CALLV CALLn RETURN

These instructions run a labelled part of a sequence and return. CALL, CALLV and CALLn save the next step number to a *return* stack and jump to the labelled instruction. The RETURN instruction removes the top step number from the return stack and jumps to it. CALLV also sets a variable to a constant. CALLn (n=1 to 4) is obsolete and sets the value of the variables that emulate the four counters, V61 to V64. These four variables are also used to store the outputs set for DACs 4-7.

```
        CALL   LB             ; use LB as a subroutine
        CALLV  LB,Vn,expr     ; Vn = expr, then call LB
        CALLn  LB,expr        ; V32+n = expr (n=1, 2, 3 or 4)
        RETURN                ; return to step after last CALL
```

LB The next instruction to run. This section of code should end with a RETURN. Programmers refer to called sections like this as *subroutines*. From [11.00] you can use +n or -n (n is the number of steps) to branch to n steps after or before the current step.

Vn CALLV copies the value of expr to this variable. The obsolete instructions CALL1 sets V61, CALL2 sets V62, CALL3 sets V63 and CALL4 sets V64.

expr A 32-bit integer constant that is copied to a variable. For CALLn only, if expr is 0, the variable is set to 256 to be compatible with previous versions.

You can use CALL inside a called subroutine. This is known as a *nested* CALL. If you call a subroutine from inside itself, this is known as a *recursive* CALL. The return stack has room for 64 return addresses. If you use more than this, the oldest return address is overwritten, so your sequence will not behave as you expect.

This example generates different pulse widths from DAC 0. The sequence is written to be independent of the sequencer rate (it must be high enough so that the widths are possible). In this case the rate is set to 5 kHz. The example sets DAC 0 to zero, then pulses for 20 milliseconds twice, once using CALL and once using CALLV. Then after a delay, there is a 50 millisecond pulse.

```
        SET    0.2,1,0        ; Run at 5 kHz, normal DAC scale
        DAC    0,0            ; make sure DAC0 is zero
        MOVI   V3,ms(20)-2    ; these two instructions...
        CALL   PUL            ; ...have the same effect as...
        CALLV  PUL,V3,ms(20)-2; ...this one. 20 ms pulse
        DELAY  s(1)-1         ; wait 1 second, then...
        CALLV  PUL,V3,ms(50)-2; ...a 50 ms pulse
        HALT                  ; So we don't fall into PUL routine
PUL:    DAC    0,1            ; set DAC value
        DELAY  V3             ; wait for time set
        DAC    0,0            ; set DAC back to zero
        RETURN                ; back to the caller
```

CALL/CALLV and RETURN let you reuse a block of instructions. This can make sequences much easier to understand and maintain. The disadvantage is the additional steps for the CALL and RETURN. If you need to set a variable, use CALLV and there is only the overhead of the RETURN instruction.

JUMP

The JUMP instruction transfers control unconditionally to the instruction at the label. Many instructions allow the use of an optional label to set the next instruction, so you can often avoid the need for this instruction. You can also jump using the contents of a register as the destination, or relative to a label (LB):

```
        JUMP   LB             ; Jump to label
        JUMP   (Vn),OptLB     ; use MOVI Vn,LB to set target
        JUMP   LB(Vn),OptLB   ; Jump to instruction LB+Vn
```

LB The label to jump to. From [11.00] you can use +n or -n (n is the number of steps) to branch to n steps after or before the current step.

(Vn) The value of variable Vn sets the instruction number to jump to.

LB(Vn) Jump to the instruction given by label LB plus the contents of Vn.

`OptLB` An optional label to jump to if `(Vn)` or `LB(Vn)` is not an instruction number. The first instruction is 0, the last depends on the size of the sequence. From [11.00] you can use `+n` or `-n` (`n` is the number of steps) to branch to n steps after or before the current step.

State machine

You can use the `JUMP` instruction to implement a state machine. A state machine is characterised by activities in each state, and transitions between states. State machines are a convenient way to tidy up a complicated sequence of operations involving conditions such as the level of an input signal. For example:

State	Activity	Transition
0	Wait for digital input 0 to be low	To state 1
1	Wait for digital input 0 to go high	To state 2 on high
2	Wait for digital input 0 to go low	To state 3 on low, digital output high on change
3	Wait for 10 seconds	To state 0, digital output low on change

This could be coded as:

```
            SET      1.000 1 0
            VAR    V1,State=State0 ;initial state
            VAR    V2,Until=0      ;Used to time state 3
            VAR    V3,Now

IDLE:       ...                    ;background actions >=
            JUMP   (State)         ;run state machine  >=

STATE0:     DIBNE  [.......0],IDLE ;wait for low       >State 0
            MOVI   State,State1,IDLE ;move on          >"
STATE1:     DIBNE  [.......1],IDLE ;wait for high      >State 1
            MOVI   State,State2,IDLE ;                 >"
STATE2:     DIBNE  [.......0],IDLE ;wait for low again >State 2
            DIGOUT [.......1]      ;Digital out high   >"
            TICKS  Until,sTick(1)  ;1 second ahead     >"
            MOVI   State,State3,IDLE ;                 >"
STATE3:     TICKS  Now,0           ;get current time   >State 3
            BLT    Now,Until,IDLE  ;wait for 1 second  >"
            DIGOUT [.......0]      ;Digital out low    >"
            MOVI   State,State0,IDLE
```

The `...` at label `IDLE` stands for instructions that you want to run in the background while the state machine runs. Of course, the more background instructions you include, the less frequently the state machine checks the state of the inputs.

HALT

The `HALT` instruction stops the output sequence and removes all overhead associated with it. It does not stop the sequencer clock, which continues to run. Any cosine output will stop, but will restart when the sequence restarts. To restart the sequencer, press a key associated with a sequence step or click a key in the sequencer control panel. If you associate a display string with this instruction, it appears in the sequencer control panel.

```
         HALT                  >Press X when ready
```

NOP

The `NOP` instruction (No Operation) does nothing except use up one sequencer clock tick. It can be thought of as the equivalent of `DELAY 0`.

Variable arithmetic

These instructions perform basic mathematical functions while a sequence runs. You can also compare variables and branch on the result.

ABS	Set a variable to the absolute value of another
ADD	Add one variable to another
ADDI,(ADDIL)	Add a constant value to a variable
MOV	Copy one variable to another
MOVI,(MOVIL)	Move a constant value into a variable
MUL,MULI	Multiply two variables, multiply by a constant
NEG	Move minus the value of a variable to another
SUB	Subtract one variable from another
DIV,RECIP	Division and reciprocal of variables

Compare variable

These instructions compare a variable with a variable or a 32-bit expression or a table entry and branch on the result. All comparisons are of signed 32-bit integers.

```
        Bxx     Vn,Vm,LB        ;compare with a variable
        Bxx     Vn,expr,LB      ;compare with a constant
        Bxx     Vn,[Vm+off],LB  ;compare with a table entry
```

`xx` This is the branch condition. The `xx` stands for: `GT`=Greater Than, `GE`=Greater or Equal, `EQ`=Equal, `LE`=Less than or Equal, `LT`=Less Than, `NE`=Not Equal.

`Vn` The variable to compare with the next argument.

`Vm` A variable to compare `Vn` with or table index variable.

`expr` A 32-bit integer constant to compare `Vn` with.

This example collects the latest data value from channel 1 (assumed to be a waveform), waits for it to be in a preset range for 1 second, then outputs a pulse to a digital output bit.

```
START:  CHAN    V1,1            ; get channel 1 data
        BGT     V1,4000,START   ; if above upper limit, wait
        BLT     V1,0,START      ; if too low, wait
IN:     MOVI    V2,S(1)/4       ; timeout, 4 instructions/loop
INLOOP: CHAN    V1,1            ; to check if still inside
        BGT     V1,4000,START   ; if above upper limit, wait
        BLT     V1,0,START      ; if too low, wait
        DBNZ    V2,INLOOP       ; see if done yet
REWARD: DIGOUT  [.......1]      ; Task done OK
        DELAY   S(1)            ; leave bit set for 1 second
        DIGOUT  [.......0]      ; clear done bit
        ...                     ; next task...
```

We want the data to be in range for one second. There are 4 instructions in the loop that tests this, so we set to the loop to run for the number of steps in a second divided by 4. For this to work correctly, the sequencer must be running fast enough so that 4 steps are no longer than the sample interval for the waveform channel.

MOVI

This instruction moves an integer constant into a variable. `MOVIL` is an obsolete instruction that does exactly the same thing. The syntax is:

```
        MOVI    Vn,expr,OptLB    ; Vn = expr
```

`Vn` A variable to hold the value of `expr`.

`expr` An expression that is evaluated as a 32-bit integer.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

MOVI is not the same as the VAR directive. The VAR directive sets the value of a variable when the sequence is copied to the 1401 and does not occupy a step. The MOVI instruction is part of the sequence and set the value of the variable each time the instruction is used.

MOV ABS NEG

The MOV instruction sets a variable to the value of another with the option of adding a 32-bit number and dividing by a power of two). The NEG instruction is identical to MOV except that the source variable is negated first. ABS is also the same, except that the absolute value (negative values become positive) of the variable is taken first. ABS was added at version 7.00. The syntax is:

```
        MOV     Va,Vb,expr,shift   ; Va = (Vb + expr) >> shift
        NEG     Va,Vb,expr,shift   ; Va = (-Vb + expr) >> shift
        ABS     Va,Vb,expr,shift   ; Va = (|Vb| + expr) >> shift
```

Va A variable to hold the result. It can be the same as Vb.

Vb A variable used to calculate the result. It is not changed unless it is the same variable as Va.

expr An optional expression that is evaluated as a 32-bit integer. If this argument is omitted, it is treated as 0.

shift An optional argument in the range 0 to 31, set to 0 if omitted. For all numbers except -1, the effect of one shift is equivalent to division by 2. The result of shifting -1 is always -1. Shifting is faster than division by a power of 2.

The following examples assume that V3 holds 1000:

```
        VAR     V6,Result
        MOV     V1,V3              ; set V1 to 1000
        NEG     V1,V3              ; set V1 to -1000
        MOV     V1,V3,-8           ; set V1 to 992
        NEG     Result,V3,0,4      ; set V6 to -63
        ABS     Result,Result      ; set V6 to 63
        MOV     Result,V3,4,1      ; set V6 to 502
```

ADDI

This instruction adds a 32-bit integer constant to a variable. ADDIL is an obsolete instruction that does exactly the same. There is no SUBI as you can add a negative number. The syntax is:

```
        ADDI    Vn,expr,OptLB      ; Vn = Vn + expr
```

Vn A variable to hold the result of Vn + expr.

expr An expression that is evaluated as a 32-bit integer.

OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs..

The following examples assume that V1 holds -1000:

```
        VAR      V1,Result=-1000
        ADDI    Result,1000         ; set V1 to 0
        ADDI    V1,-4000            ; set V1 to -5000
```

ADD SUB

The ADD instruction adds one variable to another. The SUB instruction subtracts one variable from another. In both cases you can optionally add a 32-bit integer constant and optionally divide the result by a power of two. The syntax is:

```
        ADD     Va,Vb,expr,shift ; Va = (Va + Vb + expr) >> shift
        SUB     Va,Vb,expr,shift ; Va = (Va - Vb + expr) >> shift
```

Va A variable to hold the result. It can be the same as Vb.

Vb A variable to add or subtract.

expr An optional expression evaluated as a 32-bit integer. If omitted, 0 is used.

shift An optional argument in the range 0 to 31, set to 0 if omitted, that sets the number of times to divide the result by 2.

The following examples assume that V1 holds -1000, V3 holds 1000, V6 holds 100:

```
VAR     V6,Result=100
ADD     V1,V3           ; V1 = 0 (-1000 + 1000 + 0)
SUB     V1,V3           ; V1 = -1000 (0 – 1000 + 0)
ADD     V1,V3,-8        ; V1 = -8 (-1000 + 1000 - 8)
SUB     Result,V3,0,2   ; V6 = -225 (100 – 1000 + 0)/4
ADD     Result,V3,4,1   ; V6 = 389 (-225 + 1000 + 4)/2
```

MUL MULI

MUL multiplies a variable by another variable, then optionally adds a 32-bit integer constant and divides the result by a power of two. MULI multiplies a variable by a 32-bit integer constant and divides the result by a power of 2.

```
MUL     Va,Vb,expr,shift ; Va = ((Va*Vb)+expr) >> shift
MULI    Va,expr,shift    ; Va = (Va*expr) >> shift
```

Va A variable to hold the result. It can be the same as Vb.

Vb A variable used to calculate the result.

expr An expression that is evaluated as a 32-bit integer. It is optional for MUL and required for MULI. If this argument is omitted, it is treated as 0.

shift An optional argument in the range 0 to 31, set to 0 if omitted, that sets the number of times to divide the result by 2.

The following examples assume that V1 holds –10 and V3 holds 10:

```
MULI    V1,10           ; V1 = -100 (-10 * 10)
MUL     V1,V3,-8        ; V1 = -1008(-100 * 10 -8)
```

DIV RECIP

DIV and RECIP divide variables. These are relatively slow instructions in most 1401s; they take around .4 µs in a Power2, 0.2 µs in a Power3, around 1µs in the Power1 I, 3 µs in a Micro2 and Micro3. The Micro4 implements hardware division, so has very little division overhead.

```
DIV     Va,Vb           ; Va = Va / Vb
RECIP   Va,expr         ; Va = expr / Va
```

If the numerator is 0, the result is 0. If the denominator is 0, the result is 2147483647 if the numerator is greater than 0 and –2147483648 if it is negative. The 1401 truncates all results towards 0. So, 7/3 or –7/–3 is 2, and –7/3 or 7/–3 is –2.

If you are dividing by a power of 2, it is faster to use a shift (see the ADD, SUB, MOV, ABS and NEG instructions). When dividing by a fixed value it is often faster to multiply and shift.

Variable Logic

These instructions perform basic bitwise logical functions between two variables or a variable and a constant while a sequence runs.

AND, ANDI	Bitwise AND of variables, variable and constant
OR, ORI	Bitwise OR of variables, variable and constant
XOR, XORI	Bitwise exclusive OR of variables, variable and constant

AND, ANDI

These instructions bitwise AND a variable with a variable or a 32-bit expression. A bitwise AND means that each bit of the 32-bit result is 1 if both corresponding source bits are 1, otherwise the result bit is 0. For example, 3 AND 1 is 1, 0x55 AND 0xAA is 0.

```
        AND     Va,Vb       ; Va = Va AND Vb
        ANDI    Va,Vb,expr  ; Va = Vb AND expr
```

Va The variable to hold the result.

Vb A variable to AND with Va or with the expression.

expr A 32-bit integer constant to AND with Vb.

This example waits for the digital input to have bit 4 set, then branches based on the digital input value (placed in VDigIn or V56 by WAIT).

```
       WAIT    [...1....]      ; wait for bit 4 set >Wait for Bit 4
       ANDI    V1,VDigIn,0x0f  ; isolate bits 0..3 (value 0-15)
       JUMP    ACTION(V1)      ; branch based on the result
ACTION: JUMP   ACT0            ; action for value 0
       JUMP    ACT1            ; action for value 1
       ...
       JUMP    ACT15           ; action for value 15
```

OR, ORI

These instructions bitwise OR a variable with a variable or a 32-bit expression. A bitwise OR means that each bit of the 32-bit result is 1 if either corresponding source bit is 1, otherwise the result bit is 0. For example, 3 OR 1 is 3, 0x55 OR 0xAA is 0xff.

```
        OR      Va,Vb       ; Va = Va OR Vb
        ORI     Va,Vb,expr  ; Va = Vb OR expr
```

Va The variable to hold the result.

Vb A variable to OR with Va or with the expression.

expr A 32-bit integer constant to OR with Vb.

XOR, XORI

These instructions bitwise exclusive OR a variable with a variable or a 32-bit expression. A bitwise exclusive OR means that each bit of the 32-bit result is 1 if the corresponding source bits differ, otherwise the result bit is 0. For example, 3 XOR 1 is 2, 0x55 XOR 0xAA is 0xff.

```
        XOR     Va,Vb       ; Va = Va XOR Vb
        XORI    Va,Vb,expr  ; Va = Vb XOR expr
```

Va The variable to hold the result.

Vb A variable to XOR with Va or with the expression.

expr A 32-bit integer constant to XOR with Vb.

Table access

Tables are declared with the TABSZ directive and can be populated with data using the TABDAT directive. Most access to tables is through the [Vn+off] method, but there are also instructions for loading and storing a variable in a table and for incrementing or decrementing a variable used as a pointer into the table.

TABLD,TABST	Load a register from the table and store a register to the table
TABINC	Increment a register and branch while within the table

TABLD TABST

These two instructions load a variable from the table and store a variable into the table. Many instructions can load arguments from the table, so TABLD is not often required.

```
        TABLD   Vm,[Vn+off],OptLB  ; load Vm from the table
        TABST   Vm,[Vn+off],OptLB  ; store Vm into the table
```

Vm — The variable to load from the table or store into the table.

+off — An optional expression that evaluates to an integer in the range –1000000 to 1000000. If omitted, 0 is used.

Vn — The variable value plus the offset is used as a table index. If the index lies in the table, Vm is loaded from the table or stored in the table at the index. If the index is not in the table, TABLD copies 0 to Vm and TABST does nothing.

OptLB — If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

TABADD TABSUB

These two instructions add a table value to a variable or subtract a table value from a variable. These instructions were added at version 7.00.

```
        TABADD   Vm,[Vn+off],OptLB  ; add table value to Vm
        TABSUB   Vm,[Vn+off],OptLB  ; subtract value from Vm
```

Vm — The variable to add data to or subtract it from.

+off — An optional expression that evaluates to an integer in the range –1000000 to 1000000. If omitted, 0 is used.

Vn — The variable value plus the offset is used as a table index. If the index lies in the table, Vm is changed. If the index is not in the table, the instruction does nothing.

OptLB — If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

TABINC

This instruction adds a constant to a variable and detects if the result is a valid table index. If it is a valid index, the instruction branches. If it is not, the result is reduced by the table size if it is positive and is increased by the table size if it is negative, and the instruction does not branch. This gives you an efficient way to work through the table in either direction.

```
        TABINC   Vn,expr,OptLB
```

Vn — This variable is assumed to hold a valid table index.

expr — This expression evaluates to a positive or negative number that is added to Vn.

OptLB — If present, branch to this label if Vn+expr is a valid table index.

For example, the following codes plays pulses through DAC 0 based on data in the table. The table data holds groups of three items, holding the time for the DAC to stay at 0, the DAC amplitude and the time to stay at the amplitude. Some example table data is given, but the data could also be set with the SampleSeqTable() script command.

```
        SET     0.100 1 0        ;run at 10 kHz
        TABSZ   12               ;4 sets of 3 items
        TABDAT  ms(50)-2,VDAC32(1),ms(50)-3
        TABDAT  ms(100)-2,VDAC32(1.3),ms(70)-3
        TABDAT  ms(200)-2,VDAC32(1.5),ms(90)-3
        TABDAT  ms(400)-2,VDAC32(1.9),ms(110)-3
```

```
      'G MOVI    V1,0            ;use V1 as the table pointer
LOOP:    DAC     0,0             ;strt with the DAC low
         DELAY   [V1]            ;wait for first period>Low
         DAC     0,[V1+1]        ;get the DAC value
         DELAY   [V1+2]          ;wait for second period>High
         TABINC  V1,3,LOOP
         DAC     0,0             ;tidy up the dac
```

Access to data capture

Most activities in the sequencer are independent of the sampling process. However, there are times when you need to know the value of a channel to decide what to do next. The CHAN command gives you the latest waveform value or number of events on a channel. The TICKS command tells you the current time in terms of the sampling clock ticks.

The sequencer can also send information in the other direction. If the digital marker channel is active you can force it to record the current digital input state with REPORT, or you can force it to record a marker of your own choosing with MARK.

REPORT,MARK	Simulate an external E1 pulse to record/set a digital marker
CHAN	Get the latest waveform value or event count from a channel
TICKS	Load a variable with the time in Spike2 time units
TICK0	Sets the zero time used as a reference for TICKS, new at 8.00

CHAN

This instruction gives the output sequencer access to sampled data on a waveform channel and to the number of recorded events on all other channel types. You can also use this command to get the most recent value written to the DAC outputs. The variable value is 0 if the channel is not being sampled. You cannot use this command on channels 30 and 31 (Text marker and Keyboard marker) as these channels do not exist in the 1401.

```
         CHAN    Vn,chn          ; Vn = ChanData(chn)
```

chn — The channel number is 1 to 100 for sampled channels or 0 to -7 for the last value written by the sequence to DAC 0 to 7. The result is the most recent data available. For a slow waveform channel this could be a long time in the past. In triggered sampling mode, waveform data is available between triggers.

Waveform and DAC data are treated as 16-bit signed values from –32768 to 32767. You also have more efficient access to DAC values as 32-bit data in variables V57 to V64 (VDAC0 to VDAC7) without the need to use the CHAN instruction.

This example waits for a signal to cross 0.05 volts and produces a pulse. We assume that channel 1 is a waveform.

```
         SET     0.100 1 0       ;run at 10 kHz
         VAR     V1,level=VDAC16(0.05) ;level to cross
         VAR     V2,data         ;to hold the last data
         VAR     V3,low=VDAC16(0.0)    ;some sort of hysteresis level
         DIGOUT  [00000000]      ;set all dig outs low
BELOW:   CHAN    data,1          ;read latest data   >wait below
         BGT     data,low,below  ;wait for below     >wait below
ABOVE:   CHAN    data,1          ;read latest data   >wait above
         BLE     data,level,above ;wait for above    >wait above
         DIGOUT  [.......1]      ;pulse output...
         DIGOUT  [.......0],below ;...wait for below
```

DAC values and arbitrary waveform output

You can only read back the current DAC value when you set it with a sequence instruction (DAC, ADDAC, RAMP, Cosine output). If you use arbitrary waveform output, this uses a separate, more efficient mechanism; the price you pay for the increased efficiency is less knowledge of exactly what the DACs are doing.

Event count overflow in version 8 onwards

You should be aware that when using CHAN to count events it is possible for the event count to exceed the 32-bit range of the variable. That is, the count will start at 0, then increase up to 2147483647 (0x7fffffff, the most positive number in 32-bit twos complement coding). The next event increments the number to 0x80000000, which is -2147483648 (the most negative number in 32-bit twos complement coding). The value will continue to increment by getting more positive and will reach 0 (again) at 4294967296 events, after which the cycle will repeat.

For example, if your events have an average rate of 100 Hz, the first wrap around happens after some 500 hours (more than 20 days). If this is likely to be a problem for you, you can probably reorganise your code to use differences of event counts.

TICKS

This instruction sets a variable to the current sampling time past the current zero time set by the TICK0 instruction in Spike2 time units (microseconds per time unit set in the sampling configuration) and adds an expression or 0 if expr is omitted. It can detect if the result does not fit in a signed 32-bit variable. Typical use of this instruction is to wait for a defined time while performing other sequencer actions, avoiding the need to carefully count sequencer instructions.

```
        TICKS   Vn,expr,ovLab     ; Vn = Spike2 time + expr, branch on overflow
```

Vn — A variable to hold the time past the current zero time plus expr. This may NOT be V255 or V256 as these are used to save the current zero time. Remember that variables are 32-bit signed numbers, but the time in clock ticks can be up to 64-bits in size. This means that there is a limit to the result that can be stored in Vn. Put another way, the result in Vn can overflow (see ovLab for a way to detect this). If overflow occurs, the value in Vn is full scale maximum (2147483647 or 0x7fffffff) if the result is too large to fit in 32 bits or full scale minimum (-2147483648 or 0x80000000) if the result is too negative.

expr — An expression that evaluates to a constant 32-bit signed number of clock ticks that are to be added to the result of the current time less the current zero time set by TICK0. This expression may be omitted in which case the value 0 is added. The sTick(), msTick() and usTick() expression functions can be used to make the sequence independent of the microseconds per time unit value.

ovLab — If present, this is a label that is branched to if the result of "current time - zero time + expr" does not fit in a 32-bit signed variable. If omitted, the next instruction runs regardless of overflow.

This can be used with the CHAN command and variable related branches to check the timing of external pulses. The sequencer runs under interrupt, and competes for time with other interrupt driven processes in the 1401 interface. This causes some “jitter” in the timing. The jitter is typically only a few microseconds.

Changes at Spike2 8.00

Because the time in Spike2 clock ticks no longer fits in a 32-bit variable, you should be aware that the result can overflow. To allow for this, the command is extended as described above to allow you to detect overflow and the new TICK0 sequencer command and the SampleSeqTick0() script command are added to give you control of where the TICKS result is relative to.

TICK0

This instruction sets the zero time used as a reference for the TICKS instruction to the current sampling time in Spike2 clock ticks (the timing resolution of the sampling data file). This was added at Spike2 version 8 as sampling can run for many more clock ticks than can be stored in a 32-bit sequencer variable.

If you have existing sequences, and you sample for less than 2147483648 clock ticks you can ignore this command and the old sequences will continue to work. However, if you sample for longer than this (31 minutes at 1 microsecond resolution, 5 hours at 10 microseconds resolution...) and you use TICKS, you should rewrite your old sequences to periodically use TICK0 and time relative to this.

The zero time is stored in V255 (bits 0-31) and V256 (bits 32-63). Be very careful if you modify these variables using other sequencer instructions if you want TICKS to give useful results.

```
TICK0 expr,OptLB
```

`expr` An expression that evaluates to a constant 32-bit signed number of clock ticks that are to be added to the current time to generate the new zero time. This expression may be omitted in which case the value added is 0. The `sTick(), msTick() and usTick()` expression functions can be used to make the sequence independent of the microseconds per time unit value.

`OptLB` An optional label that sets the next instruction to run.

REPORT MARK

The `REPORT` instruction records a digital marker (if the digital marker channel is enabled) as if there were an external pulse that triggered a digital marker. The `MARK` instruction does the same, except it takes the argument as the value to record. `REPORT` has no arguments.

```
        REPORT  OptLB
        MARK    expr|Vn|[Vn+off],OptLB
```

`expr` The argument should have a value in the range 0 to 255. If a variable or table is used, the bottom 8 bits of the value are used.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

```
        WAIT    [.......1]      >Waiting for bit 0
        REPORT                  ;save a marker when this is set
        MARK    12              ;set code 12 as a digital marker
```

Time resolution

There is a restriction in Spike2 that events on a channel should all be at different times. If the Spike2 data file is running with a time resolution that is similar to the speed of the output sequencer, it is possible that two consecutive `MARK` or `REPORT` instructions could be logged at the same time. This will cause Spike2 to report an error when sampling ends. If you get such an error you must either reduce the data file time resolution or increase the gap between the sequencer instructions.

Randomisation

These functions use a pseudo-random number generator. The generator is seeded by a number that is based on the length of time that the 1401 has been switched on.

`BRAND`, `(BRANDV)`	Random branch with a probability
`MOVRND`	Load a variable with a random number
`(LD1RAN)`	Load counter 1 (`V61`) with a random number 1-256

BRAND

`BRAND` branches with a probability set by the argument or by a variable. This could be used when several different stimuli are required, but in a random sequence. `BRANDV` is an obsolete name for the same instruction.

```
        BRAND   LB,expr|Vn|[Vn+off]
```

`LB` Where to go if the branch is taken

`expr` This is the probability of branching in the range 0 up to (but not including) 1.

`Vn` When a variable or table entry is used for a branch, the value is treated as a 32-bit unsigned number; 0 means a probability of zero and 4294967295 (the largest 32-bit unsigned number) means a probability of 0.9999999998.

```
        BRAND   LB,0.5      ;branch with 50% probability
```

To produce a multiple way random branch you use more `BRAND` instructions. A three way equal probability branch to `LA`, `LB` and `LC` can be coded:

```
        BRAND   LA,0.33333  ;Split the first route with p=1/3
        BRAND   LB,0.5      ;0.6667 to here * 0.5 is 0.3334 (1/3)
LC:     ...                 ;If neither of the above, comes here
```

The following shows the sequence for a five-way branch with equal probabilities:

```
        BRAND   LA,0.2  ;5 way, LA probability is 0.2 (1/5)
        BRAND   FX,0.5  ;Probability to here=0.8, so to FX=0.4
        BRAND   LB,0.5  ;Probability to here=0.4, so to LB=0.2
LC:     ...             ;Probability to here=0.2
FX:     BRAND   LD,0.5  ;Probability to here=0.4, so to LD=0.2
LE:     ...             ;Probability to here=0.2
```

The best technique is to reduce the branches to a power of two as soon as possible. Case 1 of the five-way branch is split off (probability of 0.2), leaving 4 ways. The 4 ways are split with a probability of 0.5 (0.4 for each division) then the last two routes are split, again with a probability of 0.5 (0.2 for each division).

Poisson process

In a Poisson process, the probability of something happening per time interval is constant. You can generate a delay with a Poisson statistic by:

```
POISSON: BRAND   POISSON,prob ; poisson delay
```

The probability is given by `prob = 1.0 - 1.0/(mDelay*S(1))`, where `mDelay` is the mean delay required in seconds and `S(1)` is the built in function that tells us how many steps there are per second. If you would rather express this in terms of a rate, then `prob = 1.0 - rate/S(1)`, where `rate` is the expected rate in Hz.

```
TENHZ:  BRAND   TENHZ,1.0-10/S(1) ;10 Hz mean rate
        DIGOUT  [.......1]        ;set output high
        DIGOUT  [.......0],TENHZ  ;set output low, goto TENHZ
```

This example generates a digital output that pulses to produce an approximation to a Poisson distributed pulse train with a mean frequency of 10 Hz. The approximation improves the shorter the step time. The mean interval between pulses is 100 milliseconds plus the time for 2 steps and the shortest gap between pulses is 3 sequencer steps.

Scripts and variables

From a script you can set sequencer variables as 32-bit signed integers. For the range 2147483648 to 4294967295 we must use negative numbers. This script example shows you how to convert a probability into a variable value and pass it to the sequencer:

```
func vrand%(prob)
if (prob >= 1.0) then return -1 endif;
if (prob <= 0.0) then return 0 endif;
if (prob >= 0.5) then prob := prob-1.0 endif;
return prob * 4294967296.0; 'is converted to an integer
end;

Proc SetBrandVar(prob, v%) 'prob is probability, v% is variable
SampleSeqVar(v%, vrand%(prob));
end;
```

MOVRND

This instruction generates a random number in the range 0 to a power of 2 minus 1, then adds an integer constant to it and stores the result in a variable.

```
        MOVRND  Vn,bits,expr
```

`Vn` The variable to hold the result.

`bits` The number of random bits to generate in the range 1 to 32. The generated random bits fill the variable starting at the least significant bit. Bits above the highest numbered generated bit are set to 0.

`expr` An optional expression that evaluates to a 32-bit integer number, that is added to the random bits. If this is omitted, nothing is added.

Expressed in terms of the script language, the random number is one of the numbers in the range `expr` to `expr+Pow(2,bits)-1`. For example, `MOVRND V61,8,1` emulates the obsolete `LD1RAN` instruction that generates a random number in the range 1 to 256.

The following code fragment implements a random delay of between 1 and 2.024 seconds (assuming a 1 millisecond clock).

```
MOVRND  V1,10,998 ;load V1 0 with (0 to 1023) + 998
DELAY   V1        ;this uses 999 to 2023 steps
```

LD1RAN

This obsolete instruction loads counter 1 (`V61`) with a pseudo-random number in the range 1 to 256. It is implemented as `MOVRND V61,8,1`. Do not use this instruction in modern code. `V61` is also used to store the last value written to DAC 4.

Arbitrary waveform output

In addition to generating voltage pulses, ramps and cosine waves through the DACs, Spike2 can play arbitrary waveforms. The sequencer can start waveform output, test it and branch on the result, stop output or set one more output cycle. Each waveform has an associated code (often a keyboard character). The sequencer uses the code to identify the arbitrary wave to replay. The waveforms are stored in 1401 memory and can be updated during sampling with the `PlayWaveCopy()` script command.

`WAVEGO`	Start or prepare arbitrary waveform output area
`WAVEBR`	Test arbitrary waveform output and branch on the result
`WAVEST`	Start or stop arbitrary waveform output

WAVEGO

The `WAVEGO` instruction starts output from a play wave area, or prepares the output for an external trigger. Starting output can take more time than we want to allocate to a sequencer step so `WAVEGO` sets a flag and output starts as soon as the 1401 has free time (usually within a millisecond). See `WAVEST` for a precisely timed start to output.

```
WAVEGO  code{,flags{,OptLb}}
```

`code` This is either a single character standing for itself, or a two digit hexadecimal code. This is the code of the area to be played.

`flags` These are optional single character flags: `W`, `T` and `-` meaning no flag. The flags are not case sensitive. Use `T` for triggered waveform output (either an external hardware trigger or using `WAVEST T`). Use `W` to make the sequencer wait at this step until the 1401 has prepared the hardware to play. The `-` flag was new at versions [10.0, 9.08, 8.17] and is required if you have a label and no flags. Examples:

```
WAVEGO  X,-,Label ;area X, no wait, no trigger, go to label
WAVEGO  23,T      ;area coded hexadecimal 23, triggered
WAVEGO  0,WT      ;area '0', wait until trigger armed
WAVEGO  1,W       ;area '1', wait until play started
```

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs. It can be useful to issue the `WAVEGO` command with no flags and a label; to do this, use `-` or `--` as the flags: `WAVEGO X,-,OptLB`.

If you need to know when the output started, use the `WAVEBR T` option. If you need to know that the request to start the playing operation has been honoured, but you do not want to hang up the sequencer with the `W` option, use the `WAVEBR W` option.

`WAVEGO` cancels existing output just before the new area starts to play. If you use `WAVEBR` after the play request, unless you use the `WAVEGO` or `WAVEBR W` option to be certain that the new area is active, the `WAVEBR` result may be based on the previous area.

Warning
It is quite common to have a key associated with the WAVEGO command so that a key press can trigger the waveform output. Keypresses as also associated with play wave areas and can trigger them. It can be tempting to use the same key code for both a play wave area and for the WAVEGO command; this will cause a race between the code that starts the play wave area due to the key (which will win) and the sequencer code. The area will start to play, then stop, then start again.

WAVEBR

The WAVEBR instruction tests the state of the waveform output and branches on the result. No branch occurs if there is no output running or requested.

```
        WAVEBR  LB,flag
```

LB Label to branch to if the condition set by the flag is not met. From [11.00] you can use +n or -n (n is the number of steps) to branch to n steps after or before the current step.

flag An optional single character flag to specify the branch condition:

- W branch until a WAVEGO request without the W flag is complete.
- C branch until the play wave area or cycle count changes.
- A branch until the play wave area changes.
- S branch until the current output stops.
- T branch until output started with WAVEGO begins to play.

The following sequence tracks the output when we have two play areas labelled 0 and 1. Area 0 is set to play 10 times and is linked to area 1. The sequence below will track the changes. Play wave DAC output happens one play wave clock tick after the output is changed, so the sequencer can know that a DAC output is about to change.

```
        WAVEGO  0,WT      ; area 0, wait for armed
        MOVI    V1,5      ; load variable V1 with 5
WT:     WAVEBR  WT,T      ; wait for external trigger>Trigger?
W5:     WAVEBR  W5,C      ; wait for cycle   >Waiting for cycle
        DBNZ    V1,W5     ; do this 5 times  >Waiting for cycle
WA:     WAVEBR  WA,A      ; wait for area    >Wait for area
WE:     WAVEBR  WE,S      ; wait for end     >Wait for end
```

The WAVEGO command requests a triggered start and waits until the trigger is armed before moving on. It then waits for an external trigger at the WT label. Next the sequence tracks the end of 5 output cycles. At label WA the sequence waits for the area to change and finally the sequence waits for output to stop.

WAVEST

The WAVEST instruction can start output that is waiting for a trigger and stop output that is playing, either instantly, or after the current cycle ends.

```
        WAVEST  flag
```

flag This are optional single character flag and specifies the action to take:

- T Trigger a waveform that is waiting for a triggered start.
- S Stop output immediately, no link to the next area.
- C Play to the end of the current cycle, then end this area. If there is another area linked it will play.

The following code starts output with an internal trigger and then stops it after a delay.

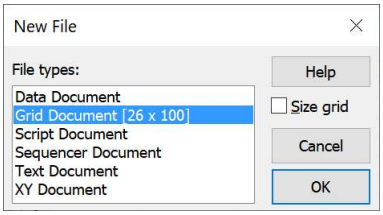
```
        WAVEGO  X,TW          ; arm area X for trigger, wait for armed
        WAVEST  T             ; trigger the area
        DELAY   1000          ; wait
        WAVEST  S             ; stop output now
```

6: File menu

File menu

The File menu is used for operations that are mainly associated with files (opening, closing and creating) and with printing. The menu ends with a list of recently used files.

New File



This command creates a new Spike2 data file for sampling based on the current sampling configuration, a grid view, an output sequence file, a script file, text file or an XY file. The command opens a New File dialog box in which you choose the type of file to create. You can activate this command from the menu and from the toolbar. The script language equivalent is `FileNew()`.

You can make files of six types: data, grid output sequencer (pulse) files, script files, text files and XY files. Double-click a file type, or select a file type and click OK; Spike2 will open a window of the specified type. Each file type has its own file name extension.

Data Document

A Time view opens plus additional windows set by the sampling configuration. The time view holds the set of channels defined by the current sampling configuration and is ready to start sampling. Data documents are not stored in memory, unlike most new files, but are kept on disk. Until they are saved after sampling they are temporary files in the directory set in the Edit menu Preferences. The file name extension is `.smrx` (or `.smr` for the legacy format). Only one sampling document can be open at a time.


You can also open a new file, ready to sample, by clicking this button on the system toolbar or by using the Run Now button in the Sampling Configuration.

Grid Document [columns x rows]

Select this option to open a grid view with a default number of rows and columns. You can create a grid view with your choice of rows and columns from the script language. The file name extension is `.s2gx`. When you select a grid view in the dialog, a new check box, Size grid, appears. If this box is checked, OK leads to the Grid Size dialog, to modify the size of the new grid. Otherwise, the grid is created with the current size.

Script Document

A script editor window opens in which a new script can be written, run and debugged. The file name extension is `.s2s`.

Sequencer Document

A new window opens in which you can type, edit and compile an output sequence. The file name extension is `.pls`.

Text Document

Text files can be used to take notes, build reports and to cut and paste text between other windows and applications. The file name extension is `.txt`.

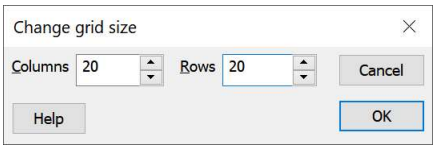
XY Document

XY windows are used to draw user-defined graphs with a wide variety of line and point styles. The Windows file name extension is `.sxy`. They can be generated by scripts, or from the Analysis menu for trend plots. Although we allow you to open them from the New File command, there is little point as you cannot do much with an empty XY view.

Result document

Result documents are NOT generated from this menu. Instead, they are generated from Data documents displayed in a Time view using the Analysis menu New Result View commands.

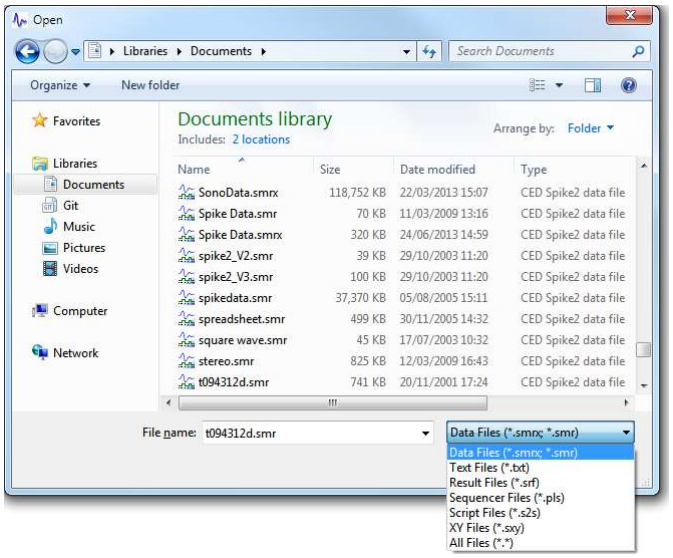
Grid Size dialog



The Grid Size dialog is used from the New File dialog to set the initial size of a grid view and from the Grid view View menu Grid Size... command to resize an existing grid. The dialog display the initial grid columns and rows; modify these values as required and click OK to set the new grid size.

Columns must be in the range 1 to 1000, rows in the range 1 to 1000000. As each cell in the grid occupies memory, setting a huge number of cells will result, in the best case, with slow operation and in the worst with Spike2 running out of memory. It will also make saving grids to disk and reading them back slow if a large proportion of the cells are not blank.

Open



This file menu and toolbar option opens a dialog in which you can select a file. The dialog features depend on the operating system. You can open six file types with this command: a Spike2 data file, a Spike2 result view, a text file, a script file, an output sequence file or an XY file. The type of the file is selected with the Files of type field.

When you select a data file, Spike2 looks for a resource file of the same name, but with the file extension `.s2rx` or `.s2r`. If this is found, the new window displays the file in the same state and screen position as it was put away. Several windows may open if the file was closed with the `Ctrl` key held down. See the Close command details.

Spike2 opens an appropriate window type to match the selected file type.

You can also open files by selecting them from the Most Recently Used list at the end of the File menu.

Read only files

You can open Spike2 data files that are read-only, but you are not allowed to write any changes back to the file. Read-only data files have [Read only] added to the window title. Unless the file is on a read-only medium (such as CD) you can usually clear the read-only state by opening the Property page of the file (right click on the file name or a list of selected files) and clear the Read-only check box. If you copy files from a read-only drive such as a CD, the copied file is marked read-only.

Read-only text-based files open normally, but you cannot edit them.

Some commands that rely on writing will not work with read-only files.

Import

Spike2 can translate data files from other formats into Spike2 data files. We make our best efforts to import data, but can only import data that is compatible with Spike2 and for which we have knowledge of the input format.

The File menu Import command is a pop-up menu with two option: Import and Options. Import leads to a standard file open dialog in which you select the file type and file name to convert. Options opens a separate

dialog to set options that are common to all import types. Spike2 remembers the last importer you used and offers that by default.

We are open to adding additional formats. To do so we need a description of the format and some example files. Items marked NO LONGER SUPPORTED are not tested (as we have no example data files) and will be removed in the future unless we get some indication that they are still useful (which will require example files for import).

Import dialog

You then set the file name for the result; Spike2 will suggest the same name with the extension changed to `.smrx`. The details of the conversion depend on the file type.

Supported formats include text files with data in columns, the CFS files used by CED programs such as Signal and SIGAVG, Spike2 for Macintosh files as well as data from many third party vendors. Spike2 searches the `import` folder in the Spike2 installation folder for CED File Converter DLLs. The script language command to convert files is `FileConvert$()`.

We will write the new files as a 64-bit `.smrx` file. If you want the output as a 32-bit `.smr` file you can export the result, or use Spike2 version 7 to import the data.

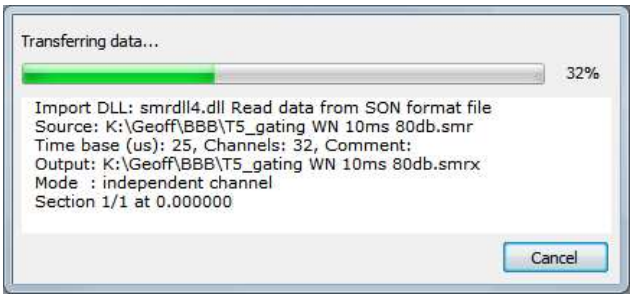
Data import has two main phases:

1. Scanning the input files to detect the file contents, channel types and data and time ranges present.
2. Transfer of data into a Spike2 data file.

The scanning phase is handled by the file converter DLL Data transfer is a co-operative effort between Spike2 and the converter DLL.

Display during import

Data file import can take a considerable time, especially if the source is very large or spread over many files. Spike2 opens a resizeable progress dialog displaying a summary of the import procedure; it also gives you the option of cancelling the import. This dialog may be unresponsive with older importers, particularly during the scanning phase. Modern importers give feedback during scanning but in some cases, you may not be able to cancel import until the data transfer phase begins.



The progress dialog closes when the file has been transferred; the text contents of the dialog are written to the Log view when the file closes, which can be useful if there were problems during the import.

Importer Configuration Files

Some importers store configuration files in the import folder; these files have the extension `.icf`. The last used configuration for importer `XXX` is saved in `XXX_Last.icf`. If a default format exists, it is saved in `XXX_Def.icf`. If the `_Def.icf` file exists, it is used, otherwise the `_Last.icf` file is used. Most importers will survive without any configuration information as the built-in default settings will do something useful. However, if you intend to import a binary or text file with the `FileConvert$()` script command it is essential that you have created configuration files, otherwise these importers will not know what to do with the data.

Import strategy

We import files of many different types. We have two basic models of external file:

1. Files with essentially independent channels that can stop or start at any time.
2. Files holding repeated frames/sweeps/episodes of data (such as captured by the CED Signal program).

The first type of file maps nicely into the Spike2 view of data, and such files usually import and look much the same as their source. The second type can be more of a problem if the file does not specify the start time of each sweep. For the second type, we import the sweeps (adjusting the start times so that sweeps do not overlap) and we add a Marker channel holding the start and end of each sweep. The Marker channel is added as channel 31 (if channel 31 is not in use or if it is in use as a Marker channel), otherwise we add a Marker channel after all imported channels.

64-bit Spike2 build problems

Some of these importers rely on DLLs or object libraries supplied by third parties. If these DLLs or libraries only exist in 32-bit versions, the 64-bit build of Spike2 cannot use them. These are marked as "32-bit only" in the Notes (below); to import data for these you must install the 32-bit version of Spike2.

DLL name

The DLL column in the table below holds the name of the import DLL in the import folder and is provided for script users who wish to force a particular DLL to be used to import your file.

File types we import

We import data from a wide variety of file types and we attempt to preserve as much of the original data content and accuracy as we can. If there is a file type that you would like imported that is not in the list, please contact us. To import a new type we need accurate file format information. We would like to thank the companies that made such information available to us for their help. Some companies do not release their file formats; in a few cases we have managed to figure out enough to extract useful information. In cases where companies do not release the information and we have not been able to figure it out, you can sometimes export data in other formats that we can import; of course, every change of format will tend to reduce the quality and quantity of data that is imported.

If you find a problem in any importer, please contact us and explain the problem; in many cases the importers were written with only a small number (sometimes 1) of examples of the source data. We will need a source data file that fails to convert; please send us the *smallest* file that fails! Gigabyte-sized files are time-consuming to transfer and analyse.

Data source	Extensions	DLL	Types imported	cmd$	Notes
ADInstruments LabChart	`adidat, adicht`	`adi`	RealWave, TextMark	Yes	Added October 2019 using the support software from LabChart 8.1.13. At [10.18] we set the date and time of the first data section.
Allego	`xdat`	`allego`	Waveform		NeuroNexus data files. Added March 2022.
Alpha MED Sciences	`DAT,MODAT`	`alphamed`	Waveform		Conductor, Performer and Mobius data. Can import files larger than 2 GB.
Alpha Omega Engineering	`MAP, MPX`	`alphaomega`	Waveform, WaveMark, RealMark, Event		Old format event data and WaveMark supported. TextMark, MPX supported. Fixed y axis scaling problem in Jan 2014. Copes with discontinuous waveform data from June 2015. [8.09] fixed potential channel alignment and crash problems, set file resolution as close to 1 us as possible and made it faster. [9.03] import Text stream comments. [10.20] handles gain settings in recent files.
Axon Instruments	`DAT,ABF`	`axon`	Waveform, TextMark		(Molecular Devices) **32-bit only** (so not available in the 64-bit build of Spike2). Imports 16-bit integer and 32-bit float. Library updated Jan/2014.
Axona	`BIN`	`axona`	Waveform		There is other information in this file type (digital and status inputs, digital and stimulus outputs, video tracking), but the importer currently only imports the 64 channels of waveforms.

Data source	Extensions	DLL	Types imported	cmd$	Notes
Binary data	BIN,DAT	binary	Waveform, RealWave	Yes	Can import 8/16/32/64 bit signed and unsigned integers data in big and little-endian formats.
Bionic/ Cyberkinetics	NEV,RND,NS*	bionic	Waveform, Marker, WaveMark, Event		Can import files larger than 2 GB. Handles version 3.0 files [10.21].
Biopac	ACQ	biopac	Waveform, TextMark		Imports 16-bit integer and 32-bit float data. Jan 2018: Can import older-formatted big-endian files.
BrainVision	*.vhdr	brainvision	Waveform, Marker	Yes	Imports 16-bit integer and 32-bit float data, plus markers.
CED CFS	DAT,CFS	cfs	Waveform, RealWave		Jan 2014: corrected marker channel times when block has a negative time offset.
COLD_Datein	*	coldadc	Waveform, TextMark		Pulsion Medical Systems.
CONSAM	SSD,DAT	consam	Waveform		From Prof. D. Colquhuon. Supports versions 1001, 1002.
DATAPAC	PAR,PBR,PCR	datapac	Waveform, RealWave, Event		Requires a .dat file holding waveform data and events require a .eft file.
DATAQ Instruments	WDQ	codas	Waveform, TextMark		DataQ Instruments Codas files
DataWave	UFF,CUTEWB	datawave	Waveform, WaveMark, TextMark, Event		Imports both Discovery and Workbench files.
Data Sciences International	*	dataq	Waveform, RealMark, TextMark		**32-bit only.** Imports files up to version 5. You need a DSI dongle to import version 5 files. Contact CED for details. Files with 4 character extensions supported. From [9.04] we amalgamated the dataqv5 and dataq DLLs into dataq.dll. See Ponemah importer for version 6 files.
Delsys Files	EMG	delsys	Waveform		Version 4 supported.
elmiko medical	*	elmiko	Waveform		
European Data Format(+)	EDF,BDF	edf	Waveform, TextMark	Yes	Since [7.06] we import dates so time of day display can be used. Imports 16 and 24-bit data. From [8.09] we import with a better time resolution, convert 24-bit data to RealWave and group annotations. [10.06] recognises more BDF types. [10.08] fixed an offset problem with asymmetrically-scaled integer data.
Grass-Telefactor	BIN	polyview	Waveform, TextMark		(PolyView) We import versions from 2.0. NO LONGER SUPPORTED (due to lack of test files).
HLR Data Format	HLR	hlr	Waveform, RealMark, Event		NO LONGER SUPPORTED (due to lack of test files). From software Platon and Pyton version 3.00 and 4.00.

Data source	Extensions	DLL	Types imported	cmd$	Notes
Heka Data Format	DAT	heka	Waveform		Since [7.06] we support old format Apple Macintosh files and time of day display mode. From [8.12] the experiment name can be up to 80 characters. From [9.03] we import multiple sweeps. From [10.20] we support floating point data import.
Intan	*	intan	Waveform		From [9.04] we import .rhd and .rhs files and read digital channels from .int files. We import "Traditional Intan" format and not "One File Per Channel" format.
Multi Channel System (MC_Rack)	MCD	mc_rack	Waveform, WaveMark, Event		We can import 12 and 16-bit data. Library updated Aug/2015, when 64-bit support was added. In order to use the MC_Rack importer you must have the MC_Rack software from Multi Channel Systems installed on your computer.
MindSet	BIN	mindset	Waveform		Mindset (16/24) data Files: MindSet, MindMeld
MindWare	MW	mindware	Waveform		
Motion Lab Systems	C3D	c3d	Waveform, Marker		From 8.12 the length of channel units was limited to 32 characters. From 9.03 we fixed bugs that caused incorrect channel scales and labelling and occasional bad data points. From 10.19 we support files with more than 65535 frames and SGI/MIPS sourced files and import Markers. The latest file specification and example files can be found in the c3d.org web site.
Native ADC	ADC	nativeadc	Waveform, TextMark		Used by Polish universities as a common data format. NO LONGER SUPPORTED (due to lack of test files).
NeuroScan	EEG, CNT	neuroscan	Waveform, Event		Can read both 16 and 32-bit data. Importer was based on file version 0.0 from May 1995.
NewBehavior	HEX	neurolog	Waveform		(Neurologger) old and 2012 formats. Jan/2014: added a dialog to select the recording session to import. Modern devices from this source export data as EDF files.
Neuralynx	NEV,NCS, NTT,NSE, NST	neuralynx	Waveform, WaveMark, TextMark		Waveform channel y scaling factor corrected Jan/2014.
OT Bioelettronica	otb+	otbplus	Waveform, RealWave		Tested with files from Sessantaquattro, Quattrocento, Due Pro and Muovi devices.
Percept PC	JSON	perceptpc	RealWave, RealMark		Medtronic™ Percept PC JSON file importer.

Data source	Extensions	DLL	Types imported	cmd$	Notes
Plexon	NEX,PLX, DDT,PL2	plexon	Waveform, WaveMark, TextMark, Event		Can read data generated by the Plexon version 1.07 library. Spike2 9.04 fixed a problem reading files with multiple sections. Spike2 10.14 has further fixes.
Ponemah 6	PnmExp	ponemah	Waveform	Yes	Data is imported as RealWave. Currently you cannot import Marker data. Spike2 10.13 fixed a timing problem with files with multiple sections and allowed import filtering by subject, channel and time range.
RC Electronics	PRM,INX, DAT	rcelectric	Waveform, TextMark		NO LONGER SUPPORTED (due to lack of test files). Imports 12 and 16-bit data.
Ripple Neuro	NEV,NS*, NF*	ripple	Waveform, WaveMark, Event, RealWave		Extends the NEV file format to handle floating point data (NF* files). Special treatment of some channels. Handles version 3.0 files [10.21].
Text files	TXT,ASC, CSV,TSV	ascii	Waveform, RealWave, Event, TextMark	Yes	Fixed a TextMark timing issue in Jan/2014. Treats .csv files as comma separators only, .tsv as tab separators only.
TMS International	S00,Poly5	tmsi	Waveform		Added Poly5 extension support in March 2022.
Tucker-Davis Technologies	TSQ,tev, sev	tdt	Waveform, WaveMark, Event, Marker, RealWave		You also need the .tev file to import the data. February 2019: Added *StrobeOff* events and fixed missing WaveMark data. Added .sev import in October 2019. Strobe value displayed as floating point in March 2022.
WAV (Microsoft)	WAV	mswave	Waveform		Handles 8, 16, 24 and 32-bit integer WAVE_FORMAT_PCM format data. From Spike2 8.07 we handle the same formats in WAVE_FORMAT_EXTENSIBLE files. 32-bit support added at version 9.03.
WaveMetrics Igor Pro	IGO,IGR, IBW,PXP	igor	Waveform		Imports both PC and Mac files up to and including version 5.
LabRecorder	XDF	xdf	Waveform, RealWave, TextMark		Extended in March 2022 to read RealMark and TextMark data and deal with gaps in waveforms.
Xltek Neuroworks	STC	neuroworks	Waveform, Event, TextMark		Imports version 3.3 onwards. The .stc file holds a list of all the data files that make up the data (.erd/ .etc). We import all the raw waveforms, an event channel of Trigger data, and user-comments made during sampling (.ent) and a channel of TextMark data indicating the original raw data file starts.

Import DLL versions

Spike2 supports several import DLL versions from 4 to 7. Version 4 DLLs are the same as those used by Spike2 version 7 and are limited to importing 32-bit time ranges and the output file size is limited by the 32-bit SON file size limit of 1 TB. The version 5 DLLs were introduced to support 64-bit times and file sizes limited only by your disk sizes and patience (though some importers may be limited by available system memory). The version 5 import DLLs can also provide feedback into the progress dialog during the scanning phase (though this is up to the DLL author). Version 6 DLLs are functionally the same as version 5, but have internal software differences. Version 7 DLLs are functionally the same as version 6, but share the Spike2 system DLLs rather than being independent of Spike2, which makes them much smaller and significantly faster to load.

When you select a file to import, you must first select a file type. If the file convert DLL is not the latest version (7), we add [version] to the end of the file importer. For example, if you had a version 4 importer for old Spike2 data files the file type list would display:

Spike2 files (*.SMR)[4]

to indicate that this was an old style importer.

Writing an import DLL

If a data format is generally available and sufficient users want to import it, we are usually prepared to write an import DLL to support the format.

However, you may have an exotic format of your own that no-one else uses, or you may have a private format that you do not wish to reveal to third parties. In these cases, you may wish to write your own import DLL. To do so you will need to be able to write code in C or C++ in the Windows environment. We have documentation available on the DLL interface. Contact CED for more information.

Import problems

Before contacting CED with a problem, please read about importing files and read the any notes for your particular file type.

If there are disk read problems or the input is damaged the import is likely to end with an error message. However, there are other problems that can occur during data import that we can work around.

Data is input section by section. Some files may have only one section, others may have many. A section can refer to a single sweep of data from a sweep-based capture system or it may refer to data captured after a particular time marker; it is a flexible concept.

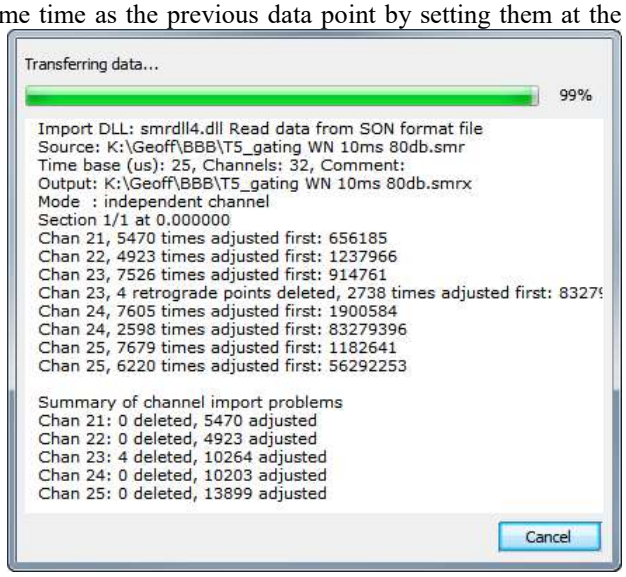
Within each section, data is captured channel by channel in ascending time order.

The output data is written in strictly ascending time order into the Spike2 data file. Spike2 data files have the rule that within a channel, all data is in ascending time order and no two items can occur at the same time. There is a basic quantum of time, the underlying clock tick, that all items are timed with. This is typically of order 1 microsecond and usually no more than 50 microseconds; it can be much smaller.

When we import data, we check that all the data meets these requirements. When we import event-based channels (event, level event, marker, WaveMark, TextMark and RealMark data), we read the data block by block. We perform the following operations on each block:

1. If we have to apply a time offset, we do so (this can happen when importing sweep-based data where each sweep has an independent time base)
2. We check that the data is in ascending order of time
3. If not in ascending time order we sort it into order
4. We delete data at negative times or at times that are beyond the tick range supported by Spike2
5. We delete data that occurs before the data at the end of the previous data block on this channel. This is labelled as *retrograde points*. This usually indicates either a damaged or badly-written file or it could be a fault in the importer. If this is a problem with your data you could consider contacting us about it.
6. If we have deleted retrograde points and the next point is at the same time as the last time in the previous block, we delete that as well as it is unlikely to be useful (new at version [10.16]).

7. We adjust the times of data that occurs at the same time as the previous data point by setting them at the time of the previous data plus one tick. This type of adjustment usually occurs as the result of "bouncy switches", but this can occur legitimately. For example, if a system wants to drop a lot of text at a particular time it might code the text in 80 character chunks all with the same time stamp.



This is an example of a data file with two types of problem. As we process each data block we generate a message warning about deleted items and adjusted times. Where a time is adjusted, we display the time (in clock ticks) of the first adjusted point in the block. This can help you locate it afterwards to diagnose the problem. We will display up to 100 warnings per channel, after which the file is processed silently. If there are any adjustments needed, a Summary section is added to the output (and to the Log file).

ADInstruments importer

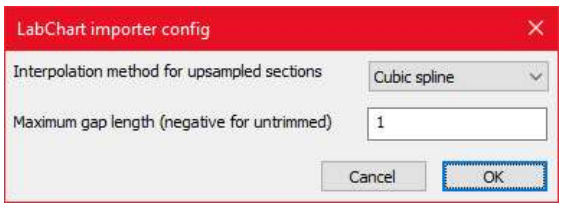
This importer reads `*.adidat` and `*.adicht` files through a software toolkit supplied with LabChart. We used the toolkit from version 8.1.13.

The ADI file format holds channel data in sections, and within each section the channels available can change and so can the channel sampling rates. To allow us to import this into Spike2 where each channel has a fixed sampling rate over the entire file and where a file has a continuous time base, though channels can start and stop, we pre-process the ADI file as follows:

1. Scan the file to identify all channels holding raw waveform data and comments. We do not import plot views or channels built from calculations or markers based on filters.
2. Scan all the sections to identify the highest sampling rate used for each channel within the file.
3. We import the data into a Spike2 data file holding all the identified channels sampled at the highest detected rate.

Configuration options

There are two options that control how we map between the ADI data file and a Spike2 file: Interpolation method and Trim gaps. These methods are set interactively by the importer Configuration dialog:



Interpolation

As the input data file may hold data sections that are sampled at a slower rate than the rate used by Spike2, we need a method to fill the gaps. There are currently four options:

Method	`cmd$`	Action
Cubic spline	`fill=cubic`	If your data is band-limited to half the sampling rate of the section, this will likely give the best result. It fits a cubic spline through the input data points to generate the values between the points.
Linear interpolate	`fill=linear`	This draws a straight line between the input data points to predict the values between them.

Skyline	`fill=square`	This sets all the 'missing' points to the value of the previous input.
Zero fill	`fill=zero`	This sets all the 'missing' points to zero.

If your data files do not hold sections with changing sampling rates, it does not matter what interpolation method you set as the data will not be interpolated.

Gap between sections

ADI data files hold data in sections, where each section starts at the same time across all the channels within it, though some of the channels may hold no data. These sections can be widely separated in time, so we give you the option of replacing all the gaps by a fixed, user-settable gap, in seconds. To set a gap from the script using the `cmd$` argument, use `gap=seconds` where seconds is the time gap you want in the range 0.0 to 100 seconds, or set a negative gap to use the times in the original file. For example: `gap=1.0` for 1 second maximum gaps or `gap=-1` to use the timing in the file. If you set a negative gap, you can use Time of Day mode on the Spike2 x axis to review data by the wall clock time.

We have an example file where consecutive waveform data blocks on the same channel overlap in time by one or more points. We cope with this by allowing the second block to overwrite the first.

TextMark import

We attempt to import comments in the file. Comment lines of more than 99 characters are truncated. You can view the comments in a table by double-clicking a comment in the time view. We use the 4 marker codes to store additional information. From the ADI importer version 1.03 onwards, these are:

Code	Value saved
1-2	Marker codes defined by the Comment Map text file,
3-4	The channel number the comment relates to (1-32767) or 0 for a global or machine-generated comment. This is coded as code2 + code3*256. If a comment was truncated, we add 128 to code 4. To decode the channel use: `code2 + 256 * (code3 band 127)`.

Version 1.02 of the imported used code 1 as the channel number, code 2 as the section number and codes 3-4 encoded the comment number.

We simple-mindedly expect comments to be discovered in time order and we reject retrograde comments. If anyone finds this a problem contact us and we can try harder (by storing comments and sorting them into time order, which may slow import down a tad). Comments that occur at the same time are spaced out by a single clock tick; this can occur for machine generated comments at the start of the file and marking other sampling events.

Comment Map

You can provide a text file that will set the first two marker codes in a comment based on matching text in the text file. This is for the case where a system automatically generates comments, using the same or similar, text for similar situations and it would be useful to give similar comments the same code. You tell the importer which text file to use with the `cmd$` option, `cmap=<path to map file>`. You can set this as the `cmd$` option in the `FileConvert$()` script command, or by typing it into the Import Options dialog. The initial implementation expects the file to be encoded as ASCII or to hold UTF-8 encoded characters. Files generated in the Spike2 text editor will be suitable.

Each line of the text file is either a comment introduced by a semicolon, a directive introduced by a #, or defines one or two marker codes and a regular expression that must match the comment text to generate the code or codes. If there is no match, the first two codes are both set to 0. Currently there is only one directive, which is:

```
#IgnoreCase
```

This causes all regular expressions from this point onwards to use case insensitive searches.

The format of lines holding text to match is:

```
<Code>,<Regular expression>
```

`<Code>` is an integer value that encodes the first two marker codes as `code1 + 256 * code2`, so it lies in the range 0 to 65535.

<Regular expression> is text that follows the rules for an ECMAScript regular expression. If the regular expression matches any part of the comment text, the first two codes of the comment are set based on the <Code> field value. If you define multiple lines in the map, the lines are tested one at a time until one matches, or all have failed. The time taken to search the map is proportional to the number of comments times the number of lines in the map. You can save time by placing the most common items at the start of the list.

We use Regular expressions rather than a simple-minded left-matching search because it will let you match lines in a pretty general manner, and will allow you to treat lines that only differ due to embedded numbers as matching. If you only need to match the first few characters of a comment you can make a simple file, such as:

```
;Simple comment map that left-matches the comment text and ignores case in matching
#IgnoreCase
1,^ECG
3,^LVP
6,^EMG
```

The ^ means match the start of the comment. Note that code 1 will be applied to comments of the form "Ecg from Cat", "ECGraph" and so on. If you want to match the text anywhere in the comment, omit the ^. You can also match the end of the comment with the $ symbol. If you need to match all comments that contain "Dog" and end with "pressure" you could use:

```
4,"Dog.*pressure$
```

You are referred to the regular expression description for the InStrRE() script command for more information on matching and that has links to web sites that explain the many intricacies and pitfalls of regular expressions. Beware that it is possible for complex expressions to take a long time to search for matches, particularly if they contain more than 1 wildcard (.* is an example of such a search).

Script controlled import

If you use the FileConvert$() script command and do not provide the cmd$ argument, the default settings are to use cubic-spline interpolation and to separate each section with a 1 second gap. To set your own preferences, you would pass a string of the form: "fill=linear;gap=1.5" for linear interpolation and a gap of 1.5 seconds between sections. The fill and gap options are not case sensitive, but we suggest that you stick to lower case.

Binary importer

The binary importer will import blocks of data that consist of interleaved channels of waveform data of the same type with all channels at the same sample rate. The type can be 8, 16 or 32-bit signed or unsigned integers and 32-bit and 64-bit IEEE Real data. Data can be arranged in little endian (most common) and big-endian (e.g. very old Apple Macintosh data). Data is expected to start at a known offset into a file (often 0) and continue to the end. By interleaved we mean that the file holds all the channels data for the first point, then all for the second and so on. For example if CmPm means Channel n, point number m, then a file holding three channels (0, 1 and 2) would have consecutive data elements:

```
C0P0 C1P0 C2P0 C0P0 C1P1 C2P1 C0P2 C1P2 C2P2 C0P3 C1P3 C2P3 ...
```

The binary importer uses configuration files Bin_Last.icfx and Bin_Def.icfx. If the .icfx files are not found, old format .icf files are used, otherwise even older .cim files are used. These files are in the current user specific, application data folder.

The binary importer supports the following names in the cmd$ string for the FileConvert$() command. If any of these fields are not supplied no change is made to the existing settings, so if a configuration file is specified, the values it sets are used unless overridden.

name	value
Conf	Configuration file name. If this string is not empty then the provided file name including its full path will be used to load the configuration file. This keyword is always applied first, regardless of its position in the cmd$ string.
Bigend	Input file origin. Set to 0 if this is a little-endian file (which is almost always the case on modern systems) or to 1 if this is a big-endian file. The default is 0. In a little-endian file, lower significance bytes are at lower file offsets than higher significance bytes. If the 32-bit hexadecimal number 0x87654321 was stored at the start of a binary file, the first 4 bytes would be 0x21, 0x43, 0x65, 0x87 in a little-endian file and as 0x87, 0x65, 0x43, 0x21 in a big-endian file.

`Type`	Input data type. Set to 0 for 8-bit signed integers, 1 for 8-bit unsigned integers, 2 for 16-bit signed integers, 3 for 16-bit unsigned integers, 4 for 32-bit signed integers, 5 for single-precision floating point numbers and 6 for double-precision floating point numbers.
`Rate`	File sampling rate in Hz. If present it sets the overall file sampling rate.
`Chans`	Sets the number of imported waveform channels.
`Names`	Channel name. You provide a list of channel names for each imported channel separated by commas, for example: `Names=Resp,HR,BP`.
`Units`	Channel units. You provide a list of units for each imported channel separated by commas, for example `Units=bpm,bpm,mmHg`.
`Header`	If present, this value sets the offset to the start of the data, in bytes.

Resources are used in the following order:

1. A default state is set.
2. `Bin_Def.icfx` is used, if it exists.
3. If no `Bin_Def` files are found, `Bin_Last.icfx` is loaded.
4. If the `Conf` keyword is used, and the nominated file exists, it is loaded and overrides the current configuration.
5. Any other keywords are then applied.

An example

The following example imports a binary data file and passes in information as described in the table above. The configuration is read from the file: `c:/s/b.icfx`, then the number of channels to import and the channel names and units are set to override whatever was set in the configuration file. In most cases, you will set a configuration file and not need to override any of the values set in it.

```
var ret$;
var src$, dest$, flag%, err%, scom$;
src$  := "";  'Blank name so user is prompted for the source file
dest$ := "";  'Blank name so user is prompted for the destination file
flag% := 0;
scom$:="Conf=c:/s/b.icfx;Chans=2;Names=HR,BP;Units=bpm,mmHg";
ret$ := FileConvert$(src$, dest$, flag%, err%, scom$);
```

Bionic/Cybernetics/Ripple

These importers are almost identical and are described together.

The NEV file format has seen several revisions. The basic idea is that the format supports an array of up to 256 (expanded to 512) electrodes that are all sampled at the same rate (typically 30 kHz). From each electrode neural events (spike shapes) are detected, each of a fixed length (typically 52 points) with typically 15 samples before the trigger point. The spikes are typically stored using either 1 or 2 bytes of data per sample point. These channels are imported if they hold any data, that is empty channels are not imported. These channels are imported on channel 8 upwards. They have channel titles formatted as: `"Elec 101"` indicating the source of the data. The channels are calibrated in `"uV"`. We copy any class code as the first marker code, the second marker code is copied from a reserved byte in the file (normally 0), the other two codes are set to 0.

Some files also hold stimulus data that corresponds with the electrodes. This can potentially add 512 more data channels holding waveforms that look like spike data, but that were outputs rather than inputs. These channels are imported if they hold any data and follow on from the electrode channels. They have channel titles formatted as `"Stim 101"` indicating the electrode that matches the data. These channels are calibrated in `"V"`. We copy class codes for stimuli exactly as for Electrode data except that the third marker code is set to 1 to indicate a stimulus.

The format also allows for auxiliary channels that can be used to detect other events. There is a digital input of up to 16-bits that is imported as marker data on channel 1 (first marker code is bits 7-0, second is bits 15-8). There are 5 'Analog' event inputs that detect changes of state on rising, falling or both edges. These are

imported on channels 2-6 as events or level channels, as appropriate (but see below for channel 6). There is also a periodic event channel, imported on channel 7.

In addition to the NEV file which holds data packets representing spikes and events, there can be additional data files holding continuous data taken from the electrodes. These files have names with extensions `.NSx` for files holding integer data and `.NFx` for files holding 32-bit floating point data (Ripple). The `x` can be 1-9. Each such file holds a group of channel taken from the electrodes that are all sampled at the same rate; these are down-sampled versions of the electrode signals (though the down-sample ration can be 1). An early version of the format had the extension `.RND` and held Raw Neural Data sampled at the same rate as the electrodes. A more recent version of the format allows the continuous data to be turned on and off, generating data sections. These channels are imported with the channel titles and units defined in the data file.

File versions

Version 2 files use 32-bit time stamps, typically starting at 0 at the beginning of the recorded data with a resolution of 1/30000 of a second. Version 3 files use 64-bit time stamps, typically being the UTC time (count of nanoseconds since January 1 1970). Waveform channels are still sampled at multiples of 1/30000 of a second, which is a bit awkward as 30 kHz is not exactly representable in nanoseconds. We have seen files with retrograde time steps (which we attempt to survive by ignoring overlapped data) and files that are said to be sampled at say 1 kHz, but actually have gaps between samples of a few nanoseconds more or less than 1 millisecond. Spike2 expects files to have a constant sample interval. If we find that a cumulative error exceeds half a sample interval, we will put in a gap in the data or we will drop a point.

How we import the data

What we do depends on which file you select for import.

If you select a `*.nev` file, we attempt to reassemble all the data files from the recording (`*.nev` and any associated `*.nsx` and `*.nfx` files) into one Spike2 output file. We identify files by having the same file name and having time matching internal base sampling rates and a system time (stored in the file) that matches to within 10 seconds (though all files we have seen have exactly matching system times).

If you select any other files except a `*.nev` file, we import this file on its own. It will contain waveform data with all channels sampled at the same rate. Channels will be assigned in the order we find the channels in the input file, starting at channel 1.

There are differences depending on which importer you choose:

Bionics/Cybernetics

We expect to import files with the extensions `NEV`, `RND` and `NSx`. However, if you choose a `NEV` file and there is a matching `NFx` file, we will include it. Event channel 6 (if present) is always imported as a Level event channel.

Ripple

We import files with the extensions `NEV`, `NSx` and `NFx`. Event channel 6 (if present) is always imported as a Marker channel and uses the digital input data as marker codes (first code is bits 7-0 and the second code is bits 15-8).

Problems

This format is used by a range of companies and there is no guarantee that they have all implemented the format in exactly the same manner. If you have a problem importing a file or set of files, please contact us and describe the problem (and send us any screen output) before sending us files. We will almost certainly need example files, but we need the smallest file(s) that illustrate the problem (usually a few MB). DO NOT send us GB of unsolicited data!

BIOPAC importer

The BIOPAC importer is in two parts:

1. An importer that we wrote based on the BIOPAC Application Note 156 that can import a range of older files with big-endian or little-endian data. We do not have examples of all possible file formats to test, however, we have seen this import files with internal version numbers from 42 (AcqKnowledge 3.7) up to 84 (AcqKnowledge 4.1.1). We know that the current version does not import files with an internal version of 108 (4.2.0). If you have files beyond the range 42-84 that do import, we would be interested to know so we can update this information (the internal revision is included in the imported file comment). The latest changes (January 2020) allowed us to import a version 38 file, which appeared to have a damaged end.
2. An importer that reads more modern BIOPAC files using the BIOPAC AcqKnowledge Software API (ACKAPI). According to the BIOPAC web site this should read files from AcqKnowledge 4.1 - 4.4.2 and from Biopac Student Lab 4.0 - 4.1 and 3.7.0 - 3.7.2. We think that this reads little-endian files only (it does not read any of the big-endian example files we have).

Adding the ACKAPI support to Spike2

To import the more modern files, you must obtain the ACKAPI pack from BIOPAC. Once installed, you will need to copy the `acqfile.dll` from the BIOPAC distribution for the correct build of Spike2 (either 64-bit in the `x64` distribution folder or 32-bit in the `win32` distribution folder) into the Spike2 application folder. Once this DLL is in place, the importer will attempt to open the file you wish to import with the BIOPAC `acqfile.dll` first, and if this fails, it will fall back to the CED code based on the published BIOPAC Application Note.

If you are not sure which version of Spike2 you are running, look in the Spike2 Help menu About Spike2 box. If this says 64-bit, you are running a 64-bit version of Spike2 on a 64-bit system. If it says 32-bit, you are running a 32-bit version of Spike2, but you could be running on either type of system. If the `C:\Program Files (x86)` folder exists, you are running on a 64-bit version of Windows.

The easiest way to find where Spike2 is installed is to use the Help menu About Spike2... command to open the About Spike2 dialog and use the Copy button to place all the path information (including the *Installation path*) on the clipboard.

On more modern versions of Windows you will need Administrator privileges to copy files into the Spike2 folder. In Windows 10 you can do the following once you have installed the ACKPAC and assuming it has installed itself in the Program files folder:

1. Log into your system with Administrator privilege.
2. In the 'Cortana' window at the bottom left of the screen, type `Command Prompt`
3. A Best match should appear with "Command prompt; Desktop App". Right-click this and select "Run as Administrator".
4. A command prompt: "Administrator: Command Prompt" will appear. You can now use the command window copy command to move the correct DLL into Spike2. For example to copy the files on a 64-bit version of Windows to a 64-bit copy of Spike2:

```
copy "C:\Program Files (x86)\BIOPAC Systems, Inc\BIOPAC File API 3.3\x64\acqfile.dll" "C:\Program F
```

or to copy the files on a 64-bit version of Windows to a 32-bit copy of Spike2:

```
copy "C:\Program Files (x86)\BIOPAC Systems, Inc\BIOPAC File API 3.3\Win32\acqfile.dll" "C:\Progra
```

If you are on a 32-bit operating system, use:

```
copy "C:\Program Files\BIOPAC Systems, Inc\BIOPAC File API 3.3\Win32\acqfile.dll" "C:\Program Files
```

You may wish to make a batch file to do this (save the command text in a text file called `copyacq.bat`, for example) as you will need to repeat this when you update Spike2.

BrainVision importer

The BrainVision importer can read files that adher to the document: *Specification of BrainVision Core Data Format 1.0, Version 2.5*, last modified 12/December/2019. To import the data, you require the file `fileName.vhdr`, which describes the data and holds the names of the marker and data files, an optional `fileName.vmrk` file, that describes markers in the file and holds file Segmentation information, plus the data file, which has the file name extension `.eeg`, `.avg`, or `.seg`).

The data file holds either 16-bit integer data or 32-bit floating point data. The examples we have used to develop the importer have all held 16-bit integer data. They have also held the data in a single, continuous segment, so we cannot guarantee that the importer will work with multi-segment files.

Channels that are imported

The format describes multiple channels of data, all sampled at the same rate. The importer copies the channel titles, scaling and units from the files and generates a channel comment that includes any reference channel set and the electrode position as: (radius, theta, phi). For example, the channel comment might be:

```
Ref=FP1, (1,90,0)
```

If a channel has no reference, the `REF=Xxx` text is omitted.

In addition to the waveform channels, an extra TextMark channel is generated holding any markers set for the file. Markers have a Type (a text string) and a Description, plus a position, a duration and the channel they are associated with. We encode this in the TextMark channel by setting the text to:

```
Type: Description
```

And using the first marker code to set a type. If you take no action, type codes are assigned upwards from 0 based on the whatever type is set for the marker. You can pre-set known type codes (so particular marker types always get the same code), see Setting Type codes, below.

The Channel that is associated with the marker is encoded in the second and third marker codes. The second Marker code holds the lower 8 bits of the channel number and the third holds the upper 8 bits (so if you have fewer than 256 channels, the second marker code is set to the channel). These codes are set to 0 if the marker applies to all channels (often the case).

The duration of the marker, in sample points, is saved in the extra 32 data bits that are accessable with with `LastTime()`, `NextTime()`, `ChanData()` and `MarkEdit()`, using the optional `code%[4]` item.

Setting Type codes

You can pre-define a list of code types to force known types to have known type codes. To do this, create a text file with the pre-defined types, one per line. For example, using types we have seen in examples:

```
; Lines that start with a semi-colon are ignored
Comment
SyncStatus
Response
Stimulus
```

These 4 types will be given the codes 0, 1, 2 and 3. If more types are found, these will get additional codes. Currently the type codes are case sensitive, however we may change this if it is an issue.

You can either give this file the name `BrainVision.mrk` or you can use the `Cmd$` argument to set a file name (see, below). If you set a file name, and the name does not include any path information (i.e. none of the characters `/`, `\` or `:`), the importer will search for the file in the following places and use the first it finds:

1. The folder holding the `.vhdr` source file.
2. The current user application data folder:
   (`C:\Users\<User><.Domain>\AppData\Local\CED\<App>\`)
3. All users application data folder:
   (`C:\ProgramData\CED\<App>\`)
4. The current user My Documents application folder:
   (`C:\Users\<User><.Domain>\Documents\<App>\`)

5. The current user My Documents folder:
   (C:\Users\<User><.Domain>\Documents\)

where <User> is the user name, <.Domain> is the user domain (if part of one), and <App> is Spike11 or Signal8, or whatever application is being used to import data. The example paths are the ones that are used on my machine.

FileConvert$() command cmd$ argument

From the script language and the Import Options... dialog you can preset the file used to read marker type codes. The items are added before scanning the .vmrk file for names, so these script-added items get priority. The syntax is:

```
FileMark=<filename>
```

<filename> should include any file extension (.mrk is suggested), and the file should be ASCII or UTF-8 format. The file name can include a path, in which case only that path is searched for the file or it can have no path, in which case the paths listed above are searched.

DataPac Importer

DataPac files have one of three types of file header (A, B or C) stored in a .par, .pbr or .pcr file plus a .dat data file holding waveform data and an optional .eft file holding event start and stop times. The type of data stored in the .dat file is often 16-bit integer, but can be float (4 byte floating point) or double (8 byte floating point). We attempt to guess the data type based on the .dat file size.

If the channel is stored as integer data, the result is the integer data value, so you will likely need to calibrate it after reading it. If the data is stored as float or double data it will be in the original units, as stored. There are no units stored with the data, so it will appear with units of "V", which is the default.

Igor

We can scan Igor files with a variety of extensions (IGO, IGR, IBW, PXP) for Waveform data. Not all waveforms found in these files are useful. We import waves composed of signed 1, 2 and 4 byte integers and 32-bit and 64-bit floating point data. We do not attempt to imports unsigned integer data (though we could if it was really important to do so) now do we import complex waves (for example the results of spectral analysis).

We have had plausible results from IBW (Igor Binary Wave) and from PXP files (though PXP files can hold many types of information, such as Pie charts and the like, that do not lend themselves to waveform import).

MC_Rack

To use this importer to read .mcd files you need the MCStream.dll component. You can obtain this, in July 2022, by downloading the MC_Rack software from the multichannel systems web site: https://www.multichannelsystems.com/software/mc-rack and click on the MC_Rack Setup link. The latest version we have tested with is 4.6.2 dated 2015-02-12.

The importer uses the MCStream software to open and read the data file. If you get error -100 when you attempt to open a file, this means that the MCStream library has failed to open the file and there is likely nothing we can do to help you. If you get other error codes, or channels you think should be present are missing, we may be able to help you if you send us the (smallest) data file that suffers from the problem.

In addition to analog and electrode channels, the importer recognises Spike shape channels and events. From version [10.16] it also attempts to import digital channels, which hold up to 16 bits of digital information sampled as if it was a waveform channel. If such a channel is located, it is imported as if it was a waveform, and also as up to 16 level event channels. These channels are created by treating the signal as 16 separate digital channels and looking for changes. Each change generates an event. The digital channel is preserved (for backwards compatibility).

Motion Labs (C3D) importer

This importer should import most generic C3D files as described in *The C3D File Format User Guide* available from: https://www.c3d.org/pdf/c3dformat_ug.pdf

These files contain 3D information channels and ANALOG channels. The basic idea is that the 3D information is obtained from a video system and generates one (x, y, z, extra) value per video frame plus a list of other channels (the ANALOG group data). The ANALOG channels can be sampled at a multiple of the video frame rate. Files do not have to have any video data, so can be used to exchange general waveform data. All the ANALOG channels are sampled at the same rate.

Each 3D channel generates 3 channels in the imported file holding x, y and z co-ordinate values. We append _X, _Y and _Z to the end of the channel names to identify them.

Each ANALOG channel generates 1 channel.

We extract the channel names, descriptions, units and scales from the Parameter section of the file, if it is stored there. We scale the data following the rules in the Motion Lab Systems document: The C3D File Format User Guide.

The importer can read files holding integer or float data (including data written by old systems with PDP-11 style float format).

From Spike2 version [9.04], the importer reads the up to 18 events that can be stored in the file header.

We do not read any events that are stored in the Parameter section of the file (we have no example file holding this information). If you need to access this data and have example files holding it, contact us and we will look into scheduling the work to implement this.

NeuroScan importer

This importer imports both `.cnt` and `.eeg` extension files. We have example `.cnt` files from 1995 and 2007 and `.eeg` files from 1995.

The .cnt file importer

This imports the files as a list of waveform channels and one Marker channel. Prior to Spike2 10.14, there was one marker generated per waveform sample sweep, which was a lot of markers, most of which held unchanging data. The 4 attached marker codes for each Marker hold:

Code	Usage
0	Lower 8 bits of the 16-bit StimType held in each event.
1	The upper 8 bits of the StimType.
2	Keyboard input. Range 0-11 corresponding to function keys.
3	Bits 3-0 hold the coded response pad, bits 7-4 hold 0x0d for accept, 0x0c for reject.

At [10.14], to make the marker data a bit more manageable, we only log changes in the state of the events. The first marker establishes the initial state, then there is a new marker each time something changes. If the first marker is not at time 0, there is no information about states before this time.

The example files we have are Version 3.0, and these import OK (though we have nothing to compare the results with).

The .eeg importer

This imports channels of waveform data.

EDF importer

The EDF importer reads EDF, EDF+, BDF and BDF+ files. It does not implement logarithmic compression of data (but let us know if this is essential for your use and we will consider adding it). It is targeted at reading continuous recordings of data; it may work with discontinuous recordings, but we have no examples of this data type.

EDF data waveforms are treated as 16-bit data with scale and offset and are imported as Waveform data.

BDF data waveforms are 24-bit data and we import them as RealWave data.

BDF files can hold a Status channel that holds up to 16 channels of digital information. If any of these channel holds changes, we import the channel as a level event channel. Currently these channels start as low at the beginning of the recording and each change toggles the state. These digital signals have the same timing resolution as the waveform data.

The time resolution of the imported data file is set as close to 1 microsecond as possible while exactly matching the file sample rate.

Annotations in EDF+ and BDF+ files

Annotations are time stamps with optional attached data and duration. We ignore annotations at negative times (before the recording started) as this does not fit the Spike2 file model (these are permitted by the EDF+ standard but we have never seen one). Each annotation record in the raw data has the format:

```
[+-]Time{<21>Dur}<20>{Text1<20>}{Textn<20>}*<20><0>
```

`[+-]` One of + or -. We ignore annotations that start with -. This is the time of the annotation in seconds from the start of sampling.

`Time` The time of the annotation in seconds, written using 0-9 and . (though we will accept almost any reasonable number format).

`{...}` An optional block (`...` represents the contents).

`<n>` A byte with the value `n`, for example `<20>` is a byte with the decimal value 20.

`Dur` A duration, in seconds in the same format as `Time`.

`*` The preceding optional block is repeated 0 or more times

`Text1` The first annotation text. If this is not empty and does not exceed a set length (50 characters, was 30 before [10.18]), we look it up in a table of known annotations. If it is found, we add a code to the TextMark, if not found, we add this to the table of annotations and give it the next available code. Text in these blocks is encoded in UTF_8 (which includes ASCII) and must not include `<20>` or `<21>` as these are used as separators/duration markers.

`Textn` Further annotations at this time with the same restrictions as `Text1`. We do not look these up in the annotation table.

If we parse the annotation without error, and the annotation held text other than the time, we add a TextMark at the given time with the annotation code, or 0 if it is too long (assumed to be free text). The standard format of the text attached to the TextMark is:

```
Text1{|Textn}*{ [Dur]}
```

Most annotations we have seen in examples either have no text, or have just `Text1` and may have a duration. However, the format we have used should make it relatively easy to parse the text. If there is a duration, it is added to the end of the marker enclosed in square brackets and separated from the test of the text by a space. You can select an alternative format by using the `FileConvert$()` script command with the `Cmd$` argument.

If none of the annotations in an annotation channel hold any text or duration, we add a simple event channel with events at the annotation times.

Standard annotation codes

The EDF format definition suggest that there is a standard list of annotations. We have implemented this list and given them fixed codes 1 to 35:

Code	Annotation
0	Annotation text longer than the limit or no text
1	`Lights off`
2	`Lights on`
3	`Sleep stage W`
4	`Sleep stage 1`
5	`Sleep stage 2`
6	`Sleep stage 3`
7	`Sleep stage 4`
8	`Sleep stage R`
9	`Sleep stage ?`
10	`Movement time`
11	`Sleep stage N`
12	`Sleep stage N1`
13	`Sleep stage N2`
14	`Sleep stage N3`
15	`Apnea`
16	`Obstructive apnea`
17	`Central apnea`
18	`Mixed apnea`
19	`Hypopnea`
20	`Hypoventilation`
21	`Periodic breathing`
22	`CS breathing`
23	`RERA`
24	`Limb movement`
25	`PLMS`
26	`EEG arousal`
27	`Sinus Tachycardia`
28	`WC tachycardia`
29	`NC tachycardia`
30	`Bradycardia`
31	`Asystole`
32	`Atrial fibrillation`
33	`Bruxism`
34	`RBD`
35	`RMD`
36-...	Sequentially assigned codes for other annotations discovered in the file.

The codes we assign to other annotations are shared between all annotation channels (files can have multiple annotation channels and they may relate to similar signals). There is nothing significant in the codes, other than 1-35 are standard and 36 onwards are in the order of discovery. If you use the `FileConvert$()` script command with the `Cmd$` argument you can remove these predefined annotations and define your own.

Codes are not limited to 0-255, and can extend to much higher numbers if you have many different annotations. Markers in Spike2 have four marker codes. If the code exceeds 255, then marker code 0 is (code `band` 255) and marker code 1 is (code / 256). That is, marker code 0 is the lowest byte of the integer code, marker code 1 is the next byte (and so on if you have more than 65535 codes).

EDF Cmd$ keywords

When used from `FileConvert$()`, the following keywords are available through the `Cmd$` argument:

name	value
`TMSep`	This can be assigned the values 0 (the default, same as omitting the keyword) and 1. If you set 0, TextMark strings are generated to be easily human readable. Multiple annotations in one marker are separate by the vertical bar character | and any duration is added to the end of the marker as " [duration]". This format is can have problems if the annotation text already contains vertical bars or square braces. If you set 1, multiple annotations are separated by the ASCII character code `Chr$(20)` and any duration is preceded by `Chr$(21)`. For example, text might appear in one or other of these forms:

```
"Annotation 1|Annotation 2 [2.234]"
"Annotation 1"+Char$(20)+"Annotation 2"+Chr$(21)+"2.234"
```

You might prefer the second format if you were going to use a script to further process the imported data and you wanted to unambiguously separate annotations and durations (characters <20> and <21> cannot be used except as separators in a legal EDF file.

`NoPreDef` If this keyword is present, all the predefined Standard Annotation Codes are removed. The next added annotation will get code 1 unless `AnBase` is used.

`AnBase` This sets the first code to use for added annotations. Each time an annotation is added by the `AnList` keyword (see below), or by encountering an unknown annotation that is not more than 30 characters in length, the annotation gets the current `AnBase` value, then the value is incremented for the next item. Put another way, the annotations have consecutive codes from `AnBase`. If you do not use `NoPreDef`, the default `AnBase` value is 36, if you do use `NoPreDef`, it is 1.

`AnList` You can pre-set a list of annotations. These are assigned consecutive codes, starting at the current `AnBase` value. To be added, the annotation must not be more than 30 characters long. For example:

```
Cmd$:="AnBase=1;AnList=One,Two,Three,\"this, or that\",last";
```

Annotations `One`, `Two` and `Three` have codes 1, 2 and 3, `this, or that` has code 4 (note use of quotes to include a comma which would otherwise be a separator). `last` has code 5.

`AnCode` Use this to set a list of annotations with defined codes. There are pairs of values assigned to the keyword. The first of each pair is a code, the second is the annotation. For example:

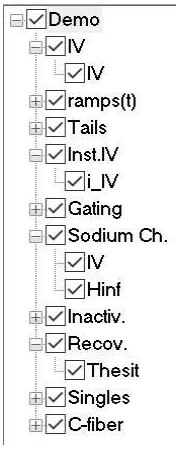
```
Cmd$:="AnCode=49,One,50,Two,57,Nine";
```

This assigns codes 49, 50 and 57 to `One`, `Two` and `Nine`. These are the ASCII codes for "1", "2" and "9".

Heka importer

Heka files have several versions. The earlier ones had a `.dat` file and additional files with the same base name and different extensions that included `.pgf` (holding the Stimulus Templates) and `.pul` (holding the Acquisition Parameters). More recent formats have a single `.dat` file that includes all the information. If you import an older format `.dat` file and the `.pgf` or `.pls` files are missing, the importer will reject the file with error -100. The import progress window (usually under the window with the error message) will mention the first file that it failed to find.

Heka files are arranged in a tree. You can visualise the structure in the user dialog that is displayed when you use the interactive File menu Import... command:



The tree from a typical file (the Heka demonstration file from their web site) is displayed on the left with some of the branches open and some closed. Any file you open will have a similar structure, but the names and number of the branches will vary, depending on the file contents.

The points where the branches meet the Root or the parent branch are called *Nodes*. Nodes of the tree are indicated by a small box holding either a + symbol (click it to open the node and display the contents) or a – symbol (click it to fold up the node and hide the contents).

All nodes of the tree have a check-box. If you uncheck a node, it, and all the data that belongs to that node will not be imported. If you unselect the *Root* of the tree (*Demo*), nothing will be imported and you will get an error if you try to import.

Below the Root are the *Groups*. A Group is composed of (presumably) related experiments. However, there is no requirement that the members of a Group are compatible (in the sense that they can be imported into the same Spike2 data file). In this case, the names of the Groups are: IV, ramps(t), Tails, Inst.IV, Gating, Sodium Ch., Inactiv., Singles, C-fiber. There is no requirement for the names of the groups to be different, and in many cases the names will repeat. If you uncheck a group, it will not be imported.

Within each Group are *Series*. A Series holds one or more *Sweeps* of data, and all the sweeps in a series are compatible (in the Spike2 sense of being readable into the same file). The name of each series indicates the protocol used to generate the data. If you uncheck a series, the data for the series will not be imported. In the example, there are visible series with the names: IV, i_IV, Hinf and Thesit.

By default, all data in the tree is selected for import (unless you use the cmd$ argument to pre-select specific items). When importing under script control, the dialog is not displayed; selections are made based on the cmd$ argument.

However, having items selected does not guarantee that they are imported. When you close the dialog to import the data, the importer searches the tree to find the first Series that is selected for import. This generates the template that is used to create an output file. The importer then scans the tree of selected items and unselects any Series that is not compatible with the template. Currently, being compatible means:

1. The Series has the same number of data channels as the template.
2. The channels have the same type as the template.
3. The channels have the same sample rate as the channels in the template.

A series does not have to have the same number of data points as the template (and some series may have a channel with no data points).

Gap-free

Normally, every sweep of data is imported as a separate chunk of data and the Marker channel is used to indicate the start and end of each sweep. If you check the Gap-free check box, the data is imported as if it was sampled in one continuous section with no indication of where each internal sweep started or ended. This is not usually what you want.

Group and Series TextMark channels

In addition to the waveform data channels found in the file, we add two additional TextMark channels. These are added to the file each time a sweep is imported when the Group or Series in the original data changes. Data is added even if the Group or Series has the same name as the previous one. The importer also keeps track of the Group and Series name and allocates them codes, starting at 1 based on the order in which they are discovered in the original file (without any regard to which check boxes are clicked). These codes are given to the TextMark items to allow you to identify them. If you are running the importer from a script, you can pre-set the identifiers using the GROUP= and SERIES= values in the FileConvert$() cmd$ argument.

FileConvert$() command cmd$ argument

From the script language and the Import Options... dialog you can preset the codes associated with group or series names. The items are added before scanning the data file for names, so these script-added items get codes starting from 1. For example setting FileConvert$() command cmd$ argument to "GROUP=Gr1,Gr2,Gr3;SERIES=S1,S2,S3,Series4" will pre-set three group names and 4 series names (note that group and series name matching is case insensitive and uses ECMAScript regular expressions).

If you do not set group names, all groups are selected. If you do not set series names, all series are selected (but only if its group is selected). If you set neither a GROUP nor a SERIES then everything is selected. The first selected group/series combination sets the type of data that is to be imported and if any of the subsequent selections are incompatible, they are rejected.

A match means that the regular expressions matches the group or series name. The examples given above for GROUP will match any group name that contains any of Gr1, Gr2 or Gr3. To force an exact match you would use "GROUP=^Gr1$,^Gr2$,^Gr3$" or the more succinct "GROUP=^Gr[123]$". See the regular expressions description to see all the (very many) possibilities.

If your matching text must include commas (which are separators), you can enclose the text in double quotes. So, to match 10,11,12 in a series name and also exactly match S1 you could set (from a script):

```
cmd$ := "SERIES=\"10,11,12\",^S1$";
```

The \" combination is to escape the double quote within the text string (otherwise they would terminate the string and cause a syntax error). If you type this text into the Import Options... dialog you would use: SERIES="10,11,12",^S1$ as there is no need to escape the double quotes.

If your choice of GROUP and SERIES ends up selecting nothing, import will fail with an error.

New features

Before version [9.03], the importer would import one sweep of data from the first selected item in the tree. It was also limited to files with a maximum of 20 Groups and 100 Series within a group. It did not import the names of the Groups and Series as TextMark data.

Before version [10.20] the importer did not know about floating point data and imported it as rubbish. Also, if a required file was missing only the `.dat` file was listed, so it was not clear what had gone wrong.

Before version [10.21] script import always failed. The use of regular expressions was added at this revision.

Ponemah importer

This importer reads all waveform data as RealWave. It currently does not import Marker data as this is not supported by the DSI import software we have access to. The import code we use as the basis of the import filter is taken from the DSI web site. The last time we checked it (November 2021), it was at:

https://support.datasci.com/hc/en-us/articles/360052428493-Reading-Ponemah-6-x-data-into-Matlab

The data file is imported with a time resolution of 0.1 microseconds (which is the resolution used to record them). If you have any problems using this importer, see the Problems section at the end of this page for possible solutions.

Channel information we extract

We generate channels of data from the Ponemah file in the order that the channels are found. Channels are ordered by `<Subject>`, then by `<ChanId>`, so all the data for one subject is grouped together in consecutive Spike2 data channels.

The channel Title is copied as supplied, as are the channel units. The channels are read as RealWave data and the scale and offset are set so that if the channel were converted to a Waveform channel (16-bit integer), the range of data in the channel would fit within the 16-bit range. If the data range includes 0.0, the offset value is set to 0, otherwise it is set to give the most accurate representation of the data possible.

The channel comment is built from the following Subject fields, separated by a comma:

```
<Name> + ", " + <Species> + ", " + <Gender> + ", " + <SubjectId> + ", " + <ChanId>;
```

for example (taken from one of the Ponemah example files):

```
Qax6998, Dog, , 1, 3
```

In this case, the `<Gender>` field was not set.

FileConvert$() command cmd$ argument

From the script language and the Import Options... dialog you can choose to filter the data you import by subject and channel, and you can limit the time range of imported data. This can be useful with huge files (we have successfully imported an 80 GB file spanning 2 months of data with 100+ channels, but it took 2 hours)! The filtering is done by defining Name=Value pairs as: `Name1=Value1;Name2=Value2;...;NameN=ValueN`

The first group of filters is used to match data by subject and channel number within the Ponemah file. The names you are matching are those that generate the channel comment, channel title and channel units, described above. If you have a huge file with lots of channels, use the time range filter to import a few seconds of data at the start of the file and look at the channel comments to work out the text to match.

Matching is by a case insensitive regular expression. If you do not supply an expression, there is no filtering. If you supply a matching expression, the channel is accepted if the field matches anything. For example, if you want to match the subject name and you supply `Rat` as the match, this will match subjects called `Rat1`, `Rat1023` and `Fraternal` as these all contain a case insensitive `rat`. To match the entire name use the `^` and `$` anchors to mark the start and end, so `^Rat$` will match only `rat` and not `rat1` or `brat`.

You can also match a list of alternatives, for example to match `cat` or `dog` you could use `^(cat|dog)$`. You can find the full syntax of the accepted regular expressions at this Microsoft site: https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference.

If you supply multiple matching keywords, all must match to accept the channel. Remember that keyword and values are all case insensitive.

Name	Value
Subject	The name of the subject. Only channels for the nominated subject are imported. If the subject is not found, no data is imported. If no subject is supplied, all subjects are imported. For example: `Subject=Rat1`
Species	The species to match. For example: `Species=^Dog$`
Gender	We have never seen an example of a gender supplied in a file. You could try `^m` and `^f` (something starting with `m` for male and `f` for female).
SubjId	The subject ID, which is a number. Remember that `1` will match any number with a 1 in it, so to match a particular number (such as 1) set `SubjId=^1$` as the configuration.
Chan	You can use this to match a particular <ChanId> value (see above). For example, to match Ponemah channel 12, set `Chan=^12$`, to match channels 23 or 24 or 30 set `Chan=^(23\|24\|30)$` in the configuration.
Title	Match the channel title, for example, to select all channels that include EEG in the title, set `Title=EEG` in the configuration.
Units	You can filter channels based on their channel units. For example, to match channels with units of mV set `Units=^mV$` in the configuration.

You can also filter data by time range. The channel filters run first and these reduce the file to a list of acceptable channels. The very first item in the file after filtering by channels determines the start time of the file, and will be at time 0 seconds in the Spike2 file if no time range is set. If you switch to time of day display mode, this will still show the correct time in the original time zone. You can use the tOff and tLen options to reduce the imported time range. If you do this, the earliest imported data will still display at 0 seconds. In Time of Day axis mode, the times will still match the original Ponemah file.

Name	Value
tOff	The time offset in seconds from the first data in the file to import. Only data that passes the subject/channel filter is considered when deciding what is the first time in the file. If this is omitted, the import starts from the first data in the file (equivalent to setting tOff=0.0. The first data imported is at time 0.0 seconds in the imported file. The Time of Day is adjusted to compensate for the offset. The time offset is set to an accuracy of 0.1 microseconds (0.0000001 seconds).
tLen	The length of the data to import, in seconds. If omitted or 0.0, all the data from tOff to the end is imported. The data length can be set to an accuracy of 0.1 microseconds.

To import all data channels for the Subject called Rat3 from 1 hour into sampling for 10 minutes of data you could use:

```
Subject=^Rat3$;tOff=3600;tLen=600
```

The length of time required to import filtered data seems to depend on the duration of the data and number of channels (as you would expect) and also, to some extent, on how much data must be skipped to located the imported data.

How to handle NaN (Not a Number) problems

Data transferred via a radio link can suffer from drop outs caused by poor signal strength or animal movements. Such signals are imported as NaN (Not a Number) values. These can be a nuisance as they cause all mathematical operations that involve them to also be marked as NaN. To work around this, you can add the Skip NaN channel process. This replaces the bad values with missing data. You can further repair the missing data with the Fill Gaps channel process to linearly interpolate over the gaps created by Skip NaN. You can also consider using the Linear Predict option to permanently replace NaN values with 'likely' data values.

Problems opening the importer or error -100 when using it

If you have previously imported Ponemah 6 files with your current installation, error -100 usually means that the file you have selected ins invalid. However, if you have never managed to import any Ponemah 6 files this may mean that the Ponemah importer is not installed, or is damaged.

For the importer to work correctly, the Spike2 installation folder must have within it a folder called Import. Within this folder there must be a file ponemah.dll and a folder called ponemah which must hold the following files:

log4net.dll, Ponemah.Logger.dll, Ponemah.WaveformFile.dll, Ponemah6xExtractor.dll, SQLite.interop.dll, System.Data.SQLite.dll

If any of these files are missing you may get error -100 when you attempt to import, or you will not be offered Ponemah as an import filter. These files should all be installed as part of the Spike2 installation. The SQLite.interop.dll file has two versions, one for 64-bit and one for 32-bit Spike2 installations, so be careful if you attempt to 'fix' installations by copying files.

Another possible reason for the import to fail is that the file Sonview.exe.config is missing (should be in the `Spike11` folder in the same place as the `Sonview.exe` file). This is a text file with the contents:

```
<?xml version = "1.0" ?>
<configuration>
       <startup>
              <supportedRuntime version="v4.0" />
       </startup>
       <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
        <probing privatePath="import/ponemah"/>
      </assemblyBinding>
   </runtime>
</configuration>
```

This importer requires that a minimum of .Net 4.0 is installed. If you are running Ponemah software on your computer you are probably OK, or if you are running an operating system more modern than Windows 7 (as surely you are). Otherwise you will need to Update your version of .Net from the Microsoft Update Centre. Search for `.Net Update` on the web. Ideally you want .Net version 4.7.2 or later.

Percept PC

This importer reads files written in JSON format by the Medtronic™ Percept PC system. The data imported is read from the BrainSenseTimeDomain and BrainSenseLfp sections of the file. The importer was written so that it imports a particular example file (with two waveform channels and associated LFP data), and it should import similar files, or files with additional similar pairs of channels and LFP data. From each pair of channels and the associated LFP data, the importer generates two waveform channels, one RealMark LFP channel and one RealMark channel holding the Therapy Snapshot data (one set of values per data block).

Channel names

The channel names used in the files are of the form: `NUMBER1_NUMBER2_SIDE`, for example `ZERO_THREE_RIGHT`. When we recognise this format, we shorten it to: `03 Right` so that it fits within the expected channel title space. We shorten the LFP data to `"03"` and the therapy data to `"03 Thr"`. We preserve the original channel names in the channel comment.

Relative timing between channels

In theory, the channels should all be timed to the nearest 50 ms. However, there appears to be a bug in the BrainSenseLfp:LfpData:TicksInMs field that causes a discontinuity in the millisecond timing every 3276.8 seconds into the data session. Although we could make heroic efforts to work around this, we were told by the group requesting this importer that an accuracy of "to the nearest second" was good enough, so we have not done this. In any case, it would not be possible to always be certain that a workaround was correct. Note that the timing within a channel should be exact (unless data was lost in a transfer, which we assume does not occur).

What we import

We attempt to import the time domain waveform channels and the associated LFP and Therapy Snapshot data. We do not attempt to import patient and clinical data, setup or calibration data. The waveform channels in the file (two in the example file we were given, called `ZERO_TWO_LEFT` and `ZERO_TWO_RIGHT`) are imported as RealWave channels (with the titles `02 Left` and `02 Right`). Each LFP trace holds the LFP data from a left/right pair of channels (with channel title `02`) plus 4 additional values, being the LPF and current in mA

values for the Left and Right channels. The LFP trace also holds one set of `TherapySnapshot` data per data block.

The data we read from the JSON file is as follows:

BrainSenseTimeDomain

This section holds blocks of data, each holding waveform data for a single channel. The data blocks are in time order, so if a channel has multiple blocks, the later blocks are at a later time that previous blocks and do not overlap with them. Each channel identified in this section generates a RealWave data channel.

The fields we use are:

Channel

This field holds a channel number, of the format `"ZERO_TWO_LEFT"` or `"ZERO_TWO_RIGHT"`. We use this field to form the channel title, but we shorten the name to 02 Left and 02 Right to we more compatible with Spike2. It is likely that this data will always occur in channel pairs, but we can import any channels. We use the channel name to identify the number of different channels.

FirtsPacketDataTime

This field holds the time and data of the first data packet of this block of data to an accuracy of 1 seconds: `"2023-01-18T08:57:26.000Z"`. There is also a field that contains the times of arrival of each data packet in milliseconds (this time seems to be relative to the start of the session). We have not made use of the millisecond timer to align packets, though this would be possible.

SampleRateInHz

The sample rate of the channel in Hz: `250`. For data import to function the channel rate must not change within a file. If the rate does change, the importer will reject blocks with a rate that differs from the first rate seen for a channel. Different channels can have different rates (likely not an issue with this file type).

TimeDomainData[...]

This field is an array of data values, representing the channel data in uV sampled at the SampleRateInHz rate. There are likely to be many tens of thousands of values in the array.

Fields we do not use are:

Pass

Blank in our example. We do not know what this is.

GlobalSequences, GlobalPacketSizes, TicksInMses

These fields hold text with lists of integer numbers separated by commas. Each list is the same length and hold the packet sequence number (modulo 256), the data items in each packet, and the packet time in milliseconds (always a multiple of 250 ms in the example we have). The packet sequence numbers have gaps in the numbering `("0,1,2,3,5,6,7,8,10...")` where a packet with LFP data is inserted.

Gain

This is an integer field that is 225 in our example file (but we have seen values of 224 and 220 in other sections used for setup and calibration).

BrainSenseLfp

This section contains data relating to pairs of channels. Each data item has a sequence number, and then `mA` and `LFP` data for Left and Right. Each pair of channels we identify generates one RealMark channel with 4 attached data values. The channel title is set to `"nm"` where `n` and `m` are the common numbers from the Channel field, for example `"02"`. The attached data values and indices we generate in the RealMark data are:

#	Name	Contents
0	`LFPr`	The right channel LFP value (arbitrary units).
1	`Ir`	The right channel current, in mA.
2	`LFPl`	The left channel LFP value (arbitrary units).
3	`Il`	The left channel current, in mA.

The fields we use to collect this data are:

Channel
This field holds a pair of source channel numbers, of the format `"ZERO_TWO_LEFT,ZERO_TWO_RIGHT"`. We use this field to form the channel title, but we shorten the name to 02 as the data relates to both the left and right channels. It is likely that this data will always occur in channel pairs. We use the channel name to identify the number of different channels.

FirstPacketDataTime
This field holds the time and data of the first data packet of this block of data to an accuracy of 1 seconds: `"2023-01-18T08:57:27.000Z"`.

SampleRateInHz
The sample rate of the LPF data, in Hz. This has the value 2 in the example file we were given. We assume that the data is a continuous waveform (no gaps) starting at the FirstPacketDataTime at this sample rate. If the TicksInMs field is repaired (see below), we could use that to give more accurate timings.

LfpData[...]
This is an array of data blocks from which we extract the values that we return for each RealMark data point. Each block has the following fields:

`Seq` The block sequence number. In our example file, these values were the ones missing from the *GlobalSequences* values mentioned above.

`TicksInMs` This should be the data block session time, in milliseconds to an accuracy of 50 ms. Unfortunately, this resets back to 0 every 3276.8 seconds, which makes it a lot less useful. If this is ever fixed, or we figure out a way of using this reliably, we may make use of this as the item time.

`StatusBytes` A string holding `"00 00 00 00"` that we guess are 4 byte values expressed in hexadecimal that we read and store as the 4 marker codes associated with the data item in each RealMark item.

`Right` The right hand channel data values for `LFP` and `mA` data.

`Left` The left hand channel data values for `LFP` and `mA` data.

Note that the LPF values are in arbitrary units. There are also occasional huge values (4,294,967,295: the maximum possible 32-bit unsigned integer) which we presume to mean invalid data. Valid LPF values are positive, so we have chosen to replace these invalid data markers with the value -1, which makes the channel data much easier to deal with; at the very least it allows you to Optimise the display without causing all the LFP data to become invisible.

TherapySnapshot
This part of the BrainSenseLfp section holds the settings used for the associated data block. We generate a separate data channel called, for example `"02 Thr"`, that holds RealMark data. There is one data point per LFP data block and each data point has 22 attached items that are read from the TherapySnapshot. The data stored in each index is given in the table. Where there is a range of indices, 2-3 for example, the lower has the left channel data, the higher has the right channel data.

#	Data stored in the index
0	The High-pass filter setting in Hz
1	Sensing blanking duration in microseconds
2-3	Pulse width in microseconds.
4-5	Rate in Hz.
6-7	Lower limit in mA
8-9	Upper limit in mA
10-11	Frequency in Hz
12-13	Frequency index
14-15	Upper LFP threshold
16-17	Lower LFP threshold
18-19	Averaging duration in milliseconds

20-21 Detection blanking duration in milliseconds

We expect that this data will be useful to users who write scripts to analyse data and who wish to track how changes to these parameters affect it.

The example data file we worked with had 7 data capture episodes spread over 75 minutes with data channels `O2 Left` and `O2 Right` sampled at 250 Hz, the LFP data for these 7 episodes was sampled at 0.5 second intervals and we generated 7 Therapy Snapshot data samples, one at the start of each of the 7 episodes, timed to match the LFP data. In terms of Spike2 channel numbers we generate 4 channels:

Channel	Contents
1	The LFP data points with 4 attached values per point at 0.5 second intervals.
2	The waveform data from channel `ZERO_TWO_LEFT` sampled at 250 Hz.
3	The waveform data from channel `ZERO_TWO_RIGHT` sampled at 250 Hz.
4	The TheraphySnapshot data with 22 attached values, one data point per data capture episode.

Example script to list therapy data

The following script reads and displays the therapy data (assumed to be held in channel 4):

```
var code%[4];           'Item codes (unused for this channel)
const chThr% := 4;      'The channel holding therapy data
var thr[22];            'Space for the data values
var t:= -1;             'start time of the search
repeat
    t := NextTime(chThr%, t, code%, thr[]); ' Get next set of data
    if (t >= 0) then    'If we got data, display it
        PrintLog("%6.1f %g\n", t, thr[])
    endif;
until t < 0;            'until no more data found
```

The output of this, with the example data file, was:

```
   1.0 1,2000,60,60,125,125,0.6,1.4,1.8,2.4,18.55,17.57,19,18,30,30,20,20,3000,3000,2000,2000
 469.0 1,2000,60,60,125,125,0.6,1.4,1.8,2.4,18.55,17.57,19,18,30,30,20,20,3000,3000,2000,2000
1106.0 1,2000,60,60,125,125,0.6,0,1.8,2.4,18.55,17.57,19,18,30,30,20,20,3000,3000,2000,2000
1661.0 1,2000,60,60,125,125,0,0,1.8,2.4,18.55,17.57,19,18,30,30,20,20,3000,3000,2000,2000
2638.0 1,1470,60,60,145,145,0.6,1.4,2.4,2.8,18.55,17.57,19,18,30,30,20,20,3000,3000,2000,2000
3221.0 1,1470,60,60,145,145,0.6,0,2.4,2.8,18.55,17.57,19,18,30,30,20,20,3000,3000,2000,2000
3734.0 1,1470,60,60,145,145,0,0,2.4,2.8,18.55,17.57,19,18,30,30,20,20,3000,3000,2000,2000
```

The first column is the time offset, in seconds, from the start of the file (which begins with the first recorded data block). The following columns are the 22 data values.

Text importer

When a user imports a text file interactively, the last user configuration is saved as an XML format file called `Ascii_Last.icfx` in the current users application data path. If you rename this (in the same place) to `Ascii_Def.icfx`, then this will always be used to set the initial configuration unless you use the `Conf` option (described below). You can save configuration files with your choice of name from the text import dialog that appears when you use the text importer interactively.

Configuration files are applied after doing an initial scan of the file. They can only make changes that are compatible with the data that is discovered by the initial scan; they cannot create additional channels.

The text importer has its own interactive Help file with more information about supported text file formats. If your file holds a header followed by a rectangular table of columns, each column representing equally spaced data (waveforms), then you should have little difficulty importing it. You will need to refer to the specific text importer help to cope with columns containing event times or TextMark or RealMark data.

The most common channel type is a waveform, which is recognised as a column of data containing numbers. If the first column holds numbers that increase with a fixed spacing, this is assumed to be a timebase. In addition

to numbers, the text 'NaN' is recognised as a number and evaluates as 0.0. Otherwise, any non-numeric text in a column causes the column to be treated as text.

The text importer supports the following names in the `cmd$` string for the `FileConvert$()` command.

name	value
`Conf`	Configuration file name (including the `.icfx` extension). If this string is not empty then the provided file name including its full path will be used to load the configuration file. This keyword is always applied first, regardless of its position in the `cmd$` string.
`Find`	In cases where the text holds data blocks of different types you can change how the initial scan chooses the block that is most typical of the data, which determines the number of columns. You can set the values: 0 This is the default. The block with the largest number of numeric items (columns-first numeric column)*lines. 1 The block with the largest number of items (columns * lines). 2 The block with the most lines. 3 The block with the most columns.

What to do if the importer does not recognise your file

If your text data does not meet the rules for import by the importer, you could consider writing a script to do the task. This is usually not very difficult (as long as you can use the script language); the complications of the general text importer come from the fact that it has to cope with a wide variety of data formats.

Files you want to import will usually be in a fixed format that you know well. You will usually have multiple columns of data and possibly a file header. The commands you will need to use will include:

`FileOpen`	To open input files of type 8 (external text files with no associated window).
`FileNew`	To create a new output data file of type 7.
`FileSaveAs`	To save the resulting data file; use type -1.
`ReadSetup`	To configure the `Read()` command so you can easily split the lines in the file into columns
`Read`	To collect a single line of input and convert it into variables. If you need to inspect the lines first (to detect the line type), use `Read()` to collect an entire line into a string, followed by `ReadStr()` to extract columns of data from suitable lines.
`ReadStr`	To convert a string holding a known format into variables.
`ChanNew`	Create a channel in a data file, usually for writing to with `ChanWriteWave()`. If you are creating a non-waveform channel, use `MemChan()`.
`ChanWriteWave`	Write data to a Waveform or RealWave channel created with `ChanNew()`. You can also use memory channels to create waveform data.
`MemChan`	Create a memory channel of any type to write to with `MemSetItem()`.
`MemSetItem`	Add a data item to an existing memory channel.
`MemSave`	To save a memory channel to a disk channel to make it permanent.

The advantage of writing your own importer is that it allows you to extract all the information from the file. If you use our importer, you are restricted to importing files that match our expectations and we only allow you to read channel titles and units from any file header, and these only when they match the data columns.

As long as the files are not too large, copying read data into memory channels and saving the result as disk channels is usually the easiest approach as this leaves Spike2 to sort out any effects due to gaps or overlaps in waveform data.

XDF (eXtensible Data Format)

Also known as LabStreamingLayer. A mixture of XML meta-data and binary wave data. Does not import 64-bit integer ADC data. You can find a description of this format here. The importer can handle the example files on this site plus some additional files from a user.

This data format makes provision for multiple data streams from different devices which may have time bases that run at slightly different speeds, which can become significant for long files. The current importer does NOT attempt to compensate for this. It would be possible to do this should this become a significant problem.

The format stores data as streams. Each stream holds one or more channels of data of the same type. Each stream is either regularly sampled (as interleaved waveform data) or holds time stamped data. We import time stamped data either as RealMark data (where channels become items in each RealMark), or as TextMark data (we expect only a single channel per stream).

Xltek Neurowork importer

This importer expects to find all the files for one import in a folder. All the files (and the enclosing folder) have a common (and somewhat long) name. This is usually of the form:

```
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

where the `xxxx` are hexadecimal digits. There can be problems if the full path name to a file reaches 260 characters, which is not that difficult to achieve when the folder name and file name account for more than 100 characters.

A typical set of files in a folder could be:

```
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.eeg
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.ent
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.epo
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.erd
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.etc
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.snc
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.stc
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx_001.erd
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx_001.etc
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx_002.erd
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx_002.etc
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx_003.erd
User-defined_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx_003.etc
```

The `.erd` and `.etc` pairs hold the raw data and there can be many pairs of these files. The importer looks for the `.stc` file as the key to the import process as this file holds the information to tie the experiment together. For our importer to work, the file names must all be correct and match the `.stc` file name (but we do not care if the folder name is different). We import all the raw waveforms, an event channel of Trigger data, and a channel of TextMark data indicating the original raw data file starts.

In addition to importing data from the `.erd`/`.etc` files, we also parse the `.ent` files, which hold descriptions of montages and user comments and automated comments. These comments are imported as a TextMark channel. Each comment in this file has a type (Comment/Data/Type), and we use the type to set the first marker code of the TextMark. We observe that user-entered comments seem to be given a type of "Custom", and these are given type code 00. Any other types seen are given sequential type codes from 00, in the order that we encounter them in the file. We can assign additional fixed codes to other types, if required. There is an additional field (Comment/Origin) that we have observed set to "Acquisition" or "Review". If this is "Review", we set the second type code to 01, otherwise it is 00.

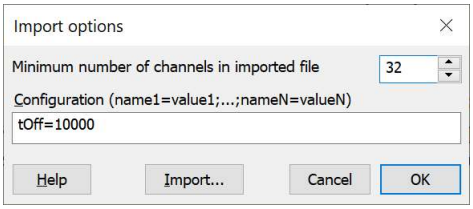
Big-endian and Little-endian

These terms refer to how the bytes that make up data values that use more than one byte of storage are arranged when stored into memory and when written to disk. Consider the 32-bit integer value (4 8-bit bytes of data) written in hexadecimal as 0x03020100. Traditionally, data is stored in memory with the bytes in the order of their significance. If they are written with the least significant byte first, the data is said to be little-endian, if it is written with the most significant first, it is said to be big endian.

Storage byte index	0	1	2	3
Little-endian	0x00	0x01	0x02	0x03
Big-endian	0x03	0x02	0x01	0x01

As long as you write and the read data with the same endianism, there is no problem. Data on Windows and Intel-based Macintosh computers is stored in little-endian format, but data from other system may be big-endian. In particular, the older Motorola/PowerPC based Macintosh computers used big-endian formats. You can read more about endianism here.

Import options dialog



This dialog (File menu->Import->Options...) sets the minimum number of channels in each file imported by the Import command and optionally, the configuration string that is used by some importers for additional control over the import process.

Channels

Extra channels are useful if you intend to further process the files and need channel working space. The minimum number of channels in a Spike2 data file has always been 32. Spike2 version 9 onwards supports files with up to 2000 channels. Setting more channels that you really need is wasteful of disk and memory space. System memory or resources may limit the number of channels.

When you import a file with `n` source channels, Spike2 creates an output file with `n*3/2` channels or the number of channels you set in this dialog, whichever is the larger. The maximum number of channels is currently limited to 2000. Do not set huge numbers of channels unless you really need them as each channel uses up file header space.

The number you set here is stored in the Preferences section of the Registry under the title: "`Minimum channels in imported file`". See the `Profile()` script command for details.

Configuration

This field (added at [10.13]) sets importer-specific options in the same way as the `cmd$` argument of the `FileConvert$()` script command. The string is of the form: `name1=value1;name2=value2;name3=value3` where the names are case insensitive. It is entirely up to the importers what names they recognise and what they do if you type rubbish; most importers will ignore anything they do not recognise. For more information about specific importers click here and then find the importer in the table. If the importer supports a command line, the **`cmd$`** column contains `Yes`. Click the `Yes` for details of supported keywords.

If in doubt, leave this field empty. The text entered here has no effect on the `FileConvert$()` script command.

The text you enter here isstored in the Preferences section of the Registry under the title: "`Import configuration`". See the `Profile()` script command for details.

Import...

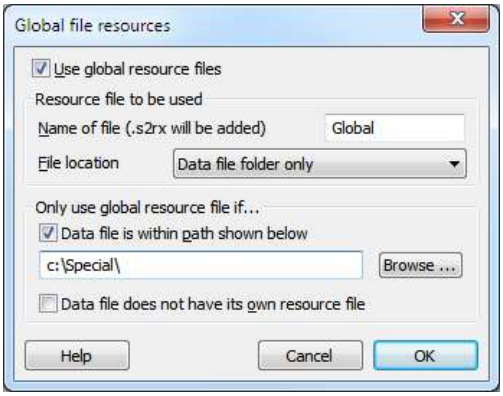
Click this button to close the dialog and run the importer with these options.

Global Resources

If the current view is a time, result or XY view, this File menu command appears as Resource Files->Global Resources..., otherwise it appears as Resource Files....

Normally, each Spike2 time, result and XY data file has an associated resource file with the same name and `.s2rx` or `.s2r` extension. These per-file resources remember the screen layout, cursor positions, active cursor parameters and other settings. They allow each file to open with exactly the same screen appearance it had when it closed at the cost of an extra file on disk.

However, per-file resources do not let you use the same display and cursor settings for a sequence of files. If you enable global resource files, you can use a single resource file (or one resource per folder) to control the screen appearance of multiple data files. This is particularly powerful when you review data stored on a read-only device such as a DVD drive.



Global file resources dialog

Unlike the per-file resource files, which are updated automatically whenever you close a resource file, global resource files are never updated automatically. Use Update Global Resource to update the current global resource and Save Resources As to create a new resource file with the current settings for the current view.

The use of global resources is managed from the Global file resources dialog, which has these fields:

Use global resource files

Check this box to use global resources. If this is unchecked, no global resource files are used and each data file has its own resource file. Please remember that using this dialog makes no difference to the resources used by any open time view. It changes the resource file that is used when a saved time view opens. If you check this box and no global resource file is found, Spike2 behaves as if the box were not checked.

Name of file

This field sets the name of the resource file. The name should not include a path or the file extension. These are added automatically, as required. If this field is blank, no resource file is used.

File location

This field sets where to search for the global resource file. You have three choices: *Spike2 installation folder only*, *Data file folder then Spike2 folder*, *Data file folder only*. The Spike2 installation folder is wherever the `sonview.exe` application file is located. The data file folder is the folder from which the data file is opened.

Only use global resource file if

If you do not check any boxes in this area of the dialog, global resource files are applied to all data files. This dialog area allows you to restrict the files that use the global resource file in place of the per-file resources.

Data file is within the path shown below

If you set a path and check this box, only files that lie within this path are considered for global resources. For example, if you only wanted to apply global resources to files read from your DVD drive, you might set this to `E:\` (if `E:\` is the path to your DVD).

Data file does not have its own resource file

Check this box if you would prefer to use the per-file resources if a resource file with the correct name and in the same folder as the data file exists.

Resource Files

This menu item replaces Global Resources when the current view is a time, result or XY view. It opens a pop-up menu from which you can select Global Resources, Apply Resource File, Save Resources As and Update Global Resource.

Apply Resource File

This command applies a user-chosen resource file to the current time, result or XY view. You can use this to apply a complex window arrangement or active cursor measurement to multiple files. The script language equivalent is `FileApplyResource()`. You can also use the Global Resources command to automatically apply a specific resource file.

Note

Resources are saved to disk when the view they belong to closes. If you try to apply resources from an open view to the current view, you will apply the last saved resources of the view, not the current state (which might be different). You can use the File menu Save Resources As... interactive command or the `FileSaveResource()` script command to ensure that resources have been saved before applying them.

Save Resources As

This command saves the resources associated with the current time, result or XY view to a resource file. The command opens a file save dialog for you to set the file name.

Update Global Resource

This command is enabled for the current time, result or XY view if global resources are enabled and a global resource file name is set. It updates the global resource file with the current view settings. If the global resource file does not exist, one is created in the folder set by the Global Resources dialog (or in the application folder if both the data file folder and the application folder are allowed).

Utility programs

This menu item allows you to run additional programs that are installed with Spike2. Selecting a program from this menu item is equivalent to locating the program with the Windows Explorer and running it. Programs are listed if they are found in their expected locations. The list of possible programs includes:

Repair 64-bit .smrx file

This launches the S64Fix program that can recover data from a damaged `.smrx` file. You should consider running this program if Spike2 reports an error with a file or if data appears and disappears depending on where the display starts in the file.

Repair 32-bit .smr file

This launches the SonFix program that can recover data from a damaged `.smr` file.

S2Video

Launches the S2Video program, used to record video from a Microsoft DirectShow™ compatible camera/device during data sampling.

Offline compress MP4/AVI file

If you record video/audio uncompressed you can use these programs to compress the files after sampling.

Test CED 1401 interface

If you suspect a problem with your CED 1401 interface you can use this program to verify that the device passes its self-test. If you have suitable equipment, you can use this program to calibrate the analogue system and check the integrity of the Event inputs and outputs.

Test CED 1902 conditioner
If you have installed the Signal Conditioner support and have a CED 1902 conditioner, you can use this program to check that it is operating correctly.

SoundCard Talker
This option runs an example Talker that allows you to sample waveform data from a Windows audio input as part of data sampling.

Example Talkers
Spike2 is supplied with example Talkers that can be used to familiarise yourself with the general principles of Talkers (TalkerEx) and that can sample the Mouse position as a waveform (MouseTalk).

Close, Close and Link

This command closes the active window. If you use this command on an unsaved window, you are prompted to save it before closing the window. However, if the Edit menu Preferences option is set to not prompt to save modified result and XY views, you will not be prompted to save these views and they will be lost when you close them.

Close all associated windows
If you hold down the `Ctrl` key and click on the File menu when a time view is the current window, the Close command becomes Close and Link. Select this to close the time view plus all associated windows. This does not work with a newly sampled file, save it first. If the Edit menu Preferences option is set to not prompt to save result and XY views, associated result views are deleted. However, in addition to saving the state of the data file in a `.s2rx` resource file, Spike2 also saves the state and contents of associated result view windows in the resource file. Next time you open the data file the result views are recreated from the resource file in the same state as they were closed.

Close an unsaved sampled data file
If you attempt to close a time view that has sampled data and has not been saved, and the sampling session ran for at least 5 seconds and some data was sampled, Spike2 will ask you if you really want to close an unsaved file and tell you the duration. This dialog is deliberately arranged to look different from dialogs you see in other circumstances asking if you want to save files, to make the default action non-destructive. This is because a user complained that they lost a valuable sampled file by keeping accepting default actions when half asleep during an all night sampling session! Note that you can still recover such a file by exiting Spike2 and restarting it, when it will notice that there is an unsaved sampled data file and offer to recover it.

Close a time view with memory channels
This is a potentially destructive act as it may have taken you time and effort to create the memory channels and they will be lost if you close the file. There is an option in the Edit menu Preferences, General tab to configure if you are warned or not. The recommended setting is to not be warned if the file close command came from a script.

Revert To Saved

You can use this command with a text file, a script file or an output sequence file. The file changes back to the state it was in at the last save.

Save and Save As

These commands are available when the current window is a text, script, output sequence, result or XY view or a time view holding newly sampled data that has not been saved or a time view that was created by the script command `FileNew(7,...)`. Save writes the file with its current name unless it is unnamed, in which case you are prompted for a name. Save As writes the file with a different name and gives you the option of saving the data as a different type.

Time view files are kept on disk, not in memory, as they can be very large. Changes made to these files are permanent as they are made on disk. When you save a newly sampled data file, you give the file a name (replacing a temporary name). If you save it to a different drive from that set in the Edit menu Preferences, Spike2 copies the file to the new drive, then deletes the original. Moving a large file can take several seconds.

Saving an unsaved time view causes Spike2 to recalculate the file length, which determines the time range that can be displayed on the x axis. Spike2 only considers channels that have been written to disk, so memory channels and virtual channels are not taken into account.

Sequence, text, result, XY and script files are held in memory. Changes made to these files are not permanent until the file is saved to disk.

Save As for Result and XY views

In addition to saving Result and XY views in their native format, you can also choose to save them as text or as an image. The available formats are:

Text file

This is the same as the Edit menu Copy As Text command, but with the output sent to a text file and not the clipboard. This writes selected channels or all visible channels if no channel is selected.

Bitmap file, JPEG image, Portable Network Graphic file, Tagged Image Format file

These options copy the screen area containing the window to a file; the result is a copy at the screen resolution. If you need to scale the image, or want to edit it, a Metafile copy is often better. The Portable Network Graphic format will usually be the smallest on disk unless the screen contains a real-word image or a sonogram, when JPEG may be smaller (but at the cost of artefacts around axis lines and text). Bitmap images will usually be the largest.

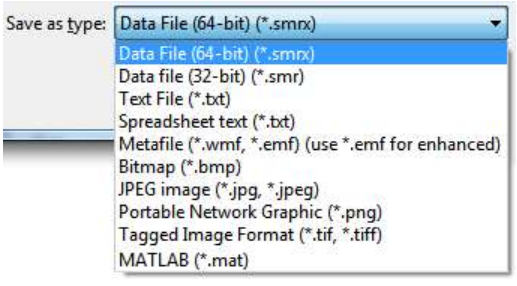
Metafile

This option copies the window as a Windows (Placeable) Metafile (`*.WMF`) or as an Enhanced Metafile (`*.EMF`). Enhanced Metafiles are theoretically better as they support the cubic spline and sonogram drawing modes. However, some graphics programs do not import Enhanced Metafiles very well. The default format is a Windows Metafile. To output in enhanced format, set the file extension to `.EMF`. For example, to save to the file `fred` as a Windows Metafile, set the file name to `fred` or `fred.wmf` but to save as an enhanced metafile set the file name to `fred.emf`.

These file formats can be scaled without losing resolution and are the preferred format for moving Spike2 images to drawing programs. The Edit menu Preferences… option lets you increase (or decrease) the output resolution. This can be important when saving time view and result view data as the number of vectors produced when drawing high resolution may stop some drawing programs from reading the file.

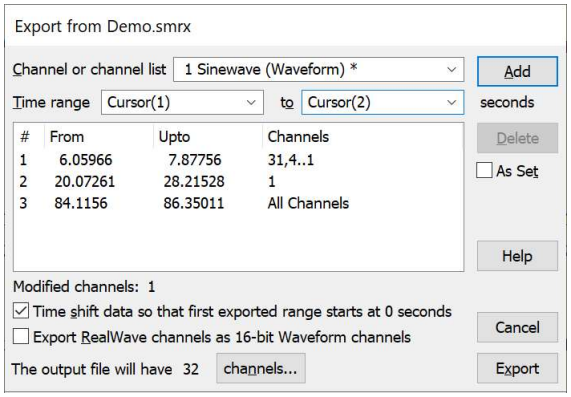
Export As

This menu item replaces Save As when the current window is a saved time view and opens a File Save dialog. You can choose from: Data file (64-bit) (*.smrx) to export as a new 64-bit Spike2 file, Data file 32-bit) (*.smr) to export as an old-style 32-bit Spike2 data file, Text file or Spreadsheet text (*.txt), Metafile (*.wmf or *.emf) for a scalable image and a variety of bitmap image formats. More types are listed (for example export to a MATLAB file) if you have installed external exporters. The script equivalent is `FileSaveAs()`.



Choose a format and either select an existing file to overwrite or type a new file name, then click Save. What happens next depends of the export format:

Data file (64-bit or 32-bit)



A dialog opens in which you select channels and data sections to export. Output is to a 64-bit `.smrx` file unless the selected file name ends `.smr`, when it is to an old-format 32-bit file. Unless you require compatibility with Spike2 version 7 or older or with third party software that does not support `.smrx` files, we suggest you use the new format.

If any selected channels are modified by marker filters or by channel processes, these modifications will apply to the output. The dialog displays a list of modified channels as a warning; the drop down list of channels marks modified channels with an asterisk (*).

The dialog builds a list of time ranges with associated channels. The list displays in order of start times. Time ranges are allowed to overlap. Time ranges can be typed in as x-axis dialog expressions or inserted from the drop down list.

You can choose channels from the drop down list or type in a channel specification. You can export channels from the original file plus memory and virtual channels; duplicated channels are not listed.

If you select one of the Range n items, the channel list and time range fields update to match the selection.

Add

Set channels and a time range with the top section of the dialog and click Add to insert into the range list. Do this as often as required. Bad entries in the channel list or time range or a to time less than the start of the range disable the Add button.

Click Delete to remove a selected list entry.

As Set

When you click the Add button, the Time range fields are evaluated and both the evaluated times and the original contents of the range fields are saved in the range list. If you check the As Set box, the time range list displays the same text as you used when you clicked Add, otherwise it displays the time range in seconds calculated when Add was used. This also affects what is copied back to the Time range fields when you select one of the entries in the list of time ranges.

Modified channels

This field lists all the channels that are currently selected for output that are modified by having a marker filters or a channel processes applied. The exporter writes the modified version of these channels, which may not be what you intend.

Time shift data...

Check this box to time shift the output data so that the Range 1 From time becomes 0.0 in the output file. If the source file stores the sampling start time of day and you shift the time base, the same shift is applied to the start of sampling time.

Export RealWave channels as 16-bit Waveform channels

Check this box to convert RealWave channels (normally stored as 32-bit floating point values) to 16-bit integers using the channel scale and offset. This is usually only required when exporting as a 32-bit file so that versions of Spike2 before 4.03 can read the data. This field is omitted when external exporters use the dialog.

Channels...

This button opens a dialog in which you can set the maximum number of channels in the output file, from 32 to 2000. External exporters omit this field. 32-bit `.smr` files can only have up to 400 channels.

Big file

This field is visible when you export to a 32-bit `.smr` file. Check the Big file box if you want the file written in Big file mode. As files written in this mode are not compatible with versions of Spike2 before version 7 you

should not check this box unless your output file is likely to exceed 2 GB in size. If you export to a 64-bit file this field is omitted as there are no size restrictions. External exporters also omit this field.

Export
Once you have formed a list of times and channels, click Export to write the selected time ranges and channels to the new data file. If the As Set box is checked, all the time ranges are re-evaluated using the current state of the data file, display positions and Cursors. Channels are written in order of ascending channel number. Where possible, channels are copied to the same channel number in the output file. If this is not possible, for example for a memory channel, the channel is written to the lowest numbered unused channel in the output file. If the export process takes more than a couple of seconds, a progress dialog appears and you have the option of cancelling the export.

Tidy up a data file or add extra channel space

If you create new channels in a data file and then delete them, the space allocated on disk for the new channels is not released and still belongs to the file (and will be reused if more channels are written). You can use the Export As command to generate a new copy of the file that does not include this extra space. You can also use this dialog to add more channel space for data proessing. Follow these steps:

1. Open the file to clean up and select the File menu Export As command.
2. Choose a suitable file name and click Save.
3. In the new dialog click on Add to export All Channels from 0 to MaxTime().
4. Clear the Time shift... check box (unless you want the data shifted).
5. Set the number of channels if you want to increase it.
6. Click Export. The new file is added to the File menu most recently used list.

The script language equivalent of this can be found in the description of the `FileSaveAs()` command.

Text file

This is the same as the Edit menu Copy As Text command, but with the output sent to a text file and not the clipboard. This writes selected channels or all visible channels if no channel is selected.

Locale (using comma for decimal point)
In some counties, the character used to separate the integer part of a floating point number from the fractional part is a comma, not a decimal point. If your computer system is set to a locale where this is the case (in Germany, for example), you can enable locale-based text formatting in the Edit menu Preferences General tab.

Spreadsheet text file

Copy a time view as a text file for easy import to a spreadsheet. This is the same as the Edit menu Copy Spreadsheet command, but with the output sent to a file. This writes selected channels or all visible channels if no channel is selected.

Bitmap, JPEG image, Portable Network Graphic, Tagged Image Format file

These options copy the screen area containing the window to a file at the screen resolution. If you need to scale the image, or want to edit it, a Metafile copy is often better. The Portable Network Graphic format will usually be the smallest on disk unless the screen contains a real-word image or a sonogram, when JPEG may be smaller (but at the cost of artefacts around axis lines and text). Bitmap images will usually be the largest.

Metafile

This option copies the window as an Enhanced Metafile (`*.emf`) or as a Windows (Placeable) Metafile (`*.wmf`). Enhanced Metafiles are better as they support the cubic spline and sonogram drawing modes and larger file sizes. The default format is as Enhanced Metafile. To output in the older format, set the file extension to `.wmf`. For example, to save to the file `fred` as an Enhanced Metafile, set the file name to `fred` or `fred.emf` but to save as an old-style Windows Metafile set the file name to `fred.wmf`.

These file formats can be scaled without losing resolution and are the preferred format for moving Spike2 images to drawing programs. The Edit menu Preferences… option lets you increase (or decrease) the output resolution. This can be important when saving time view and result view data as the number of vectors produced when drawing high resolution may stop some drawing programs from reading the file.

Which Metafile format to use?
The Windows Metafile format (WMF) was the original and dates back to 16-bit Windows 3.0; it is no longer recommended for use. Enhanced Metafiles (EMF) date back to 1993 and were the 32-bit Windows replacement for the WMF format. It used to be the case that WMF was well supported by graphics programs and EMF less so. However, nowadays we observe that the reverse is the case, so we strongly recommend that you use the Enhanced Metafile format unless you have strong reasons for using the older format.

Other vector graphics fromats
We are often asked to write images in other formats, for example SVG. There are programs that will convert EMF files to SVG, for example Inkscape can import EMG and then write it as SVG. However, Spike2 images tend to be composed of a large number of vectors and drawing programs are not usually optimised to work well in these cases.

External exporters
These use the same dialog as for exporting to a data file to select channels and a single time range. There may be additional dialogs depending on the target format. See the `Export` folder for additional printed documentation for external exporters or here in the help system. The only external exporter currently defined is for MatLab files.

MATLAB external exporter

If you selected MATLAB support when installing Spike2 you can export a time view, result view or an XY view to a MATLAB file. Choose MATLAB data as the Save As type in the Export As dialog in a time view or in the Save As dialog for result and XY views. You can also control export to MATLAB format from a script.

Working with large files
The writing to MATLAB files is done by calls into MATLAB-supplied routines. There are limits on the size of MATLAB objects that can be written. Use format version 7.3 for huge files (beware that they are compressed, which slows export). You need a 64-bit MATLAB to read gigabyte sized files. If you intend to export large files you must be realistic about how long it may take (several minutes).

Alternatives to exporting data as MATLAB files
It is possible to read (and write) both old format `.smr` and new format `.smrx` files from MATLAB using the SON64 interface for MATLAB that is downloadable from our web site. This package uses the same library as Spike2 for manipulating the data files and allows you to access all the data regardless of file size.

Another alternative is to use the MATLAB script support routines to exchange data with a MATLAB engine via the script language.

Export fails after MatLabOpen() used
If file export works, but fails after you use a script that calls `MatLabOpen()`, see the MatLab Script support, Matlab problems page.

Workspace variable naming

The workspace variables in a mat-file must have names that are both unique and legal. If a variable name is not unique it will overwrite the workspace data with the same name. MATLAB cannot read variables with illegal names. Spike2 mat-file export creates variables with names based upon the source view name and the channel name. You can independently select if either of these is to be used. This produces variable names that are both useful and unique. The default setting is to use the source view name but not the channel name. The variable name settings are used to build the variable name as follows:

If the source view name is to be used, it is placed at the start of the variable name followed by an underscore character '_'. Following this a channel identifier is added. This is either the channel title (if using the source channel name is selected and the name is not blank) or Spike2 builds a name using '`Ch`' followed by the channel number. For example, when exporting a view (of any type) called `Expt1` containing 3 channels; channel 1

called 'ECG', channel 3 called 'AP' and channel 7 with a blank name we would get the following MATLAB variable names:

Using source view and channel name:	`Expt1_ECG, Expt1_AP, Expt_Ch7`
Using source view name only:	`Expt1_Ch1, Expt1_Ch3, Expt_Ch7`
Using channel name only:	`ECG, AP, Ch7`
Using neither name:	`Ch1, Ch3, Ch7`

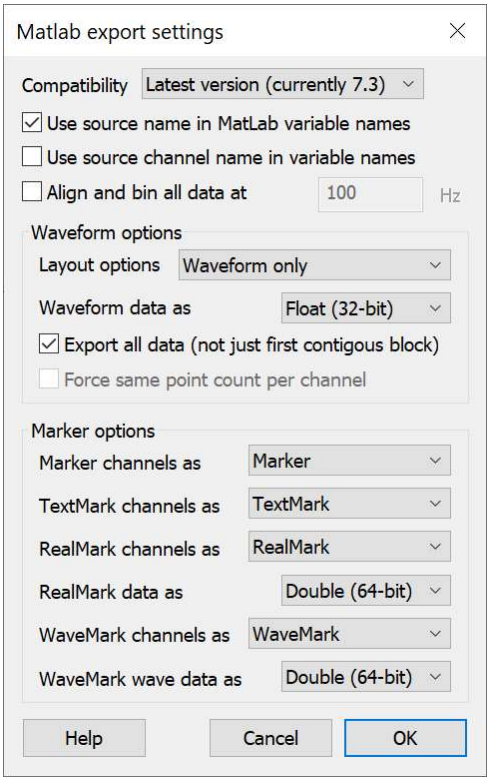
Having generated a name using these rules, Spike2 checks and, if necessary, modifies the name to guarantee that it is legal. The rules for a legal name are that it must not be more than 63 characters, must begin with an alphabetic character and must only contain alphanumeric characters and the underscore character '_'. Spike2 modifies the name by appending a 'V' to the start of the name if it does not begin with an alphabetic character, converting all illegal characters to underscores and finally truncating if more than 63 characters long.

WARNING: If you end up with variable names that are the same, the variables will overwrite each other when you read the file into MATLAB and you will only see the last read variable.

To export data as a MATLAB file, open the File menu Export As command from a time view or the Save As command from a result or XY view and select MATLAB data (*.mat) in the Save as type drop-down list. Set a file name and click Save to proceed to the next dialog where you choose the data to export and the format to export it in. The details of the new dialog depend on the source view type:

Time view data

When exporting time view data, you first get a dialog that allows you to select the channels to export and the time range that you want to export over. There is also a check box that selects offsetting the start of the time range being exported to zero. This dialog is a simplified version of the dialog used to select data to export to a SON file, however you can only select a single set of channels and time range.



Note that, when exporting waveform data to a mat-file, only the first contiguous section of waveform data within the selected time range will be exported. The export data format cannot manage more than one section of contiguous waveform data.

When you have selected your channels and time range you are then presented with a second dialog that controls mat-file export options. These options allow you to set which MATLAB file format to use, how the MATLAB workspace variables will be named, what data is generated for different channel types and the format used for various types of Spike2 data. These options are also available when you export using a script.

Compatibility

If you are using an older version of MATLAB, you must use the compatibility field to select a suitable output file format. Version 7 and 7.3 files are compressed, which can be slow.

Use ... in variable names

These two options in the dialog control how the mat-file variable names are constructed as described previously.

Align and bin all data at

If you check this box, the field to the right of the text is enabled and you can set the frequency at which the output data is generated on all channels. Waveform data and RealMark data draw as a waveform are re-sampled to this frequency using cubic spline interpolation and all other event and marker data is converted to binned event counts using the specified frequency to set the bin width. The options for event and marker channels also change; this is described below.

Waveform options
These controls specify how Waveform and RealWave data is handled.

Layout options
The Layout options item can be set to Waveform only or Waveform and times, if the second option is selected then an array containing all of the sample times is created as well as the array of waveform values.

Waveform data as
The Waveform data as item sets the type of data created for waveforms, it can be set to Integer (16-bit), Float (32-bit) or Double (64-bit). Integers use least memory (but need scaling to user units), float has the full accuracy of the data and is recommended. If you choose Integer for RealWave data, the values are converted to integer using the channel scale and offset (limited to the range -32768 to 32767).

Export all data (not just first contiguous block)
Waveforms channels in Spike2 data files can contain gaps. Normally, data export stops when a gap is encountered. If you check this option, the export ignores the gaps. If you select this option and your data does contain gaps, you will probably want to set the layout option to Waveform and times so that you can detect where the gaps are.

Force same point count per channel
This option is available when all the waveform channels have the same sample rate. Unless all the channels are aligned to the same sample time, in an arbitrary time range some of the channels will have n data points and some will have n+1. If you check this option, the end time for the data export is adjusted forwards in time by up to one sample period so that all the waveform channels will have the same number of points. This can only be successful if the waveform data on all channels is contiguous (no gaps) over the time range set for export and the end time does not hit the last data point on any channel.

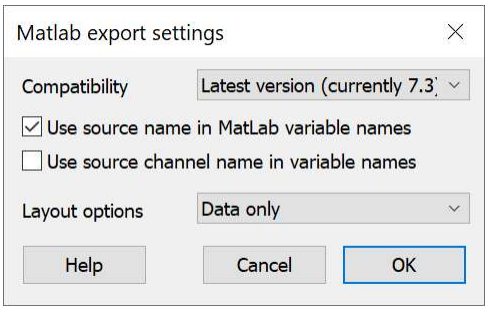
Marker options
The rest of the dialog sets how the various types of marker data are handled. The controls labelled Marker channels as, TextMark channels as, RealMark channels as and WaveMark channels as define what data is generated for the specified type of channel. If these items are set to Event then the channels are treated as events and a single array of times is generated. If they are set to Marker then an additional array containing the marker codes is generated, and (for the extended marker types only), setting them to TextMark, RealMark or WaveMark means that another array containing the additional marker information is also created.

The remaining two controls labelled RealMark data as and WaveMark wave data as set the type of data produced for these channel types, RealMark data can be exported as 32-bit floats or 64-bit doubles while WaveMark wave data can be exported as 16-bit integers as well in the same manner as Waveform data.

If your check the Align and bin all data at check box, the options for markers are replaced by a single check box labelled Generate times for binned data. Check this to generate an array of bin times as well as the bin counts in the same way as for the Layout options item for waveform data.

Error handling
If there is an error in the underlying channel data (for example, if the data file associated with a time view is damaged), the damaged channel will not be exported and you will get an error message. You can recover damaged `smr` files with SonFix and damaged `smrx` files with S64Fix.

Result view data

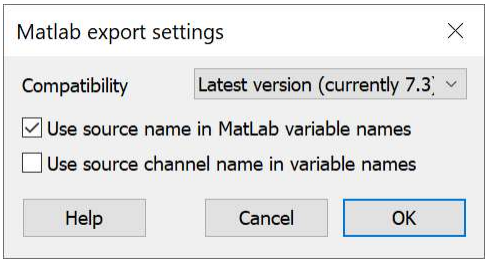


When exporting a result view to a mat-file you are first presented with a simple dialog to select the X axis range of the data to be exported.

This is followed by a second dialog that sets mat-file export options. Use the Compatibility field to select the file format to use. Unless you have an older version of MATLAB you should select the Latest output format.

The next two controls are check boxes to control the channel variable naming as described above. The only other control, labelled Layout options, can be set to Data only if you only want to export the result bin values or Data and times if you want to export both the bin values and the bin x axis positions.

XY view data



For an XY view you do not get a dialog to control what data is exported; all the visible channels are exported and only data points that lie within the displayed X and Y axis ranges are used.

A dialog is provided to control how the data is exported. This is a simplified version of the result view export options dialog; it contains the two standard check boxes to select the use of the source view name and channel names to create the MATLAB variable name plus the choice of which version of the file system to use.

Mat file data format

A mat-file represents a collection of MATLAB variables rather than simple data values, which allows for a complex representation of Spike2 data within MATLAB. Each Spike2 channel exported into a mat-file is represented by one variable This variable is a structure containing fields that hold information about the channel and the channel data. For example, a structure holding data from a waveform channel has the fields `title`, `frequency` and `start` each of which is a simple (scalar) datum holding the channel title, the sampling frequency in Hz and the time of the first data point. In addition there is a field called `wave` that is a `n` x `1` matrix holding the waveform data values and an optional field called `times` which is a `n` x `1` matrix holding the sample time for each of `n` waveform points. The structure varies according to the channel type, though some fields are common to all channels.

Time view

Exported time view data also exports a structure called `file` that contains two fields:

`name` A string variable holding the complete path and name of the SON file.

`start` A 6x1 matrix of 16-bit integer data holding the date and time at which sampling of this file started (time 0 in the file), as year, month, day, hours, minutes, seconds. This format is suitable for use as a MATLAB `datetime` value.

Waveform RealWave and binned RealMark data drawn as a waveform

Waveform data is exported as a 1-dimensional array of waveform values with an optional associated array of sample times. The fields in the channel structure are:

`title` a string holding the channel title.

`comment` a string holding the channel comment.

`interval`	a double holding the sample interval or bin width in seconds.
`scale`	a double holding the channel scale factor. The scale and offset values convert 16-bit ADC data to real values using the equation: real = (ADC * `scale`) + `offset`.
`offset`	a double holding the channel `offset` value (see above).
`units`	a string holding the channel units.
`start`	a double holding the time of the first waveform point in seconds
`length`	a double holding the number of waveform items.
`values`	a `length` *x* 1 array holding the waveform values, either raw or produced by interpolating to the bin times. Depending upon the output options selected, this array could be double or single-precision real values or 16-bit integers.
`times`	a `length` *x* 1 array of doubles holding the sample times of the waveform values in seconds. This field is only present if waveform times are selected.

If you choose to export a RealWave channel as 16-bit integers, the Spike2 channel scale and offset (as seen by double clicking the channel title in Spike2 or accessed from the Spike2 script language with the `ChanScale()` and `ChanOffset()` commands) are used to convert real values into integers. If the result exceeds the 16-bit signed range (-32786 to 32767), the result will wrap around.

Event channels binned and un-binned

Event data is represented as a 1-dimensional array of event times with (for Level channels) a corresponding array of levels. If the data is binned, this is replaced by an array of bin counts and an optional array of bin times. This format is used for all types of marker channel if they are exported as event data as well as for event channels. If the data has been binned, this format is used for all types of marker. The fields in the channel structure for both types of data are:

`title`	a string holding the channel title.
`comment`	a string holding the channel comment.
`length`	a double holding the number of events or binned values.

For un-binned data, the extra fields are

`resolution`	a double holding the underlying timing resolution in seconds.
`times`	a `length` *x* 1 array of doubles holding the event times in seconds.
`level`	a `length` *x* 1 array of byte values holding the level for level channels only. The value is 1 if the transition at the corresponding time was upwards, 0 for a downwards transition

For binned event and marker data these fields are replaced by:

`interval`	a double holding the bin width in seconds.
`start`	a double holding the time of the first bin in seconds.
`values`	a `length` *x* 1 array of doubles holding the binned event counts.
`times`	a `length` *x* 1 array of doubles holding the times in seconds for the corresponding bins. This field is present if bin times are selected.

Marker channels

Marker data is represented in the same way as simple event channels but with the addition of an extra length-by-4 byte array of marker codes. This format is used for all types of extended marker channel if they are exported as markers. The fields in the channel structure are:

`title`	a string holding the channel title.
`comment`	a string holding the channel comment.
`resolution`	a double holding the underlying timing resolution in seconds.
`length`	a double holding the number of markers.
`times`	a `length` *x* 1 array of doubles holding the marker times in seconds.

`codes`	a `length` *x* 4 array of byte values holding the marker codes.

TextMark channels

TextMark data is represented in much the same way as simple markers but with the addition of an extra 2 dimensional array holding the marker text. The fields in the channel structure are:

`title`	a string holding the channel title.
`comment`	a string holding the channel comment.
`resolution`	a double holding the underlying timing resolution in seconds.
`length`	a double holding the number of markers.
`items`	a double holding the maximum number of characters in each marker.
`times`	a `length` *x* 1 array of doubles holding the marker times in seconds.
`codes`	a `length` *x* 4 array of byte values holding the marker codes.
`text`	a `length` *x* `items` array of char values holding the marker text.

RealMark channels

RealMark data is represented similarly to simple markers but with the addition of an extra 2 dimensional array holding the real values and units information. The fields in the channel structure are:

`title`	a string holding the channel title.
`comment`	a string holding the channel comment.
`resolution`	a double holding the underlying timing resolution in seconds.
`units`	a string holding the channel units.
`length`	a double holding the number of markers.
`items`	a double holding the number of real values per marker.
`times`	a `length` *x* 1 array of doubles holding the marker times in seconds.
`codes`	a `length` *x* 4 array of byte values holding the marker codes.
`values`	a `length` *x* `items` array holding the marker values. These are either single or double-precision real values, set by the export option selected.

WaveMark channels

WaveMark data is represented in the same way as simple markers but with the addition of an extra 3 dimensional array holding the waveform values and additional calibration information as provided for waveform values. The fields in the channel structure are:

`title`	a string holding the channel title.
`comment`	a string holding the channel comment.
`resolution`	a double holding the underlying timing resolution in seconds.
`interval`	a double holding the waveform sample interval in seconds.
`scale`	a double holding the waveform scale factor. The scale and offset values convert 16-bit ADC data to real values using the equation: real = (ADC * `scale`) + `offset`.
`offset`	a double holding the waveform offset value (see above).
`units`	a string holding the waveform data units.
`length`	a double holding the number of markers.
`items`	a double holding the number of waveform values per trace per marker.
`trigger`	a double holding the offset to the trigger point within a trace in the waveform data. This value will be from 0 to items-1;

`traces`	a double holding the number of traces within the waveform data (normally the number of electrodes).
`times`	a `length` *x* 1 array of doubles holding the marker times in seconds.
`codes`	a `length` *x* 4 array of byte values holding the marker codes.
`values`	a 3-dimensional array (`length` *x* `items` *x* `traces`) holding the marker waveforms. For single trace data the third dimension is 1 so it becomes a `length` *x* `items` matrix. Depending upon the output options selected, this array could be double or single-precision real values or 16-bit integers.

Result channels

Result data is represented in much the same way as waveform data but without any calibration information. Extra information is provided about the X axis. The fields in the channel structure are:

`title`	a string holding the channel title.
`units`	a string holding the channel (Y axis) units.
`xunits`	a string holding the X axis units.
`interval`	a double holding the result bin width in the appropriate X axis units.
`start`	a double holding the X axis value of the first bin.
`length`	a double holding the number of result bins.
`values`	a `length` *x* 1 array of doubles holding the result bin values.
`times`	a `length` *x* 1 array holding the X axis values for the corresponding bins (these are not always time values). This field is only present if result bin times are selected.

XY channels

XY data is represented as two 1-dimensional arrays of X and Y data plus ancillary information. The fields in the channel structure are:

`title`	a string holding the channel title.
`units`	a string holding the Y value units.
`xunits`	a string holding the X value units.
`length`	a double holding the number of XY points.
`xvalues`	a `length` *x* 1 array of doubles holding the x values of the data.
`yvalues`	a `length` *x* 1 array of doubles holding the y values of the data.

Script export to mat files

Export to mat-files is also available from the script language by using the `FileSaveAs()` function with the file type set to 100. With this type in use, the file name selection dialog (if provided) will use a '`*.mat`' file name filter, an extra `exp$` argument becomes available to allow MATLAB-export specific options to be set and the time range and channels to be exported are set by `ExportChanList()`. Only the first set of time range plus channels is used, and `ExportChanList()` is additive in action, so you should first use `ExportChanList()` with zero (or one) arguments to clear out any stored sets of information before setting the time range and channels you want. The extra `exp$` argument that sets options is a string of the form:

```
name=value|name=value|...|name=value
```

where `name` specifies some export option and `value` sets it's value. You can include as many options as you want, options that you omit are set to the default value. Option names are not case-sensitive. It is not an error to use an unknown option.

BEWARE: If you set `UseCName` to 1 and you have channels with duplicated names, Spike2 will write the data, but MATLAB will only see the last channel as each variable with the same name overwrites the previous one when data is read.

Time view export options

`UseSName` selects use of the source name in the channel variable name. Set 1 (default) to use the source name, 0 if not.

`UseCName` selects use of the channel name in the channel variable name. Set to use the channel name, 0 if not. The default is 0.

`BinFreq` selects binning of the data. Set 0 (default) for un-binned, otherwise set to the binning frequency in Hz.

`BinTimes` selects generation of an array of the times of the bins (ignored if the data is not being binned). Set 1 for bin times, 0 if not. The default is 0.

`WaveTimes` selects generation of an array of waveform sample times in the channel variable. Set 1 for times, 0 (default) for none.

`WaveData` selects the data type generated for waveform channels. 0 for 16-bit integers, 1 for single-precision real data and 2 for double-precision real data. The default is 2.

`WaveAll` Set to 0 to stop waveform or RealWave export at a gap. Set to 1 to ignore gaps. The default is 0.

`WaveSameN` Used when all waveform channels have the same sample rate. Set to 1 to adjust the end of the time range so that all channels have the same number of points. Set to 0 (default) to use the time range as supplied.

`MarkAs` sets how marker data is exported. Set to 0 for events and 1 for markers. The default is 1.

`TMarkAs` sets how text marker data is exported. Set to 0 for events, 1 for markers and 2 for text markers. The default is 2.

`RMarkAs` sets how real marker data is exported. Set to 0 for events, 1 for markers and 2 for real markers. The default is 2.

`RMarkData` selects the data type generated for the real values in RealMark channels. 1 for single-precision real data and 2 for double-precision real data. The default is 2.

`WaveMarkAs` sets how wave marker data is exported. Set to 0 for events, 1 for markers and 2 for wave markers. The default is 2.

`WaveMarkData` selects the sort of data generated for the waveform in WaveMark channels. Set to 0 for 16-bit integers, 1 for single-precision real data and 2 for double-precision real data. The default is 2.

Result view export options

`UseSName` selects use of the source name in the channel variable name. Set to 1 if you want to use the source name, 0 if not. The default is 1.

`UseCName` selects use of the channel name in the channel variable name. Set to 1 if you want to use the channel name, 0 if not. The default is 0.

`BinTimes` selects generation of an array of result bin times in the channel variable. Set to 1 if you want bin times, 0 if not. The default is 0.

XY view export options

`UseSName` selects use of the source name in the channel variable name. Set to 1 if you want to use the source name, 0 if not. The default is 1.

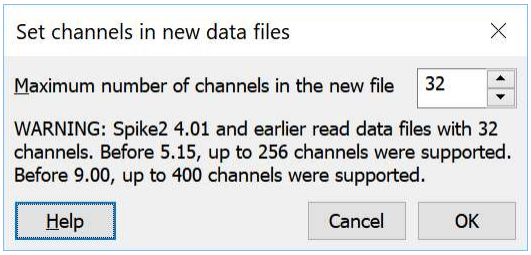
`UseCName` selects use of the channel name in the channel variable name. Set to 1 if you want to use the channel name, 0 if not. The default is 0.

Compatibility

`Compat` Sets which version of the MATLAB filing system to support. 0 (default) for the most recent MATLAB file format known to the system, 1 for version 4 and earlier, 2 for version 6, 3 for version 7 and 4 for version 7.3 (supports files > 4 GB).

Channels in the exported file

The Set... Channels... button in the Export As dialog opens the Set channels in new data files dialog. The minimum number of channels in a data file is 32 and the maximum is currently limited to 2000 (64-bit files have a theoretical limit of 65535 channels, but this is not yet supported by Spike2).



Setting a larger number of channels increase the size of the file header, which holds the channel title and any units and sample rate information. It does not allocate data space in the file, so apart from an increase in the channel header size and in memory usage per file, there is no other penalty. The 32-bit filing system does not allow you to change the number of channels once the file is created. The 64-bit filing system theoretically can add more channels after the file has been created and used, but there is no support for this in Spike2 yet. It is a good idea to allocate enough channels for your foreseeable data analysis needs.

If you will share your files with people using older versions of Spike2 you should be aware that before Spike2 version 4.02, only (32-bit) data files with 32 channels were readable. Version 5 reads 32-bit files with up to 256 channels (5.15 onwards can read 400 channels). Version 8 can read 64-bit `.smrx` files with up to 400 data channels. You need at least version 9 to read files with more than 400 channels.

Load and Save Configuration

These commands manage Spike2 configuration files (`.s2cx` extension). Configuration files hold the full sampling set up including any output sequence file, waveform output and window arrangement and the types of on-line analysis required.

Load Configuration

You cannot update the sampling configuration if Sampling is in progress. This opens a sub-menu with the options Open..., Load Default... and a list of recently loaded configuration files.

Open... This opens a file select menu from which you can select a configuration file (extension `.s2cx`) holding a full sampling configuration, or you can select a Spike2 data file (`.smr` or `.smrx` extension) and Spike2 will do the best it can to extract a sampling configuration that would generate a similar data file to the selected file.

Load Default... This option is new in Spike2 [11.00]. It attempts to load the default configuration. This is the same configuration file that Spike2 loads on start up. The seatch is first for the file `DEFAULT.s2cx` (unless this is disabled in the Edit menu Preferences), then for the file `LAST.s2cx`. Each file is searched for in the User data folder, then the Local Application data folder and finally in the Spike2 installation folder.

File list From Spike2 [10.13] you can also select from a list of up to 10 recently-used configuration files. Files are added to this list when they are loaded or saved. The most recently used file is at the top of the list.

The script language equivalent of this command is to use `FileOpen()` with a file type of 6.

After reading a configuration, the Sampling Configuration dialog opens to display it. If you wish to sample immediately, click the Run Now button.

You can also click a button in the Sample Bar to load a pre-set configuration and start sampling.

Spike2 ignores application window and dialog positions held in `.s2cx` configuration files if less than 25% of the window is on the screen area, or if the window title bar would be above the screen (as this makes it very difficult to move the window).

Limitations of reading configurations from data files

Data files hold only the channel settings; they do not hold window arrangements, output sequencer, waveform output, Talker or data processing settings. Select Data files (*.smrx;*.smr) in the dialog to read a data file in place of a configuration file. **There is no guarantee that a configuration read in this way will exactly match the data file**; you must check the result carefully and you may need to adjust the resulting configuration. If the waveform rates do not match the file rates, navigate to the Resolution tab and experiment with setting Burst mode and using the Suggest button.

You cannot reconstruct the sampling configuration for Talker-based, Derived or Processed channels from a data file.

Problems reading configurations
If Spike2 detects a problem reading the configuration it opens a message box with an explanation. If the message box has a Help button, this leads to an explanation of what has gone wrong and may have suggestions for fixing it. For example, loading configurations with uninstalled Talkers can lead to problems.

Save Configuration
This command saves the current sampling configuration to a `.s2cx` configuration file. It opens a File Save dialog in which you can choose where to save the configuration.

Save Default Configuration
This command is normally hidden. You activate it by holding down the `Ctrl` key before you select the File Menu. This saves the sampling configuration to the file `DEFAULT.s2cx` in the current users application data directory. Once you do this, each time Spike2 starts it will load this configuration unless you set the *Ignore Default Configuration* check-box in the Edit menu Preferences Sampling tab. Choosing this option clears the *Ignore Default Configuration* flag if it is set. Alternatively, locate the `DEFAULT.s2cx` file and delete it.

Tip: to get the path to the current configuration on the Windows Clipboard, open the Sampling Configuration dialog and right-click on the title bar and select the option to Copy the path to the file.

Default configuration files: DEFAULT.s2cx, LAST.s2cx
If the configuration file `DEFAULT.s2cx` is found, it is loaded when Spike2 starts. To save this file, hold down the control key while activating the File menu and use the Save Default Configuration command. The standard file `LAST.s2cx` holds the last configuration that was used for sampling. If `DEFAULT.s2cx` cannot be found, and `LAST.s2cx` exists, `LAST.s2cx` is loaded. Spike2 saves `LAST.s2cx` each time sampling stops. If no file is found, Spike2 starts up with a standard configuration with no data channels set.

Prevent default files from being used
There is a setting in the Edit Menu->Preferences in the Sampling Tab to disable the use of all default files to ensure that the last used sampling configuration (`LAST.s2cx`) is always used.

Where does Spike2 search for these files?
Spike2 is normally installed into the Program Files folder, but we want to remain compatible with old versions of Spike2, which were stored in a user-selected folder. To do this, we search a list of places to find the `default` and `last` files. The search order is:

1. The application data directory (but read below for what this means)
2. The directory in which Spike2 is installed

Prior to Spike2 version 9, we searched each directory for the `s2cx` file first, then for the `s2c` file (for backwards compatibility). However, version 9 onwards no longer supports `.s2c` configuration files (so if you want to use one, last written by Spike2 version 7.10, you should open it in version 8, use it to sample data, which converts the format, then save it). The application data directory is the first of the following that exists:

1. The current users application data directory
2. The all-users application data directory
3. The "My Documents" folder

Double-click .s2cx files to open in Spike2
The Spike2 installation program attempts to set the system file associations so that double-clicking a configuration files opens it in Spike2. If double-clicking a `.s2cx` file in the shell (desktop or a file folder window) does not open the file in Spike2, or opens an old version of Spike2, you should check that the file association are set correctly. If you have Administrator right the simplest way to set file associations is to run Spike2 as Administrator by right-clicking the Spike2 icon and selecting Run as Administrator. You should only need to do this once.

If you run in User mode on an administrated system you will have to ask your IT department to temporarily give you enough rights to do this.

Exit

This command closes all open files and exits from Spike2. If there are any text or output sequence files open that have not been saved, you will be prompted to save them before the application terminates.

If there are newly sampled and unsaved time views, you will be asked if you are sure that you want to exit without saving in the same way as if you had attempted to close the Time window.

Send Mail

If your system has support for Mail installed (for example Microsoft Exchange), you can send documents from Spike2 to another linked computer. This option vanishes if you do not have compatible mail support.

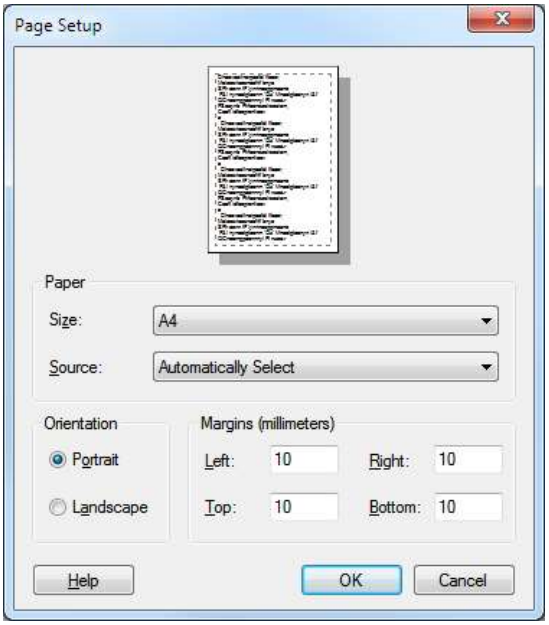
Text-based documents and result views can be sent, even if they have not been saved to disk (Spike2 writes them to a temporary file if they have not been saved).

You can send a Spike2 data file, but only if you are not sampling to the file and it has been saved on disk and is unmodified. Spike2 makes a temporary copy of the file in the system temporary folder before mailing it to avoid problems with mail programs that will not send a file if it is open in another application. You must have enough spare disk space for at least 2 copies of the data file.

Printing

Page Setup
Page Headers and Footers
Print Preview
Print Visible, Print and Print Selection
Print Screen

Page Setup



This opens the printer page set up dialog. The dialog varies between operating systems and printers. See your operating system documentation for more information. The important options that are always present include the paper orientation (portrait or landscape), the paper source (if your printer has a choice), and the printer margins.

The Orientation option (portrait or landscape) applies to all output except the Print Screen option, which has its own selector for the output orientation.

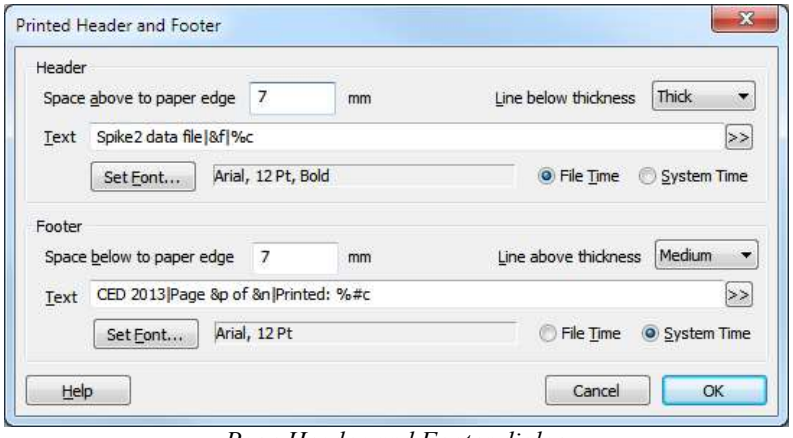
The printer margins will appear in inches or in millimetres, depending on the locale set for your computer. These margins are used for all printed output. The left and right margins are applied to everything, including headers and footers. The top and bottom margins apply to everything except headers and footers, which have their own top and bottom margin settings (see the Page Headers description). The top and bottom margins you set here are further modified if a header or footer would collide with them.

Most printers have an unprintable area near the paper edge. If you set margins smaller than the unprintable area, the margins are increased so that all your output is visible. If you set margins that reduce the printable area too much, the margins are ignored.

Registry use

Spike2 saves the page margins in units of 0.01 mm in the system registry. You can find them in the `HKEY_CURRENT_USER\Software\CED\Spike2\PageSetup` folder as `REG_DWORD` values: `LeftMargin`, `RightMargin`, `TopMargin` and `BottomMargin`.

Page Headers and Footers



Page Header and Footer dialog

You can apply headers and footers to all printed output. The headers set in this dialog are used with all Time, Result, XY and text-based views. The header and footer text is displayed in the page format set by the Page Setup command.

Other printed output (for example from the Print Screen command) uses all these settings except the header and footer text, which is provided by each of the other printing commands.

Header and footer positions

The horizontal position is set by the left and right margins in the Page Setup dialog. The vertical positions are set by the space above the header and the space below the footer fields in inches or millimetres, depending on the locale. If your header or footer encroaches on the top and bottom margins set in the Page Setup dialog, the top and bottom values are adjusted to keep the output clear of the header and footer.

Line thickness

You can choose between: None, Hairline, Thin, Medium and Thick. A Hairline is the thinnest line possible on the output device. The other settings should be self-explanatory. The header line is drawn below any header text, the footer line is drawn above any footer text. If there is no header or footer text, no line is drawn.

Text

This is the text to display as the header or footer. If this field is empty, the header or footer is omitted (including any line). You can split the text into left-justified, centred and right-justified sections with the vertical bar character. You can also insert codes that are replaced by document and time information. The simplest way to do this is by clicking the >> button to the right of the text and choosing an option.

Set Font

Click this button to choose a font for the header and the footer. Font sizes are limited to 2 to 30 points.

File/System Time

You can insert times into both the header and the footer. However, you have to choose between the current time and the file time. This allows you to display the file time in the header and the current time in the footer, or vice versa.

Document information

The following codes are replaced by document information:

`&f`	File and path	`&F`	Upper case file and path
`&t`	Document title	`&T`	Upper case document title
`&p`	Page number	`&n`	Total number of pages
`&&`	The ampersand sign (`&`)		

Time and date codes

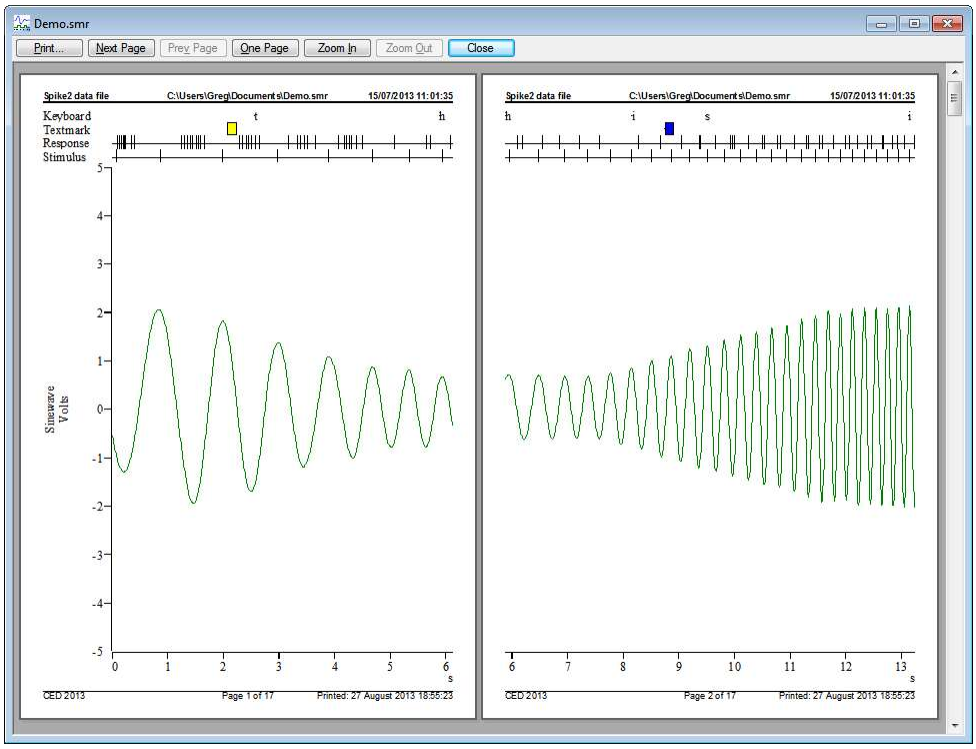
You can insert times and dates using % followed by a character code. The combination is replaced as described in the table. You can also use `%#c` and `%#x` for a long version of dates and times. You can remove leading zeros from numbers with `#`, for example `%#j`.

`%a`	Short day of week (Mon)	`%p`	Indicator for A.M or P.M.
`%A`	Long day of week (Monday)	`%S`	Seconds as number (00-59)
`%b`	Short month name (Jan)	`%U`	Week of year, Sunday based (00-53)
`%B`	Long month name (January)	`%W`	Week of year, Monday based (00-53)
`%c`	Date and time for locale	`%w`	Sunday based weekday (0-6)
`%d`	Day of month (01-31)	`%x`	Date formatted for locale
`%H`	Hour in 24 hour format (00-23)	`%X`	Time formatted for locale
`%I`	Hour in 12 hour format (01-12)	`%y`	Year without century (00-99)
`%j`	Day of year (001-366)	`%Y`	Year with century, e.g. 2004
`%m`	Month as number (01-12)	`%z/Z`	Time zone name
`%M`	Minute as number (00-59)	`%%`	The percent sign (`%`)

Registry use

Spike2 saves the header and footer settings in the system registry. You can find them in the folder `HKEY_CURRENT_USER\Software\CED\Spike2\PageSetup` as strings: `Header`, `Header info`, `Footer` and `Footer info`. The `Header` and `Footer` items hold the text strings. The two `info` items are strings that code up the font name, point size, bold and italic settings, distance to the paper edge in units of 0.01 mm, line thickness (0=none, 1=Hairline, 2=Thin, 3=Medium, 4=Thick), and File time (0) or System time (1).

Print Preview



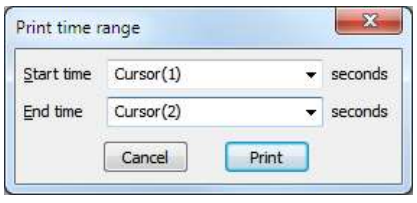
This option displays the current time, result, XY and text-based window as it would be printed by the Print option. You can zoom in and out, view one or two pages, step through pages of multi-page documents and print the entire document using a toolbar at the top of the screen. Close leaves this mode without printing.

The preview takes place inside the frame of the time, result or XY view. You can access the File menu to change the headers and footers and print margins.

Print Visible, Print and Print Selection

These commands print time, result, XY and text-based windows. Any scroll bar at the bottom of the window is not printed. The Edit menu Preferences dialog sets the line widths used on screen and for printing. Print Selection prints the selected area of a text window. Print Visible prints the visible data in the current window. Print prints a specified region of a time, text or result view; each printed page holds the x axis range of the window. Print in an XY view is the same as Print Visible.

Print

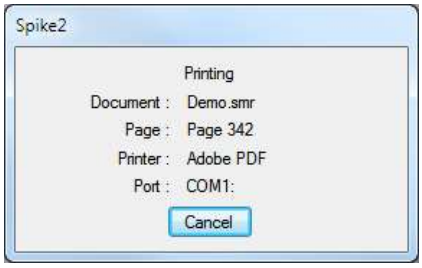


To print an entire data file, set the width of the time window holding the file to the page size required in the print, then select Print. Set the start time to zero and use the drop down list to set the end time to Maxtime(). Beware that Print could require several hundred miles of paper to complete the printing job in the worst case! If you displayed 10 milliseconds of data across the screen in a file that is 1000 seconds long, there are 100,000 pages to print.

Print Visible

This is equivalent to the Print command with the x axis range set to the displayed x axis (except for XY views, where it is identical to Print).

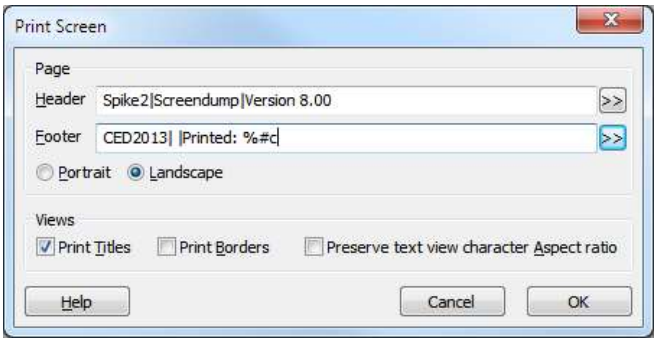
After selecting Print



Both commands open the standard print dialog for your printer. You can set the print quality (in general, the better the quality, the longer the print takes) and you can also go to the set up page for the printer. Once you have set the desired values, click the Print button to continue or the Cancel button to abandon the print operation.

During the print operation (which can take some time, particularly if you selected a lot of data) a dialog box appears. If your output spans several pages, the dialog box indicates the number of pages and the current page, so you can gauge progress. If you decide that you didn't want to print, click the Cancel button.

Print Screen



The Print Screen command prints all time, result, XY. Grid and text based views to one printer page. The views are scaled to occupy the same proportional positions on the printed page as they do on the screen. The page margins are those set by the Page Setup and Page Headers dialogs.

The command opens a dialog with two regions: Page and Views. In the Page region you can set a page header and footer and choose to print in Landscape or Portrait mode. The header text is printed with an underline across the width of the page. The footer text is printed with a line over it across the page width. The fonts used and line thickness are as set in the Page Headers dialog. If the header or footer is blank, both it and the associated line are omitted.

You can divide the header and footer into a left justified, centred and right justified sections with the vertical bar character, for example: Left|Centered|Right. You can include the current date and time in the header or footer by including, for example `%c`, as described for the Page Headers and Footers dialog. The >> button can be used to insert the time and date and vertical bar separators into the header and footer.

In the Views region you can choose to print view titles and draw a box round each view. You can also ask Spike2 to attempt to preserve the aspect ratio of characters in text windows. This feature is currently not supported.

Registry use

Spike2 saves the Print Screen settings in the system registry. You can find them in the `HKEY_CURRENT_USER\Software\CED\Spike2\PageSetup` folder. The text strings are saved as `PSHeader` and `PSFooter`. The remaining values are saved as `REG_DWORD` values of 0 (not selected) and 1(selected): `PSViewTitle`, `PSBorder`, `PSScaleText` and `PSLandscape`.

7: Edit menu

Edit menu

This menu holds the standard Edit functions that all programs provide. The majority of the menu is associated with commands that move data to and from the clipboard.

Undo and Redo

In a text, script or output sequence window this is used to undo or redo the last text edit operation. You cannot undo operations that have been saved to disk or text operations that were done by a script.

In Time, Result, XY and Grid windows, you can undo most operations that change the appearance of the window. Cursor movements do not undo; this is by design so you can zoom in, adjust a cursor, then undo to zoom out, leaving the cursor adjusted.

Cut

You can cut editable text to the clipboard in any position in Spike2 where the text pointer is visible. You cannot use this command in Spike2 data document windows or in result windows.

Copy

You can copy editable text to the clipboard plus selected fields from the Cursor Regions and Cursor Values dialogs. If you use this command in a time, result or XY window, the contents of the window, less the scroll bar, are copied to the clipboard as both a bitmap and as a metafile. See also Copy As Text.

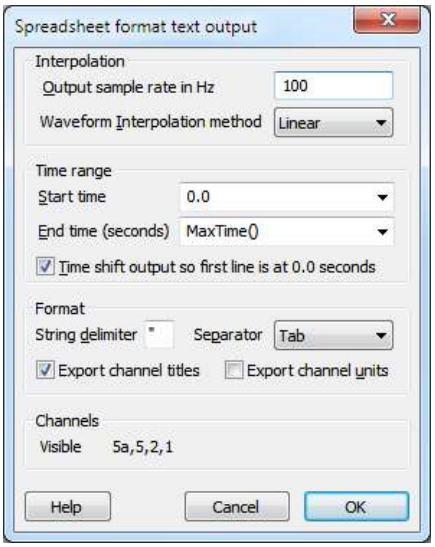
Metafile output

To export an image to a drawing program for further annotation and manipulation, you can paste the image as either a bitmap or as a metafile. Metafiles are usually the preferred choice as you can treat the image as lines and text for further work. You can set the metafile scaling and if the image is saved as a Windows Metafile or as an enhanced Metafile in the Edit menu Preferences.

Other formats (PDF, SVG, EPS, ODF...)

The free open source program InkScape will accept the output of the Copy command from a Time, Result or XY view as an image, or you can save the image to a file as a metafile and import it into InkScape. You can then export the image in a wide variety of formats.

Copy for Spreadsheet



This copies selected time view data channels to the clipboard as text. Use Export As to copy to a text file. If there are no selected channels, it copies visible channels. The Channels area displays how the channels were chosen (Visible or Selected) and the list of channels that will be output.

The data is written in columns. The first column holds the time of each row (in seconds). The remaining columns hold the data, one column per channel.

Channel output format

Where possible, the output matches the display. Channels drawn with a y axis output values that match the display. Channels drawn in State mode output the state code as text. Sonogram draw mode is output as waveform mode.

From [10.12], Level event channels (unless drawn as a Rate or Mean frequency) display the level at the time of the output line (as 1 or 0). Prior to this version, they drew in the same way as Event- and Event+ channels.

All other drawing modes output the number of events from the time of the current line up to the time of the next line.

Interpolation

Spike2 allows each channel to have independent rates and types. To make Spike2 data easily accessible to other programs we re-sample the data to a common rate, set by the Output sample rate in Hz field.

In general, the sample rates of Waveform, RealWave and WaveMark data will not match the output rate you have chosen and the program must interpolate. You can choose between Linear, Cubic spline interpolation, or use the value at the Nearest data point. Waveform data drawn in Sonogram mode is treated as a waveform. All other channels that need interpolation use linear interpolation.

Time range

The Start time and End time fields set the time range to copy. You can also shift the data so that the first line has a time of 0.0 to make comparisons of time ranges easier.

Format

Values are written in numeric or text columns. There is a separator between columns; choose from Tab, space or comma in the Separator drop down list. The String delimiter character is placed around text output. If you check the Export channel titles box, the first line of output holds titles to identify the columns (the channel number and title string). If you check the Export channel units box, the second (or first if the titles box is not checked) output line holds the units of each column.

Copy As Text Result view

This command is available in result views. It copies the bin values in the window for all visible channels to the clipboard as text. To copy all bins, double click the x axis and select Show All, then copy. The first output column holds the x value at the left of each bin. Each channel contributes one column of bin contents plus a column of error bar sizes if error bars are enabled and a column of bin counts if bin counts are enabled. The first line holds column titles. This command does not copy channels drawn as raster data. The first example is from a result view holding two waveform averages:

```
"Time" "Filtered"    "Sinewave"
0      0.84302088    3.2980477
0.01   0.27032173    2.2613753
0.02   -0.45484864   1.043198
0.03   -0.74969506   0.63429488
```

The columns are separate by Tab characters. The second example is from the same data with error bars enabled and drawn in SEM mode.

```
"Time" "Filtered"    "SEM"         "Sinewave"    "SEM"
0      0.84302088    0.038987886   3.2980477     0.032728175
0.01   0.27032173    0.021425654   2.2613753     0.036360926
0.02   -0.45484864   0.027970654   1.043198      0.056437311
0.03   -0.74969506   0.037386934   0.63429488    0.050863126
```

Copy As Text XY view

This menu item is available in XY views. It copies the visible points for visible channels to the clipboard. If the view has one channel or the Measurement system created it with the All channels use same X option, the output is a rectangular table with the first line holding column titles. There is one column of x values followed by one column per channel for each y value. The columns are separated by a tab character:

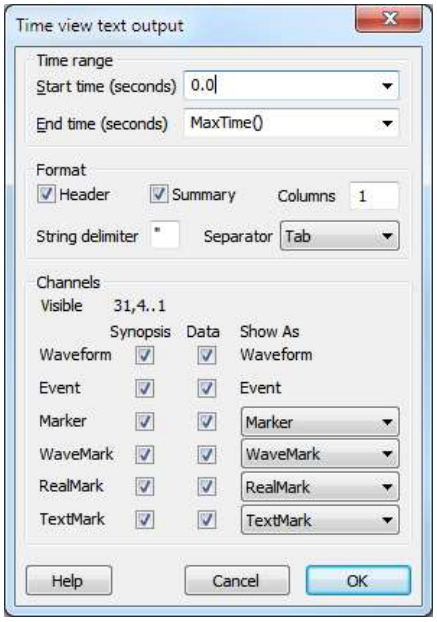
```
"X" "Channel 1"  "Channel 2"
0   0.244140625  0.0170616301
2   3.122558594  0.0001053187043
4   2.211914063  0.0006319122258
6   1.889648438  -0.0005265935215
8   0.8642578125 -0.0005265935215
```

In all other cases, channels are output separately. For each channel, the first output line holds "Channel : cc : nn" where cc is the channel number and nn is the number of data points. The data points are output, one per line as the x value followed by the x value, separated by a Tab character.

```
Channel : 1 : 5
0.0170616301     0.244140625
0.0001053187043  3.122558594
0.0006319122258  2.211914063
-0.0005265935215 1.889648438
-0.0005265935215 0.8642578125
Channel : 2 : 5
0                0.0170616301
2                0.0001053187043
4                0.0006319122258
6                -0.0005265935215
8                -0.0005265935215
```

Copy As Text Time view

In a time view the command opens a dialog in which you specify the time range of data to copy, how you want the data to be copied and the output format. Select the channels to copy in the time view before you use this option. If you select no channels, Spike2 copies visible channels. Text representations of sampled data can be very large and awkward to manipulate with the clipboard. Alternatively, you can write the text output to a file with the File menu Export As command.



You can enable and disable a header section for the entire output and a summary section of all the channel information. You can also set the number of columns to be used when writing waveform and event times and the separators used between fields and to delimit text. For each type of channel you can enable the channel synopsis and data output. You can also choose to dump a channel type in its native format, or in any compatible format. Output sections are preceded by a keyword so that other programs can parse the file.

Time range

Select the time range of data to copy as text in this dialog. Either type in the range, or use the drop down lists to select start and end times for

the data output. When you are satisfied with the range, click on Cancel or OK to start the output.

Spreadsheet format

It is sometimes more convenient to export data as text in a spreadsheet format. This does not attempt to export every data sample and event; instead it resamples the data at a user-defined rate to generate a rectangular table with columns for channels and rows for time. This is usually much simpler to interpret and import into other programs than the somewhat complicated format required to describe the raw data.

Format

The Format section of the dialog selects which items are output and how they are separated and delimited:

Field types, Separators and String delimiters

All time view Copy As Text information is written in fields. Each field is either numeric or text. Between each field there is a separator, which can be set to be a Tab character, a blank or a comma in the Separator drop down list. Most programs that accept tabulated numbers will accept space, a Tab or a comma. The examples below use space as a separator.

A numeric field holds numbers only, either floating point numbers with a decimal point, or integer numbers. A text field is a sequence of characters that may include spaces. Text fields may hold numbers, but numeric fields cannot hold text. You can mark the start and end of a text field with a special character (usually ") so that a program reading the field can include blanks (spaces) and punctuation within a field without confusion. All keywords are treated as text fields. You can set a one character delimiter in the String delimiter field, or leave it blank for none. The examples use " as a delimiter.

All output sections start with a keyword that can be used to identify them. All sections apart from the first are preceded by 1 or more blank lines.

Columns

You can choose to have the data for waveform channels and Evt+ and Evt- times written in multiple columns. This does not write one channel per column (use Copy Spreadsheet for that format). The standard format is to write one item per line for these data types. Before version [10.12], Evt+- channels were written in multi-column format, now they include the first marker code (level), and are written as one item per line. You can force the old format with an Edit Preferences option in the Compatibility tab.

Header

The first part of the output is a header that displays information about the data file. This is given after the keyword `INFORMATION`. The header consists of the file name followed by a minimum of 5 and up to 8 lines of the file comment followed by an empty line. The header block (if present) is always at least 8 lines long; trailing blank comments after the first five are skipped. Prior to version [10.12], only 5 lines were written; A `smrx` file allows up to 8 lines of comments.

```
"INFORMATION"
"demo.smrx"
"These five lines are always present and"
"contain"
"the file comment"
"for this"
"data."
"More lines, up to 8, can be displayed for .smrx files"
```

Summary

The summary section starts with the keyword `SUMMARY` followed by a summary of the channel information for each selected channel and ends with an empty line. From [10.12] we display the same information for each channel, though some of the columns may not apply to all channel types. The columns and types are:

Column	Chan#	Type	Title	Rate	Units	Scale	Offset	Points	Traces	Pre	Comment
Format	Text	Text	Text	Real	Text	Real	Real	Integer	Integer	Integer	Text

Example output

```
"SUMMARY"
"1"  "Waveform" "Sinewave"   100  "mx" 0.584 0.84 0  0 0  "Variable frequency sinewave"
"2"  "Evt-"     "Stimulus"   100  ""   1     0    0  1 0  "Stimulus pulse, once per cycle"
"3"  "Evt-"     "Events"     100  ""   1     0    0  1 0  "Created"
"4"  "TextMark" "Textmark"   1    ""   1     0    36 1 0  ""
"5"  "RealMark" "Gas||O2|N2" 30   "Pa" 1     0    3  1 0  ""
"6"  "TextMark" "Comments"   0.27 ""   1     0    40 1 0  ""
"7"  "WaveMark" "nw-1"       100  "mV" 1000  0    32 1 10 "Created 01/08/2019 from channel 1"
"8"  "WaveMark" "Tetrode"    100  "mV" 1000  0    32 4 10 ""
"31" "Marker"   "Keyboard"   1    ""   1     0    0  1 0  "Keyboard"
"v1" "RealWave" "Virtual"    100  ""   1     0    1  1 0  ""
"v2" "RealWave" "Virtual"    100  ""   1     0    1  1 0  ""
```

Chan#
This is formatted as text as some channels (virtual, memory and duplicate have channel identifiers that are not numeric: v1, m1, 2a

Type
This test field holds the channel type as one of the strings: Waveform, Evt-, Evt+, Evt+-, Marker, WaveMark, RealMark, TextMark and RealWave.

Title
The channel title as text. In the case of RealMark channels, this can be subdivided by vertical bars. The first section is the generic title, for use when no specific trace is intended. The following sections hold the title for each attached value. If a section is blank, the generic title is used.

Rate
For Waveform, RealWave and WaveMark channels this is the sampling rate of the waveform (per trace for WaveMark channels). For all other channel types it is the expected maximum event rate (averaged over several seconds) that was used by Spike2 to set the input buffering requirements. If this channel was not sampled by Spike2, this field could be meaningless.

Units
For Waveform, RealWave and WaveMark channels this holds the channel units. It is blank for other channel types. In the case of a RealMark channel, the units can be subdivided by vertical bars in the same manner as the title.

Scale, Offset
These are the values used to convert between 16-bit waveform samples and user units. For a RealWave channel these are the values Spike2 uses to convert the 32-bit floating point values to 16-bit integers (when required). See the description of channel scaling for details of how to use these factors.

Points
This field is the number of values attached to each RealMark, the maximum number of characters attached to each TextMark or the number of waveform points per trace attached to each WaveMark. Other channel types show this field as 0.

Traces
This is the number of waveform traces attached to each WaveMark channel (1, 2 or 4), otherwise is unused (and set to 1).

Pre
This is used with WaveMark data only and is set to the number of pre-trigger points per trace. This will be in the range 1 to the number of points per trace minus 1.

Comment
The channel comment.

Old Summary format
This is the format used before version [10.12]. You can force this format from 10.12 from the Edit Preferences dialog Compatibility tab. The information given for the channel varies with the channel type. The first three

fields are the same for all channels, being the channel number, the channel type and the channel title. The remaining fields are:

Waveform	Units, Ideal rate, Actual rate, Scale, Offset
RealWave	Units, Ideal rate, Actual rate, Scale, Offset
WaveMark	Units, Ideal rate, Actual rate, Scale, Offset, Points, PreTrig
Event	Predicted mean rate
Marker	No other information

Here is some typical output. The SUMMARY section is terminated by a blank line.

```
"SUMMARY"
"1" "Waveform" "Sinusoid" "Volts" 100.0 100.0 1.0 0.0
"2" "Evt-" "Synch" 5.0
"4" "WaveMark" "Shapes" "Volts" 100.0 100.0 1.0 0.0 30 5 1
"31" "Marker" "untitled"
```

Channel information

The Channel section of the Time view text output dialog holds controls to control the output based on the channels types. For each channel, the output starts with a channel information section that is introduced by the keyword CHANNEL, followed by the channel number. The channel number is a text field. If the Synopsis box that matches the channel type is checked, the channel data type, comment and title are output on the next three lines. The channel information block ends with a blank line.

```
"CHANNEL" "1"
"Waveform"
"Signal generator"
"Sinusoid"
```

The information that follows the channel information varies with the channel type and is only output if the Data field is checked for the channel type. If a channel type is derived from a Marker, you can choose to output the data as though the channel were a simpler type to reduce the quantity of information. For example, if you were only interested in the times of WaveMark data, you can copy it as though it were Event data.

Waveform

A channel containing waveform data has the channel information followed by the data. The data starts with the channel units and ideal sample rate. The next line contains the keyword START followed by the start time of the data in seconds and the time increment per data point, also in seconds. It is possible for gaps to occur in the waveform data. A gap is shown by a line with the word GAP followed by the start time of the new section and the time increment. The data then follows as a list of waveform values.

```
"CHANNEL" "1"
"Waveform"
"Signal generator"
"Sinusoid"

"Volts" 100.0000
"START" 0.00 0.01
-3.7208
-3.8356

"GAP" 4.23 0.01
-2.2901
-2.6075
-2.8858
```

Event

Event data comes in three types, rising edge, falling edge and both edge triggered (Evt-, Evt+ and Level). Since version [10.12], Level events are output in their own format (you can force the older format with an Edit menu Preferences option). The Evt- and Evt+ types are represented in the same way as a channel header followed by a blank line, then a list of times terminated by a blank line. The times are written with the number of columns set in the Format section of the dialog.

```
"CHANNEL" "2"
"Evt-"
"Synchronisation pulse from Signal generator"
"Synch"

0.2022
1.3209
   .
   .
```

Level (Evt+-)

In the case of an event channel triggered on both edges, the state before the first transition is shown at the start of the data, by one of the words `HIGH` or `LOW`. `HIGH` means that the first time in the list represents a code 1 to 0 transition, `LOW` means the reverse. If there are no times in the list, these labels are the state in the period (`LOW` for code 0, `HIGH` for code 1). For each event, the time is followed by a number (usually 1 or 0). 0 means the data was low at this time, any other value (which will be 1 for data sampled by Spike2) means the data was high at the time. Prior to version [10.12] this was part of the Event format and there was no number following the time to indicate the level.

```
"CHANNEL" "2"
"Evt+-"
"Synchronisation pulse from Signal generator"
"Synch"

"LOW"
0.2022 1
0.2138 0
   .
   .
```

Note that in a modern 64-bit `smrx` data file, Level data is stored as a Marker. Each event has a time and 4 marker codes. The displayed value is the first marker code. In the old 32-bit format, Level data was stored as a list of event times. Each stored data block had a marker to indicate if the first event time in the block was high or low and the state was assumed to alternate.

Marker

Marker data consists of a series of times and 4 bytes of marker information. The channel synopsis is as for other channels, and the data is displayed as a time followed by a string of 4 characters which represent the 4 bytes of information. If the bytes are non-printing characters a ? is shown instead. The 4 bytes are also displayed as numbers after the string. You can force marker data to be dumped as event data or as level data.

```
"CHANNEL" "16"
"Marker"
"The comment for this channel"
"Title"

1.0906 "p???" 112 0 0 0
3.0336 "a???"  97 0 0 0
5.9802 "u???" 117 0 0 0
6.9018 "l???" 108 0 0 0
```

RealMark

The first line of RealMark data output holds the channel units, the expected mean RealMark rate and the number of real values attached to each event. This is followed by lines holding the same data as for a Marker followed by the list of RealMark values. You can force RealMark data to be dumped as marker, level or event data.

```
"CHANNEL" "5"
"RealMark"
"This is the RealMark channel comment"
"Memory"

"Units" 30 2
1.0906 "????" 112 0 0 0 1.8923 4.8567
1.8603 "????" 113 0 0 0 1.8224 4.123
```

TextMark

The first line holds the channel units (usually an empty string as TextMark channels have no units), the expected mean rate and the maximum number of characters allowed per item in this channel. The following lines hold data for each TextMark in the time range. The line starts with the same format as for Marker data, followed by the text string. You can force TextMark data to be dumped as marker, level or event data.

```
"CHANNEL" "4"
"TextMark"
"Comment for TextMark channel"
"Title"

"" 1 100
0.5234 "????" 112 0 0 0 "This is where I added the secret ingredient"
9.8603 "????" 113 0 0 0 "Control point 1"
```

You can also export this data type by double clicking any TextMark and copying from the TextMark dialog.

WaveMark

WaveMark data is structured as a marker plus a short section of waveform data. You can force a WaveMark channel to be dumped as though it were a marker, level, event or waveform channel. When displayed as WaveMark data, the channel synopsis is the same as for waveform data, but also has the number of waveform points associated with each event and the number of pre-trigger points. The dump of data starts with a line holding the channel units, the ideal sampling rate, the number of data points, the number of pre-trigger points and from [10.12] onwards, the number of traces:

Units, Ideal rate, Points, PreTrig{, nTrace}

This is followed by blocks of data, one for each WaveMark event in the time range. The first line of the block holds the keyword WAVMK, followed by the time of the first data point in the event, the time interval between waveform points and the four marker bytes. This is followed by the WaveMark data values and a blank line:

WAVMK,eventTime,adcInt,mark0,mark1,mark2,mark3
data values (1 column per trace)
blank line

This block is repeated for each event in the time range. A typical WaveMark channel begins:

```
"CHANNEL" "1"
"WaveMark"
"Channel comment for demonstration"
"Title"

"mV" 100.0 30 5 1
"WAVMK" 0.0210 0.00005 1 0 0 0
-0.03098
0.03189
0.09430
.
.

"mV" 0.8110 0.00005 3 0 0 0
0.15610
0.21713
0.27710
.
.
```

Multi-trace data

Prior to [10.12] a channel with `n` traces and `p` points per trace was displayed as all `p` points of the first trace followed by all `p` points of the second, and so on. Each trace started on a new line (if the columns were set to more than 1). Now, the data is displayed as `p` lines of data with `n` columns per line, one per trace:

```
"CHANNEL"      "8"
"WaveMark"
"Comment goes here"
"Tetrode"
```

```
"Units" 30 32 10 4
"WAVMK" 0.00000 0.01000 0 0 0 0
-0.03098 -0.64331 -1.75903 -1.75903
0.03189  -0.67657 -2.24762 -2.24762
0.09430  -0.70694 -2.41455 -2.41455
.
.
```

Paste

You can paste the text on the clipboard into a text, script or output sequence document. When you paste text, Spike2 checks that each pasted line ends with the correct end of line characters (for example, Macintosh and Windows use different sequences). The paste operation corrects incorrect sequences. To correct text that has come from a different operating system and that looks strange, select all the text, cut, then paste.

You can visualise the end of line characters in a Spike2 text window by opening the appropriate text settings window with the View menu Font command and check Line ends.

Delete

This command is used to delete the current text selection, or if there is no selection, it deletes the character to the right of the text caret. Do not confuse this with Clear, which in a text field is the equivalent of Select All followed by Delete.

Clear

When you are working with editable text, this command will delete it all. Clear removes everything; Delete removes the current selection.

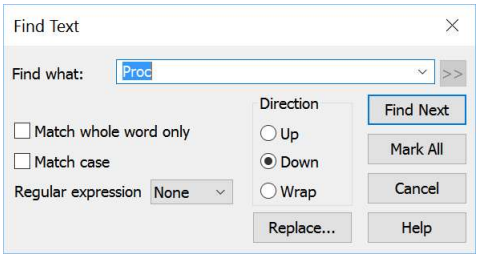
If you are in a result window, this command will set all the bins to zero and redraw the window contents.

In an XY view, this command removes all data points, leaving all channels empty and redraws the window.

Select All

This command is available in all text-based windows and selects all the text, usually in preparation for a copy to the Clipboard command. The short-cut keyboard command is `Ctrl+A`.

Find, Find Again, Find Last



The Edit menu Find command opens the Find Text dialog. If there is a single-line selection in the text window, this sets the initial contents of the Find what field. The dialog is shared between all text-based views. It is closely linked to the Find and Replace Text dialog and shares all its fields with it; opening this dialog closes the Find and Replace dialog. Click Replace... to swap to the Find and Replace dialog. There is also an Edit Toolbar with buttons (and short-cut keys) to control bookmarks and searches.

A successful search moves the text selection to the next matching string. Searches are done line by line; you cannot search for text that spans more than one line. The `InStrRE()` script command can search across line ends in a text string.

Find Next starts a search for the text in the Find what field. The Mark All button sets a bookmark on all lines that match the search text, but does not move the selection. Searches are insensitive to the case of characters,

unless you check Match Case. Check the Match whole word only box to restrict the search so that the first search character must be the start of a word and the last search character must be the end of a word.

The script language equivalent of this dialog is the `EditFind()` command.

Search direction

Select `Up` to search backwards, towards the start of the text. Select `Down` to search forwards towards the end of the text. Select `Wrap` to search forwards to the end, then wrap around to the start and stop when you reach the current position. Searches do not include the currently selected text.

Regular expression

Regular expressions provide a powerful syntax for searching text for patterns. You can set this field to three value:

None — No use of regular expressions. Searches must be a match for what you type. You can choose to match whole words only. The meaning of a word depends on the type of the file that is being edited, but is usually just what you expect.

Simple — This uses the simple regular expression syntax that has always been supported by the Spike2 text editor.

ECMA — This used the more complex ECMAScript regular expression syntax that is supported in this dialog from version 9 onwards.

If you select either regular expression grammar, this disables Match whole word only as regular expressions have their own way to match word starts and word ends. It enables the >> button, which displays a list of regular expressions to insert into the search string; the regular expressions listed depend on the grammar.

Find Again, Find Last

The Edit menu Find Again and Find Last commands repeat the current search forwards or backwards.

Simple regular expressions

The text editor has always supported a simple (but non-standard) regular expression syntax that has fewer features than the ECMAScript regular expressions that can also be used. This topic describes the simple regular expression syntax, it is similar to the Basic syntax available in the `InStrRE()` script command.

The simplest pattern matching characters are:

- `^` Start-of-line marker. This character must be at the start of the search text. The following search text will only be matched if it is found at the start of a line.
- `$` End-of-line marker. This must occur at the end of the search text. The preceding text will only be matched if it is found at the end of a line.
- `.` Matches any character.

To treat these special characters as normal characters with regular expressions enabled, put a backslash before them. A search for "`^\^.\.`" would find all lines with a "^" as the first character, anything as a second character and a period as the third character.

You can use `\a`, `\b`, `\f`, `\n`, `\r`, `\t`, `\v` to match the ASCII characters BELL, BS (backspace), FF (form feed), LF (line feed), CR (carriage return), TAB and VT (vertical tab). Searching for `\n` and `\r` will not normally match anything as `\n` or `\r\n` mark line ends and the search is of complete lines ignoring end of line markers.

To search for one of a list of alternative characters, enclose the list in square brackets, for example `[aeiou]` will find any vowel. For a character range use a hyphen to link the start and end of the range. For example, `[a-zA-Z0-9]` matches any alphanumeric character. To include the - character in a search, place it first or last. To include `]` in the list, place it first. To search for any character that is not in a list, place a `^` as the first character. For example, `[^aeiou]` finds any non-vowel character.

There are search characters that control how many times to find a particular character. These characters follow the character to search for:

* Match 0 or more of the previous character. So `51*2` matches `52`, `512`, `5112`, `51112` and `h.*l` matches `hl`, `hel`, `hail` and `B[aeiou]*r` matches `Br`, `Bear` and `Beer`.
+ Match 1 or more. The same as "*", but there must be at least one matching character.
? Match 0 or 1 of previous character. So `51?2` matches `512` and `52`; `Bee?t` matches `Bet` and `Beet`.

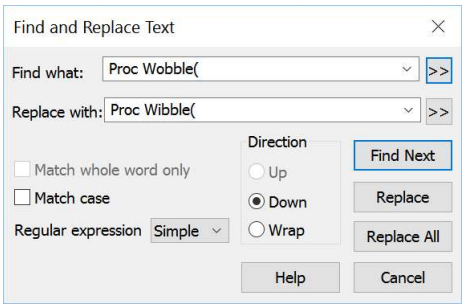
Put a backslash before these characters to treat them as normal characters in a regular expression.

You can search for the start and end of a word with `\<` and `\>`. Word characters are the set `[a-zA-Z0-9]`, or `[a-zA-Z0-9%$]` in script views. For example, the regular expression `\<[a-zA-Z]+\>` will match a text word, but not if it contains numbers. You can also use `\w` to match a word character and `\W` to match not a non-word character. Likewise, `\d` matches a decimal number and `\D` to matches a non-number character and `\s` matches white space (space, TAB, FF, LF and VT) and `\S` matches non-white space. You can use `\w`, `\W`, `\d`, `\D`, `\s` and `\S` both inside and outside square brackets.

You can match a character with a particular code using `\xnn`, where `nn` is the character code in hexadecimal.

You can also tag sections of matched text by wrapping it in `\(` and `\)`. You can then insert the tagged text later in the regular expression (or in the replace text in the Find and Replace dialog) using `\n`, where `n` is 1 for the first remembered text, up to `9` for the ninth. For example, `\(foo\)-\1` matches `foo-foo`. More interestingly, the regular expression `\(\<[a-zA-Z]+\>\)-\1` matches `Jim-Jim`, `plum-plum` and the like.

Replace



The Edit menu Replace command is available when the current view is text-based. It opens the Find and Replace Text dialog in which you can search for text matching a pattern and optionally replace it. If there is a single-line selection in the text window, this sets the initial contents of the Find what field. The search part of the dialog is identical to the Find Text dialog. The search pattern set by the `Find what` field can be a simple match, or can be a regular expression. In regular expression searches, the replacement text can refer back to tagged matches in the search text. See the Find dialog for details of regular expressions and tagged matches. The >> buttons are also enabled in regular expression mode and let you insert expressions into the search and replace text.

The script language equivalent of this dialog is the `EditReplace()` command.

Replace with

This field holds the text to replace the matched search text. In a regular expression search, you can include tagged matches from the search text using `\1` to `\9` as described for the Find dialog. For example, suppose you have variables named `fred0` to `fred17` that you want to convert into an array `fred[0]` to `fred[17]`. You can do this by setting the Find what field to `\<fred\([0-9]+\)\>` and the Replace with field to `fred[\1]`.

Replace

The Replace button checks that the current selection matches the Find what field and if it does, the field is replaced by the Replace with text field and the selection is moved to the next match. If the current selection does not match, Replace is equivalent to Find Next.

Replace All

This button searches for all matches in a forward direction starting from the beginning of the text to the end, and replaces them.

Edit toolbar

The Edit toolbar gives you access to the edit window bookmarks and short cuts to the Find, Find next, Find previous and Replace commands. If you are unsure of the action of a button, move the mouse pointer over the button and leave it for a second or so; a "Tool tip" will reveal the button function and any short-cut key associated with it.

The four bookmark functions toggle a bookmark on the current line, go to the next or previous bookmark and clear all bookmarks. The Edit menu Find command can set bookmarks on all lines containing a search string.

If the Edit toolbar is not visible, right-click on an empty area of a toolbar or of the Spike2 main window. Then select Edit bar in the pop-up menu. You can use the same procedure to hide it. The bar can be docked to any side of the application main window or dragged off the application main window as a floating bar.

The short-cut keys are:

`Ctrl+F2`	Toggle bookmark on the current line
`F2`	Go to next bookmark
`Shift+F2`	Go to previous bookmark
`Alt+F2`	Clear all bookmarks
`Ctrl+F`	Open the Search dialog
`Ctrl+G`	Find next
`Ctrl+Shift+G`	Find previous
`Ctrl+H`	Replace text

Auto Format

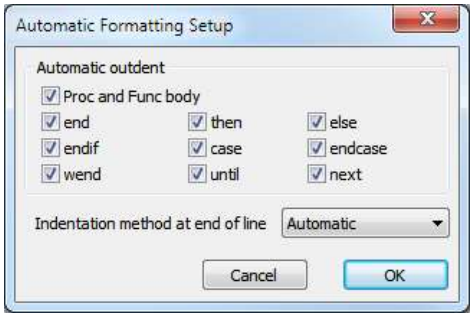
Automatic formatting is available for script views and applies standard formatting while you type and lets you reformat entire scripts or selected regions. Formatting is done by indenting lines of text based on the script keywords, so it depends on the script making syntactic sense. An indented line is one that starts with white space. The indenting is in units of the Tab size set for the view in the Script Editor Settings dialog. Indentation is done with Tab characters if you have chosen to keep Tabs in the view, otherwise indentation is done with space characters. There are two sub-commands:

Apply Formatting

If any text is selected in the current view, all lines included in the selection are formatted. If there is no selection, the entire document is formatted. If text is selected, you can right-click in the text view and choose Auto Format Selection from the pop-up menu to activate this option.

Settings...

The Automatic Formatting Setup dialog controls how text is formatted. Formatting is based on the same scheme that is used for folding text. Each script keyword that starts a block construction (`proc`, `func`, `if`, `docase`, `while`, `repeat`, `for`) increases the indent, and the keywords that complete blocks decrease the indent.



The standard CED formatting is to have all the boxes checked, however, it makes no difference to the script operation so, what you choose is a matter of personal taste. It is a good idea to use consistent indenting as it helps you to understand the structure of the script.

Proc and Func body

If this box is not checked, all text within a function or procedure is indented. Check the box to outdent the text between the `Proc` or `Func` and the `end`.

end...next

If we modelled the indentation on the folding with no exceptions, the keyword that marks the end of a block would be indented. Some people like this style, others do not. If you would prefer the keyword that ends a block to align with the keyword that starts the block, check the box. You can choose to outdent any line that starts with one of the keywords `end`, `then`, `else`, `endif`, `case`, `endcase`, `wend`, `until` and `next`.

Indentation method at end of line

This field determines what happens to the indentation of the current line and the new line when you type the `Enter` key. The settings are:

None — No automatic formatting is applied. The new line is not indented.

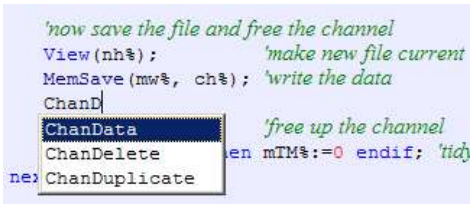
Maintain — The new line is indented to match the indentation of the current line.

Automatic — The indentation of both current and new lines is adjusted based on the Auto Format settings.

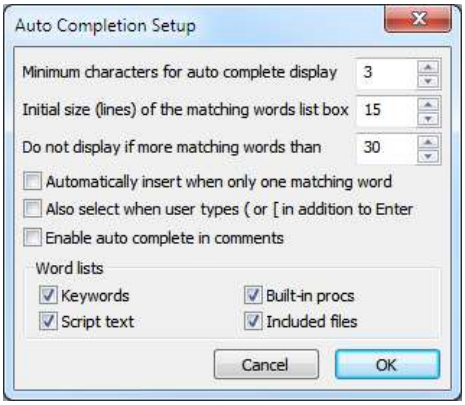
Toggle Comments

This Edit menu command is available in script and text sequencer views, and is also available in the context menu when there is a selection. It adds or removes a comment marker at the start of the line for all lines in the current selection. It decides what to based on the first character of the first line in the selection, so this will not work for indented comments.

Auto Complete



In any script, you will find that you are typing the same text items repeatedly. The editor can save you some time by popping up lists of known words that match your typing. The matching is done by looking for a word break in the text before the text caret, then matching your typing against various categories of words known to the script. See Word lists, below, for the categories that can be matched. Matched words are displayed in alphabetical order and the current word is highlighted.



You can use the up and down arrow keys to choose an item in the list and the `Enter` key to select an item or double click an item to enter it. The `Esc` key cancels the list. Alternatively, you can just keep on typing, which will narrow the choice of words to match. The Edit menu Auto Complete Setup dialog gives you some control over the words that are matched and when the auto-completion lists appears.

Word lists

The Word lists section of the dialog lets you choose the categories of words to match your typing against. If you do not want to display auto-completion lists, clear all the categories.

You can choose from script keywords (`Proc`, `Func`, `EndCase`, ...), built-in functions and procedures (`App`, `NextTime`, ...), words that already exist in the script and user-defined `Func` and `Proc` names in included files and global symbols in included files.

You may wish to disable the Script text option if you are working with a huge file and you notice an appreciable delay after typing before the list appears. Similarly, you may wish to clear the Included files option if you use a large number of included files and there is a noticeable time lag before any list appears.

Minimum characters for auto complete display

This sets the number of characters in a name that must be typed before the list will appear. You can set this to 1 to 12 characters. Setting a low value can cause the pop-up display to become annoying. You need to choose a count that gives you enough help, but not too much. Try a value of 3 to start with.

Initial size (lines) of the matching words list box

This field sets the maximum number of words to display at a time in the range 1 to 40. If there are more matching words, the list contains a scroll bar. After the list appears, you can re-size it by clicking and dragging on the horizontal edge furthest away from the matched text.

Do not display if more matching words than
If there are more matching words than you set here, the list is not displayed. You can set this value in the range 1 to 200 words.

Automatically insert when only one matching word
Because of the design of the script language, apart from variable declarations, all words you type in are likely to have been defined already. If you set this option, once your typing has reduced the list size to 1, the word is automatically inserted. You may want to increase the Minimum characters for auto complete display if you enable this option.

Also select when user types (or [
Normally, the list text is inserted when you type the `Enter` key or double-click the list text. If you check this option, the currently selected matching text is inserted when you type an opening round or square bracket, followed by the bracket you typed.

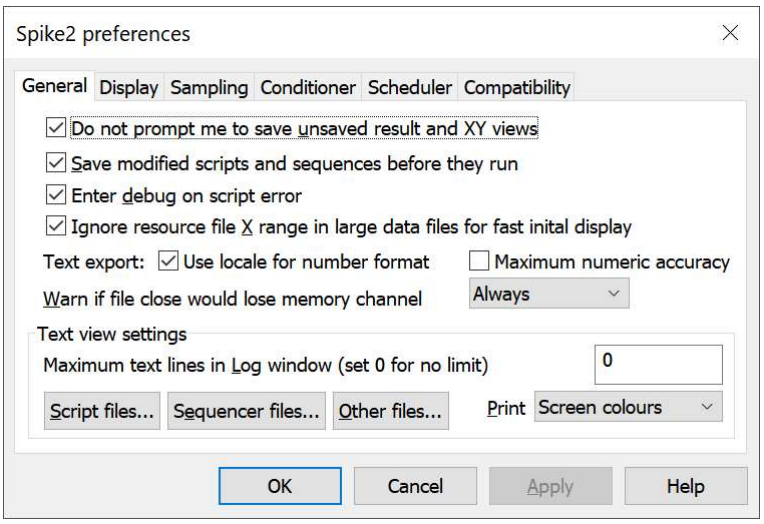
Enable auto complete in comments
Normally, automatic word completion is disabled in a comment. However, if you need to refer to script functions in your code documentation you may find it useful to check this box.

Preferences

The Edit menu Preferences dialog has tabs for general settings, display and sampling options, signal conditioner, time scheduling and for compatibility with previous Spike2 versions. Preferences are stored in the Windows registry and are user specific. If you have several different logins set for your computer, each has its own preferences. You can use the `Profile()` command to change the preferences from a script.

General

This tab holds editor preferences and general settings for saving modified views and to control what happens when an error is found in a running script.



Do not prompt me to save unsaved result and XY views
As it is often possible to recreate result and XY from the raw data, you can check this box to suppress the normal Windows behaviour of prompting you to save unsaved data. This only applies to interactive closing when the view has never been saved (so has no associated disk file). The script `FileClose()` command has a flag for prompting the user to save unsaved data and pays no attention to this setting

Save modified scripts and sequences before they run
If you check this box and run a modified script or sequence, it will be saved first. If the script has no name, the File Save dialog opens to prompt you for a file name.

Enter debug on script error
Normally, if a running script has a problem, it stops with an error message in the script window and Log view. If you check this box, the script debugger is activated on a script error and you can inspect the local and global errors and the call stack at the time of the error. You are not allowed to continue running the script.

Ignore resource file X range in large data files for fast initial display
Normally, when Spike2 opens a data file with an associated resource file, it shows the same x range as was visible when the file closed. If the file is very large this initial display can take several seconds. If you check this option, when you open a data file that is more than 50 MB in size, Spike2 displays the first second of data and not the range displayed when the file closed.

Text export: Use locale for number format
Normally, Spike2 uses standard compatibility options for all text output. If you check this option the number formatting settings defined in the operating system are used for data exported to the clipboard or file as text. The main effect is that if your operating system is set to use a comma character as the decimal point, you can now export numbers using this. This has no effect on data input and no effect on the script language. The operating system settings that are associated with your country or language are collectively referred to as the "locale".

Text export: Maximum numeric accuracy
Normally, when copying Time, Result and XY view data as text to the clipboard or file, Spike2 uses sufficient decimal places to give a reasonable representation of a floating point value. Check this box to force full accuracy. This may cause numbers you expect to see as 0.1 appear as 0.09999999999999998, however it means that no precision is lost if you transfer data as text to another program. Prior to [10.20] this option was not applied to Time view data. In a Time view, you may get better results for RealWave channels by adjusting the channel Scale factor.

Warn if file close would lose memory channel data [8.02]
Memory buffer channels only exist while the data file is open and are lost when you close the file. In many cases you will not care about this, however, if you have put considerable effort into creating a memory buffer you may wish to be warned that you are about to lose one. You are not warned if there are memory channels, but they contain no data. There are three warning levels:

Never	You will never be warned. This was the state before Spike2 version 8.02.
Not from script	You are warned unless the file is closed with the script `FileClose()` command (regardless of the state of `query%`). This is the default warning level.
Always	You are always warned unless the file is closed with the script `FileClose()` command and `query%` is set to -1.

Text view settings

These controls configure text-based views (script, sequencer, log and script-created text views) on screen and when printing. The dialogs accessed by the Script, Sequencer and Other files buttons apply changes globally (to all open files of the selected type and to all future files). You can use the same dialogs from the View menu Font commands to change the current view (there is a button to apply changes to all views).

Maximum text lines in Log window
This field prevents huge amounts of text accumulating in the Log view. If there are more lines than set here in the Log view and a script writes more, the oldest lines are deleted to leave this number. To avoid deleting lines every time a line is added (which is very slow), each time the line count exceeds the number of lines set here, the excess lines plus 10% of the maximum lines are deleted so that the view can grow normally for a while. Set 0 for no line number limit. Lines are not deleted as a result of interactive typing in the window. The script language equivalent is the `ViewMaxLines()` command.

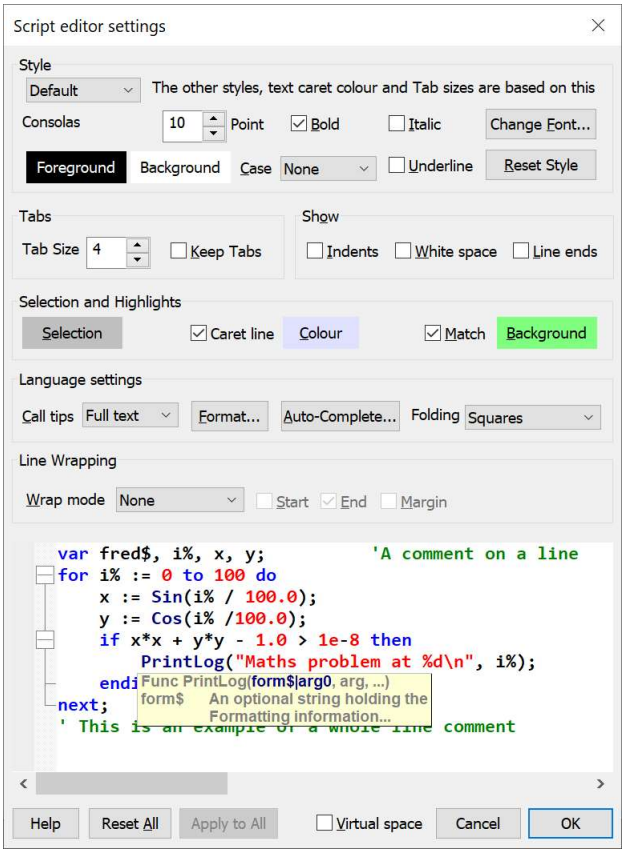
Print

This sets how to print coloured text from a text view. The choices are:

Screen colours	Use the displayed screen colours; this uses a lot of ink if the background is not white.
Invert light	If you display your text as a light colour on a dark background, this setting prints on a white background and inverts the text colours.
Black on White	Prints black text on a white background.
Colour on White	Prints in screen colours on a white background.

Script file text settings

Open the editor settings dialog from Edit menu Preferences General tab to change settings for all views of the current type, or from the View menu Font command for the current view; you can use Apply to All to apply changes to all views. The Reset All button returns all values to standard settings. The bottom half of the dialog holds example text to preview any changes.



Please note that most of the items in this dialog are not script-controllable. However, a script can set the features covered by the Style (see `FontSet()` and `FontGet()`).

Style

Each view type supports one or more text styles. For each style you can set a font (including proportionally spaced fonts). The font includes the size in points (in the range 2-256) and bold and italic settings. You can also choose to force upper or lower case and underlines for all text in the style. You can set a foreground and background colour for the style by clicking on the Foreground and Background rectangles.

In a Script view, you can control the appearance of many different aspects of the display, based on the syntax of the language. There are settings for:

#	Name	Usage
32	Default	The basis of all other styles and sets the text caret colour. The Tab size is based on the width of a space in this style. If you change any aspect of this style, all other styles that match that aspect will also change. The background colour sets the view background colour.
0	White space	Spaces, tabs, control characters and anything not covered elsewhere.
1	Comment	The style used when displaying comments.
2	Numbers	The style to use when displaying numbers.
3	Keywords	The style to use for script keywords, such as `for`, `next` and `end`.
4	String	The style for literal strings, such as `"This is a string"`.
5	Function	Used for built-in script functions, such as `PrintLog()`.
6	Operator	The style for script language operators, such as +, - and +=.
7	Identifier	The style for function, procedure and variable names created in a script.
8	Preprocessor	`#include` and any future items introduced by `#`.
33	Line numbers	The style used for line numbers and the gutter. When you select this item, line numbers will appear in the example text area.
38	Call tips	The style for pop-up call tips. This style has an extra Tip highlight colour field, used to highlight the current argument in a function. This replaces the Bold, Italic, Case and

	Underline fields, which are not displayed.
34 Braces	The style for matched square and round brackets `()` and `[]`. When the text cursor is next to a bracket, the editor searches for the matching bracket and applies this style if it is located.
35 Bad braces	The style to use when the text cursor is next to an unmatched bracket.

The # column is the `style%` argument value for `FontSet()` and `FontGet()`.

Reset Style and Reset All

The Reset Style button reverts the current style to standard settings. Reset All reverts all styles.

Tabs

This dialog region controls the size of tabs (set in units of the width of a space character in the Default style) and if Tabs are implemented by saving a Tab character in the text (check Keep Tabs) or are implemented as multiple spaces (clear the Keep Tabs check box). If you use a fixed pitch font, such as `Courier New`, then it does not matter too much if you choose to keep Tab characters or not. If you use a proportional font for anything except comments, it is better to keep the Tab characters. When automatically formatting a script, the Keep Tabs setting determines if indents are generated with Tab characters or spaces.

Show

In addition to displaying the text, you can also choose to display information about the white space in your file. The settings are:

Indents	It can be useful when manually lining up indented loops in a script to see the indent level. Check this box to display vertical lines at each tap stop in the leading white space of each line.
White space	Check this box to display spaces with a centred dot and tabs as right arrows.
Line ends	Check this box to see the Carriage Return (`CR`) and Line Feed (`LF`) characters that mark the end of a line. This is useful when working with scripts that move the caret around.

Selection and Highlights

You can modify the background colour used to indicate the selected text and optionally to indicate the line holding the caret and to mark complete 'words' that match the current selection. You can always modify the selection colour. This was added at Spike2 version [10.10].

Selection

You select text by holding down the left (usually) mouse button and dragging or by double-clicking to select a word or triple-clicking to select the current line. The selection in indicated by a change of the background colour. Before version [10.19] the background colour changed to a dark grey when the window was not active.

Caret line

You can highlight the line holding the text caret by checking the Caret line box and changing the background to a colour of your choice. This will usually be set close to the background colour to avoid being too visually disruptive. The colour you set is drawn transparently at around 40% intensity; this allows any background (such as an error marker in a script view) set in the line to be visible. The highlight is removed when the view is not active.

Match

Check the Match box to highlight complete words that match the text selection. For example, if you double-click a variable name in a script to select it, this will highlight all visible uses of the variable (including inside quoted strings). If the selection includes non-word characters, no highlighting occurs. The Background colour you set is mixed with the normal colour of the matching text at 40% intensity. If you set the same colour as for the selection you will get a result that is similar to the selected word, but with a less intense background.

Language settings

This area of the dialog is used by script and output sequencer views. It sets your preferences for call tips, automatic formatting, automatic text completion and folding. The Format and Auto Complete buttons duplicate Edit menu commands, and are described separately.

Line wrapping

This section was added as an experimental feature at Spike2 version [10.18]. Text files used by Spike2 for the script and the output sequencer are not expected to have much use for this feature, but we have had occasional requests to make it easier to cope with external text lines with very long lines. Note that script commands such as `MoveTo()` and `MoveBy()` will continue to operate in terms of lines and characters and ignore the wrapping mode. Text layout of large documents with long lines can be noticeably slower with wrapping enabled.

Wrap mode

This control determines if wrapping is enabled, and how it is done. There are four settings:

None — The standard mode. There is no text wrap and long lines end at the edge of the display area. There is a horizontal scroll bar to navigate to the end of long lines.

Word — If a line is too long to fit, it is split at the nearest word or style break (as defined for the current view type) to the end of the available space. If there is no suitable break, the line splits at the last character that fits.

Character — The line splits at the last character that fits in the available space. This can be useful for languages that do not use white space to separate words.

White space — The line splits at the nearest white space to the end of the available space. If there is no white space, the line splits at the last character that fits.

Wrap marker display

You can display an indicator to show that a line is wrapped. You can chose a combination of three places to mark wrapped lines:

Start — Displays a wrap mark at the start of each continuation line. This offsets the text slightly to the right.

End — Displays a wrap mark at the end of each line that is wrapped.

Margin — This is only of use if you display the line number margin. A wrap symbol appears under the line number on each wrapped line.

Folding

Script views and output sequence views can display a folding margin, and allow you to fold the code by clicking in the margin, from the View menu Folding command, and by right clicking in the view and selecting Toggle all folds. In a script view, fold points are based on a lexical analysis of the script text. You can choose from one of four folding styles, or have no folding margin.

Call tips

A Call tip is a block of text that pops up in a script view when you type the opening brace of a function or procedure name or you hover the mouse pointer over a procedure or function name. You can choose from None (no tips), Single line (the command name and arguments) and Full text (includes a command synopsis). If you select the Call Tip style, you can set the text and background colour of the tip and also the colour of the highlight used to emphasise the current argument.

Virtual space

Check this box to enable virtual space in the editor. This allows you to move the text caret beyond the end of a line by clicking with the mouse or by using cursor keys; you can position text (for example comments) without the need to use tabs or spaces to position the caret. If you type with the caret in virtual space, the editor inserts spaces up to the caret position and the position becomes "real". However, allowing virtual space may force you to use extra key presses to move to the real start or end of a line (`Home`/`End` keys) when you press cursor up and down to move vertically through the text.

Change History

This check box is visible when this dialog opens from the Edit menu Preferences. If you check the box the script editor will display marks in the margin to indicate changes to the code. This affects edit windows that are opened after this change is made; it has no effect on script windows that are already open. This was added as an experimental feature at version [10.16]. The marks in the margin are:

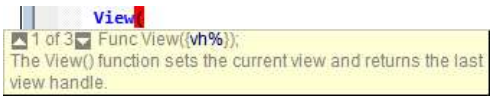
Orange box	Modified. Marks edits to code that have not been written to the file.
Green rectangle	Saved. Edits that have been written to the file.
Green-yellow	Saved changes that have been reverted (undone).
Cyan rectangle	Saved, then reverted to a previous modified state.

Anything other than a green rectangle marks text that differs from the saved state. Note that the change tracking uses the Undo system to track changes between the displayed text and the last saved state. If you manipulate the view text with a script, this disables the Undo system (on the assumption that script-based changes are permanent), so change tracking will not work.

Apply to All

This button shares space with the Change History check box and is displayed when you open this dialog with the Font button or Menu command. If you click this button, the current state of the dialog will be applied to all open and future Script editor windows.

Call tip details



For function and procedures, the call tip text contains the command name and arguments and a synopsis of the command. If the tip appears as a result of typing the opening brace, the editor attempts to highlight the current argument in the tip and if the command has multiple variants, you can select the variant to display by clicking on the arrows in the call tip. In Single line mode (not when opened by hovering the mouse pointer) you can click on the body of the tip to show the full text.

Call tips for built-in functions

For built-in functions, the call tip text holds the command name and arguments plus one sentence of description. This information comes from the text file `script.tip` in the folder where Spike2 is installed.

Call tips for user-defined functions

For user-defined `Proc` and `Func` items, the arguments are taken from the line containing the `Proc` or `Func` keyword (expected to be the first item on the line). If the line before the `Proc` or `Func` is a comment, up to 10 lines of comment are used as a description. This works for `Proc` and `Func` items in the current source and in any included files. If the line before is blank, and the line after is a comment, up to 10 lines following are used as a description. Putting the comments after the `Proc` or `Func` line has the advantage that the comments can be folded within the `Proc` or `Func`.

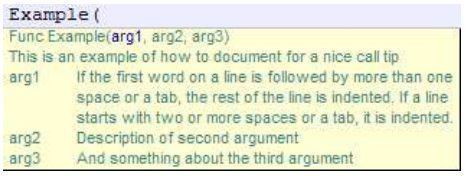
The search is terminated by any "divider" lines. A divider line is one that is at least 10 characters long (after removing the initial comment mark ') and with no more than 2 characters being different. Here are some example divider lines:

```
''''''''''''''''''''''''''''''''''''''''''''''''''
'--------------------------------------------------
'|================================================|
```

If you document your functions and procedures as in this example, they will generate tidy call tips:

```
'This is an example of how to document for a nice call tip
'arg1  If the first word on a line is followed by more than one
'      space or a tab, the rest of the line is indented. If a line
'      starts with two or more spaces or a tab, it is indented.
'arg2  Description of second argument
'arg3  And something about the third argument
Func Example(arg1, arg2, arg3)
var x,y;  'the start of the code
...
```

The example text above would produce a call tip looking like this. The comment markers at the start of each line are removed, and the multiple spaces after the first word or at the start of each line are replaced with a Tab character.



If you do not want a `Proc` or `Func` to have a call tip see the section, below, Disable call tip for an item. If you disable a call tip in an include file, this will also stop it appearing in an Auto-completion list.

Call tips for const and var names [8.03]

We also provide call tips for names defined in `const` and `var` statements and argument names defined in the first line of a `Proc` or `Func` statement. The tip will be the entire text of the line that includes the definition. Any tab characters in the line are replaced by space characters, then runs of space characters are replaced by a single space to make the tip a short as possible.

Call tips for included files and Auto Complete

You can have tips for global symbols for items in included files. You must enable this in the Edit menu Auto Complete command by checking Included files. Call tips and *Auto Complete* use the same internal logic

Go to declaration

If an item has a call tip, it also will have a right-click menu option. For user-defined items, this will let you `Go to` the declaration of the item. For built-in items, this will open Help for the item (as long as the help file is available).

Limitations

Call tips and Auto Complete have to work on scripts that are works in progress and that may not be syntactically correct (i.e. they may not compile). This means that we cannot use the same syntax analysis engine that the script compiler uses. To generate a useful result, we make the following assumptions when parsing scripts for user-defined `Proc`, `Func`, `const` and `var` statements:

- The `Func`, `Proc`, `const` or `var` keyword is the first non-white space symbol in a line.
- The `end` keyword that terminates each `Func` or `Proc` is the first non-white space symbol in a line.
- For symbol names to be detected, they must be on the same line as the `Func`, `Proc`, `const` or `var` keyword that introduces them.

It is important that the `end` keyword is recognised as we use this to skip over user-defined functions when searching for a global symbol.

Disable call tip for an item

Although having call tips is generally very useful, you may sometimes want to hide user-defined `Func`, `Proc`, `var` or `const` names from the call tip and `Go to` systems. For example, you might have an include file that implements some useful functionality through a single user-defined `Func`, but that has several helper functions and global `var` and `const` items that you do not want to be generally visible as they just cause clutter. There are two ways to do this. The simplest is to start a comment on the same line with an exclamation mark.

```
const kPrivateConst := 1.234; '! No call-tip or autocomplete
```

The second way is to separate the keyword (`Proc`, `Func`, `const`, `var`) from the symbol names that it introduces. However, it is possible that we will further develop the parser so it does not rely on text being on a single line, so this method may not work forever. We mention this as this method was documented in older versions of Spike2.

```
Proc
PrivateFunction() 'This is only used within this file
...
end;
```

Sequencer file text settings

The editor settings dialog for output sequencer files is very similar to that for script files. The example text in the lower half of the dialog displays some typical sequencer code, and there is no support for Call tips, auto formatting or automatic completion.

Style

There is a list of styles that you can apply to the different elements of the sequencer text. The Default style is used in exactly the same was as for script views as The remaining styles are:

#	Name	Usage
32	Default	The basis for all other styles and as the font that is measured to set the Tab size and sets the view background colour.
0	White space	Style used when drawing spaces, tabs and control characters (normally the same settings as Default).
1	Comment	The style used when displaying comments.
2	Number	The style to use when displaying numbers.
3	Keyword	The style to use for standard output sequencer instructions.
4	Display	Text introduced by > for display on screen during sampling.
5	Directive	Items such as `SET` and `VAR` that do not generate output instructions.
6	Operator	The style for mathematical operators like `+`, `-`, `*` and `/`.
7	Identifier	The style for user-defined variable names and labels.
8	Step	The style for the optional step numbers at the start of an instruction line.
9	Key	The style for keyboard links introduced by a single quote, for example `'H`
10	Functions	The style for built-in conversion functions like `ms()`.
11	Deprecated	Items that work, but that we suggest you no longer use, such as `CRATE`.
33	Line numbers	The style used for line numbers and the gutter. When you select this item, line numbers will appear in the example text area.
34	Braces	The style for matched square and round brackets `()` and `[]`. When the text cursor is next to a bracket, the editor searches for the matching bracket and applies this style if it is located.
35	Bad braces	The style to use when the text cursor is next to an unmatched bracket.

The # column is the `style%` argument value for `FontSet()` and `FontGet()`.

Folding support

The output sequencer editor supports code folding based on the keyboard link entry points. That is, you can fold up all code from a line holding a keyboard link up to the next line holding a link.

Change History

The output sequencer editor supports tracking changes in the same way as the script editor, if this is enabled.

Other text file settings

The editor settings dialog for all views except script and sequencer views is the same as the dialog for script views, except that there are no language settings and there are fewer styles:

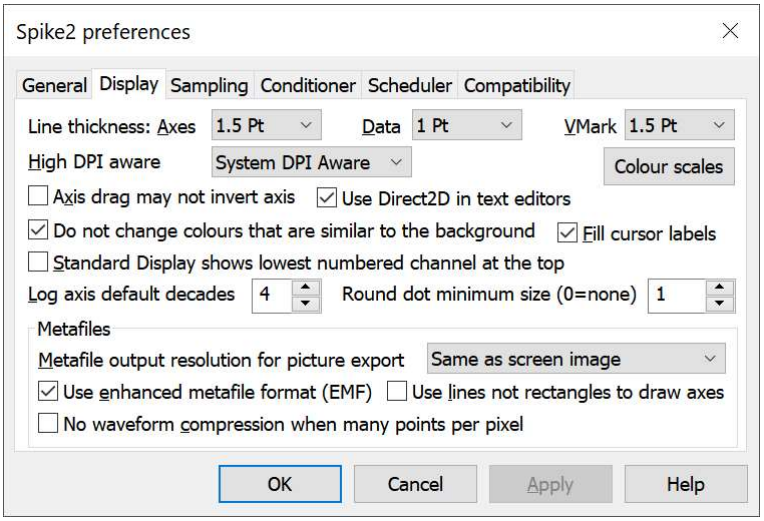
#	Name	Usage
32	Default	The basis of the other styles and sets the text caret colour. The Tab size is based on the width of a space in this style. If you change any aspect of this style, all other styles that match that aspect will also change. The background colour sets the view background colour.
0	Normal	General text, spaces, tabs, control characters and anything not covered elsewhere.

33 Line numbers The style used for line numbers and the gutter. When you select this item, line numbers will appear in the example text area.

The # column is the `style%` argument value for `FontSet()` and `FontGet()`.

Display

This dialog tab contains options for data display, printing and image export.



Line thickness: Axes, Data and VMark (Vertical Markers)

Choose from Hairline (as thin as possible) and a list of point sizes. A point is 1/72nd of an inch, which is typically about the size of a screen pixel. If coloured lines do not print on a monochrome printer, increase the line thickness or use the View menu Use Black and White command. Click Apply to redraw screen images to show the effect of any change. WARNING: wide lines take longer to draw than a single pixel line.

You can override the line thickness for particular channels with the Pen Width and XY Draw Mode dialogs. Channels drawn as dots set the dot size as a multiple of the pen width for data set in this dialog; so if you set a thick pen here, you will get large dots on screen.

High DPI aware

When Spike2 was originally written, screens typically had a resolution of 96 Dots Per Inch (DPI) or less. Nowadays screens have resolutions of 96, 120, 200 or higher DPI. To accommodate this, Spike2 has 3 sets of bitmaps for most buttons and icons. When Spike2 starts up, it makes a decision about which set to use depending on what it detects as the screen resolution. This works well for a single monitor and for multiple identical monitors. However, if you have multiple different monitors with different DPI values this may not work too well when you drag the application between monitors..

If you have a single monitor or multiple monitors with the same DPI and set to the same scaling, you need not read on. Set System DPI Aware and this should give the best result.

Microsoft has made several attempts at methods for applications to use to work around monitors with differing DPI values. This field allows you to choose the method to use:

Unaware — All graphics are rendered at 96 DPI and Windows stretches them to the actual resolution. This results in a slightly blurry look on all non-96 DPI monitors.

System DPI Aware — All graphics are rendered at the resolution of the main monitor and then mapped onto the screen at the resolution of the monitor on which the majority of the application window resides. This is the method that Spike2 is currently designed to use. Screens with different DPI/scale to the main monitor will have a slightly blurry appearance.

Per Monitor — Windows 10 1607 onwards. All windows and dialogs draw at the screen resolution (so look as crisp as possible). It is up to Spike2 to handle changes when the window changes monitor (which it currently does not) so items change size when you drag to screens of different DPI values.

Per Monitor V2	Windows 10 1703 onwards. A more advanced version of Per Monitor with better handling of system components. This can work well for some dialog boxes and for the text of text-based views (scripts, log, output sequencer), but other screen elements (such as bitmaps, picture buttons) will be the wrong size and it will require program changes to get them to work.
Unaware Scale GDI	Windows 10 1807 onwards. The same as Unaware, but some system elements are rendered at screen resolution.

Currently we suggest that you use System DPI Aware unless this causes unacceptable problems, in which case use whichever of the other modes works best for you.

Before [10.06] this option was called System DPI Unaware and allowed a choice of the first option, else the second was set.

Fill cursor labels (new at [10.05])
If you check this box, cursor labels are drawn with a solid background, which can make the labels easier to read against a busy signal background. If this box is unchecked, labels are drawn with a transparent background by XORing (inverting the colour) of text pixels.

Caveats
XORing the label is faster. A second XOR removes the effect of the first, avoiding a redraw of the data under the label when it is moved. Using a solid background is not perfect; when horizontal and vertical cursors cross each others labels and you swap from dragging one to dragging the other this will leave the moved line behind in the non-moved label. We disable solid backgrounds if you display so much data that redrawing becomes slow.

Axis drag may not invert axis
If you check this box, a drag of a y axis or the x axis in an XY view to change the scale (a drag starting in the number label region) is not allowed to change the direction of the axis. If you check the box you can still edit the axis limits to invert the sense of the axis by double clicking the axis to open the axis dialog.

Use Direct2D in text editors
The text editor we use (Scintilla), can use either the standard GDI method of drawing text or a potentially hardware accelerated Direct2D method (in Windows Vista onwards). The two methods should appear more or less identical; Direct2D may be faster.

Colour scales
Before Spike2 9.06 the Colour scale dialog was available from Sonogram displays and Clustering dialogs. As colour scales are now more generally useful (see `ArrMapImage()` and `Spline2D()` script commands) you can open the dialog from Edit Preferences.

Standard display shows the lowest numbered channels at the top
You can change the order of time and result view channels by clicking and dragging channel numbers. This field sets the order when you use the Standard display command or create a new channel. When a file is opened it affects the order in which the channels in the file are added to the view. If the box is unchecked, lower numbered channels are at the bottom.

Do not change colours when that are similar to the background
Spike2 checks how similar the colours of items such as channel data and text are to the background. If they are too similar, your colour choice is ignored and a more visible colour is used. Check this box to disable the colour check.

There are two levels of contrast: high-contrast is used when you need to read text or when dealing with very small dots, normal contrast is used for everything else. In high-contrast mode, if the foreground and background are similar in brightness, the foreground is set to white or black, whichever is most different from the background. In low-contrast mode, the foreground colour is modified by adding or subtracting a light grey from the colour (this attempts to preserve some idea of the original colour).

Log axis default decades

This field sets the number of decades to display when you swap an axis to logarithmic mode from the X or Y axis dialogs or Optimise the Y axis and the low limit is less than or equal to 0. On a logarithmic scale, negative numbers do not exist (as real numbers), and log(0) is minus infinity. In these cases, the low axis limit is set to be the high axis limit divided by 10 to the power of the number of decades you set here. For example, with 4 decades, if the upper value was set to 400, the lower value would be set to .04 (that is, `400/Pow(10,4)`).

Round dot minimum size

You can choose to draw round rather than square dots in time and result views. This field sets the minimum size (in pixels) at which to do this or you can set the value 0 to always draw dots as squares (to match older versions of the program). Not that printed output will typically have a much smaller pixel size that the screen, so dots that draw as squares on the screen may draw as circles when printed if you enable this option.

Metafiles

Spike2 saves time, result or XY views as pictures in either bitmap (a copy of the screen image) or metafile format. A metafile describes an image in terms of drawing operations. Many drawing program can import metafile as images, allowing you to scale, edit and annotate pictures without losing quality. We would suggest you use InkScape to display Spike2 metafile images and export them to a wide range of other formats including SVG, PDF, ODF and EPS. The Metafile group of display options relates to saving images from Spike2 as pictures either to the clipboard or to external files.

Metafile output resolution

Although metafile output is usually in terms of lines rather than bitmaps, the lines are still drawn on a grid. You can set the resolution of the grid, to give a higher picture quality than can be obtained with a bitmap of the screen. The higher the resolution, the more detail in the picture.

A problem for time and result views with many data points is that the higher the resolution, the more lines need to be drawn, and many drawing programs have limits on the number of lines they can cope with. You can use multiples of the screen resolution, or fixed resolutions. If you are not sure what setting to use, start with Same as screen image and adjust it as seems appropriate.

Use enhanced Metafile format

Spike2 supports two metafile formats: Windows Metafile (`WMF`) and Enhanced Metafile (`EMF`). `WMF` is a relic of 16-bit Windows and has limitations, but is widely supported. `EMF` is the standard for 32-bit programs and has many more features. However, some graphics packages do not support this fully.

With `EMF` format you can export waveforms in cubic spline mode and sonogram data as part of the metafile. With `WMF` format, waveforms in cubic spline mode export as plain lines and sonograms will be blank.

Use lines not rectangles to draw axes

This affects metafile output. Some graphics programs cannot cope with axes drawn as rectangles; check this box to draw axes as lines. We use rectangles to make sure that axes drawn with pens of other than hairline thickness join neatly.

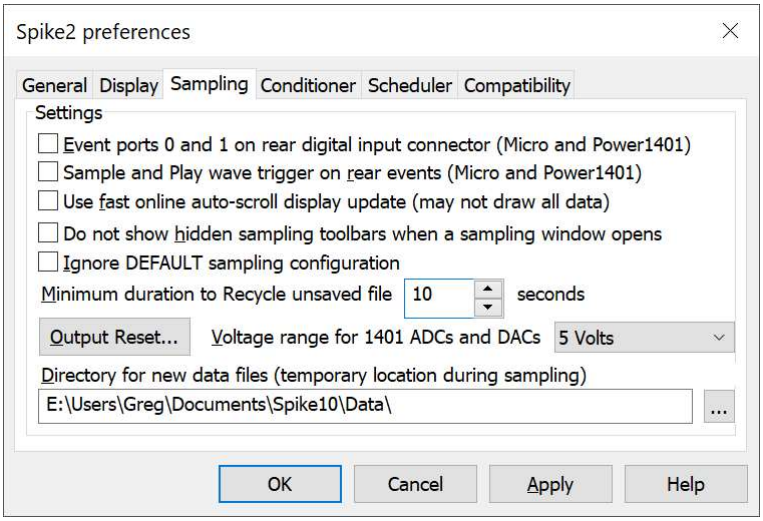
Do not compress metafile waveform output

When Spike2 draws waveform data, it does not waste time with lines that make no visual difference to the output. If there are more than 3 data points per horizontal pixel, Spike2 draws one vertical line per horizontal pixel to give the same visual effect as all the separate lines. However, when you export an image as a metafile this may not look correct as it relies on a perfect match between the vertical line width and separation.

If you check this box, no compression is done when data is saved as a metafile. For the most precise image, set the Metafile output resolution to 16000 units and check this box. When you import a metafile, most drawing programs will preserve the waveform as connected lines. Spike2 writes long sequences of data points in blocks of up to 4000 points. The last point of a block is at the same position as the first point of the next.

Sampling

This tab sets preferences associated with sampling data and the 1401.



Event ports 0 and 1 on rear digital input connector

You can source event ports 0 and 1 from the front panel or the digital input connector. Check the box to use the rear panel. Leave it unchecked to source your event signals from the front panel Event 0 and 1 inputs.

Sample and Play wave trigger on rear events

You can trigger sampling and on-line waveform output with the front panel Trigger BNC, or on the rear panel Events connector pin 4 (GND is pins 9-15). Check the box to use the rear connector. The normal trigger is a high to low TTL pulse or a switch closure. See the Owners manual for your 1401 for the electrical specification of these signals.

Use fast online auto-scroll display update

The online automatic scrolling of new data into the display during sampling can make the host computer feel unresponsive, especially in complex display modes at high sampling rates. Check this box to use a faster (but incorrect) algorithm to decide how much of the screen to repaint when the view scrolls during sampling. We may remove this option.

In the faster mode, Spike2 does not properly allow for changes in already drawn data that depend on newly sampled data that has just scrolled into the display. This is particularly visible with sonograms and channels with channel processing options such as time shifts.

Do not show hidden sampling toolbars when a sampling window opens

When you open a data file for sampling, Spike2 will normally also show the Sample control toolbar, the Sample Status toolbar and the Output Sequence control bar (if there is an output sequence). If you check this box, these toolbars remain in their current state. You may wish to do this to prevent the toolbars changing the screen arrangement.

Ignore DEFAULT sampling configuration [9.02]

When Spike2 starts it searches for saved sampling configuration. Normally, if a configuration set as the default exists it will be selected, otherwise the last used configuration is selected. If you check this box, any default configurations are ignored. Saving a default configuration clears this option as we assume that you intend to use the default.

Minimum duration to Recycle unsaved file [10.20]

Previously, if you did not save a time view file that you had created, it was automatically moved to the Recycle bin when it closed. This was a safety feature to stop accidental data loss. This can lead to the recycle bin filling up with a lot of files, which can slow down the system. Also, emptying the recycle bin of a lot of files is a slow

process. This field allows you to specify the minimum file duration to save. Files shorter (in time) than the number of seconds you set here will not be recycled if closed without saving. Set 0 for the old behaviour that Recycled all unsaved files.

Voltage range for 1401 ADC and DACs

Most 1401s have a ±5 Volt input range, but some have ±10 Volts. Spike2 detects the range of connected Power1401s and Micro1401s automatically. Choose from 5 Volt, 10 Volt and Last seen hardware. You are warned if Spike2 detects a conflict between user settings and installed hardware. The voltage range affects scaling in the sampling configuration and DAC output values in the output sequencer. It has no effect on scale values in previously sampled data files.

Output Reset...

This button opens a dialog in which you can choose to have the Digital and DAC outputs set to known values when Spike2 starts, before sampling and after sampling. Values set from the Edit menu Preferences can be overridden by values set in the sampling configuration Automation tab.

Directory for new data files

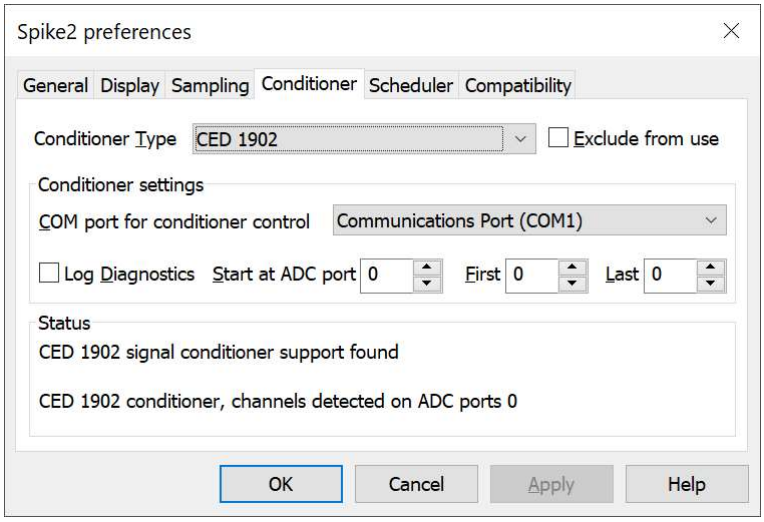
This is the directory/folder where Spike2 stores files data created by File menu New during sampling. Click ... to select or create a new folder interactively. If you do not set a directory, Spike2 uses the current directory, which may not be where you expect, so it is a better to set one. If you have more than one disk drive, choose a folder in your fastest drive. Do not use a networked drive. New data files in this folder do not have a file extension. The folder you choose must exist.

When you close a new data file, Spike2 prompts you for a file name. What happens next depends on where you choose to save the file. If the file is on the same drive as the directory/folder set in the here, Spike2 renames the file (quick). If the drive is not the same, Spike2 copies the file (slow), and deletes the original.

Do not choose a place that requires administrator privilege unless you understand the implications of this.

Conditioner

This tab configures the Programmable signal conditioner types to use, or to exclude them from use. It also sets the mapping between 1401 ADC ports and the conditioner channels.



Conditioner Type

Select the conditioner that you want to configure. There may be a delay after selecting a conditioner (or before this dialog page appears) as Spike2 searches for an installed conditioner of the selected type. You can track how this is progressing with the Status area. There are two areas of Status reports. The upper area reports on the search for the signal conditioner support software, which should always succeed unless you have deleted the signal conditioner support DLL from the Spike installation folder. The lower area reports if the conditioner hardware was detected, and if so, which ADC ports have support.

Backwards compatibility with Spike2 versions before [10.17]
Older versions support use of a single signal conditioner type, which is selected by this field. If you want to maintain compatibility you must make sure that the conditioner you want to use with the old versions is the last one you set and that the Start field is set to 0.

You can select None as the conditioner type. This is for backwards compatibility with Spike2 versions that supported a single conditioner where setting None meant no use of signal conditioners.

Exclude from use
If you check this box, Spike2 will not use the selected signal conditioner type for sampling and will not load the conditioner support or test for the conditioner being present. It will save you some time (and system memory) when using Spike2 to check this box for all conditioners that you do not use. This is particularly the case for the serial line controlled CED 1902 and Axon CyberAmp, where the scanning process can be time consuming, especially if the First and Last fields are set to unfortunate values. Also, the scanning process sends data through the selected COM port, which could have unfortunate consequences if other equipment is connected.

COM port for conditioner control
This is enabled for the CED 1902 and the Axon CyberAmp and sets the communication port used to control the device hardware. You can select a port from the drop down list of detected ports. A change to the port initiates a scan for connected and powered devices. The result of the scan appears in the Status field. You cannot detect a device if it is not powered.

Log Diagnostics
A CED engineer may ask you to check this box to help diagnose problems connecting to a device. It sends information about the connection process to the Log window. For example, typical output with a single CED 1902 might be:

```
About to open serial port 4 for use with 1902s
Found 1902 channel 0 OK
Timed out waiting for response for 1902 channel 1
```

Start at ADC port (Not Power1401 with ADC gain option)
This field was added in Spike2 version [10.17] and if you set it non-zero, the signal conditioner settings saved will still work for older Spike2 versions, but they will behave as if Start were set to 0.

This field sets the first ADC port that the zeroth conditioner unit of the selected type is connected to. This port will expect to have a conditioner unless the First field (see below) is set non-zero, in which case the first ADC port with a conditioner is the Start field plus the First field times the number of conditioned channels per device. In almost all cases, First is 0.

You will most often set this field when you have more that one type of signal conditioner and you must arrange that they process different channel ranges.

Note that a Power1401 with the gain option does not use this field as the gain is an integral part of the ADC port and cannot be mapped.

First and Last (CED 1902 and CyberAmp only)
These fields are used with the CED 1902 and the Axon CyberAmp to speed up scanning for devices. Both these devices can be assigned unit numbers (starting with 0) with switches. These values control which unit numbers are scanned for. CED 1902 devices have one channel per device and Axon CyberAmp devices have either 8 or 2 channels per device. If you use multiple CyberAmp devices, we assume they all have the same number of channels as the first device. If there are `c` channels per device, the ADC port number associated with the first channel on device with unit number `n` is:

```
Start + n * c
```

This means that if your devices have consecutive unit numbers starting at 0, these devices span a range of ADC ports beginning at `Start`. However, you have a free choice of unit numbers (as long as they are all different), so other arrangements are possible.

The First field sets the lowest unit number to scan for. Devices with lower numbers will not be detected. In the case of the CyberAmp, this device *must* be found, otherwise no further devices will be searched for.

We scan for devices in ascending unit number, starting at First until a device with a unit number greater than Last does not respond. Most users set consecutive unit numbers and leave Last set to 0.

From version [10.17] you can also set Last to minus the first device you *do not* want to search. This must be greater than -First or it is ignored. This avoids waiting for a device to time out to terminate the search.

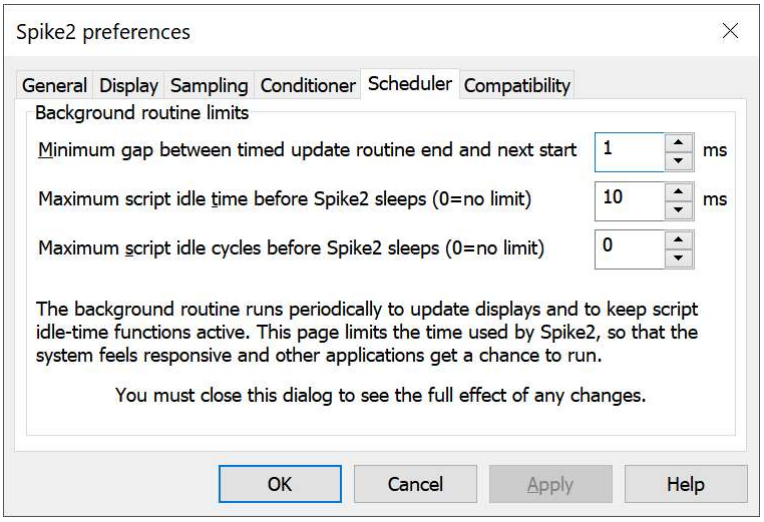
The Status panel gives information on the selected signal conditioner support. If Spike2 can detect a signal conditioner, it also holds information about the channels that have conditioner support.

Using multiple conditioner types

Spike2 version [10.17] onwards supports the use of multiple signal conditioner types at the same time. Prior to this, only one conditioner type was supported, and this was selected here. Now, support for all types is loaded (unless you Exclude them) and each conditioner type is assigned a range of channels. You are warned if you attempt to close this panel with more than one conditioner assigned to any ADC port.

if you have multiple conditioner types, it is your responsibility to ensure that there is a maximum of one conditioner per 1401 ADC port. The dialog will warn you if it detects more than one conditioner active for an ADC port.

Scheduler



This tab limits the processor time consumed by the Spike2 user thread while sampling and idling with a script running. If Spike2 takes too much time, the system feels unresponsive. The tab does not affect the time-critical threads used for data capture or the threads used for multimedia. You can read more about threads below. Spike2 runs a background routine when it has idle time and also periodically on a timer. The background routine handles the following tasks:

- When sampling or rerunning, it gives windows a chance to detect that the maximum time in a sampled data file has changed, which may cause windows to scroll and processing to occur. Any invalidated windows will update the next time Spike2 gets idle time.
- If a script is running, it gives any "idle function" in the script a chance to run. See `ToolbarSet()` and `DlgAllow()` and `DlgMouse()` to set idle functions.
- In automatic file naming mode it starts the next file running.

There are three fields that limit the time used in the background routine. They have no effect on the time used when Spike2 runs a script that does not idle (see the `Yield()` or `YieldSystem()` script commands for this). The standard values work for most cases.

Minimum gap between timed update routine end and next start

If the time interval set by this field passes without the background routine running, it is scheduled to run as soon as possible. You can use values in the range 1 to 200 milliseconds. The standard value is 10 milliseconds. The lower the value, the more time Spike2 will spend on background processing relative to other applications. This field also limits the time that Spike2 will sleep for (see the discussion of threads, below).

If you rely on background processing during sampling, for example for the Sampling mode Script Trigger [10.09] or the Measurement to Keyboard channel feature [10.10], you will likely want to set this to a lower value (maybe 5 milliseconds).

Maximum duration of script idle before Spike2 sleeps

When Spike2 gets idle time (see the discussion of threads, below), you can limit the time it uses before Spike2 goes to sleep. Spike2 uses idle time to run script idle routines, such as those created by `ToolbarSet(0,...)`. You can set from 0 to 200 milliseconds (0 means no time limit). The standard value is 10 milliseconds. The larger the value, the more the script idle routine runs at the expense of other applications.

Maximum script idle cycles before Spike2 sleeps

As an alternative to limiting the script idle routine by elapsed time, you can limit it by the number of times it is called. You can set from 0 to 65535 times (0 means no limit). The standard value is 0. Setting both this and the Maximum duration... field to 0 is unkind to other applications. Setting this to 1 is the most generous to other applications.

Threads

A *thread* is the basic unit of program execution; a thread performs a list of actions, one at a time, in order. To give you the impression that a system with one processor can run multiple tasks simultaneously, the system scheduler hands out time-slices of around 10 milliseconds to the highest priority thread capable of running. Tasks at the same priority level share time-slices on a round robin basis. Lower priority tasks rely on higher priority tasks "going to sleep" when they have nothing to do or when they are blocked (for example, waiting for a disk read). If this did not happen, low priority tasks would not run.

There are also very high priority tasks, usually associated with hardware device drivers, that can interrupt the scheduling system to respond to external events within microseconds. These interrupts usually only last a few microseconds; if a device driver needs a longer period of time it will request a time slice to complete its work and wait for it to be granted.

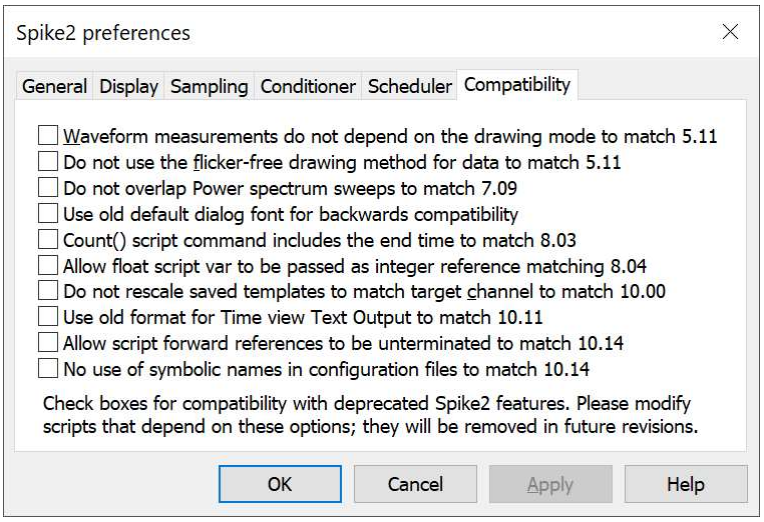
When Spike2 gets a chance to run, it processes pending messages such as button clicks, keyboard commands, mouse actions and timer events and then updates invalid screen areas. Finally, Spike2 is given idle time until it says it does not need any or new messages occur. If Spike2 needs no more idle time it sleeps until a new message appears in the input queue, which wakes it up again. The Minimum gap... timer wakes up Spike2 if nothing else happens. All these activities happen in a single, user thread.

This single user thread can also generate special threads that perform specific actions. These special threads do not process pending messages or update the screen. These separate threads are typically used to monitor activities that need fast responses (cannot wait for the single user thread to have idle time) or that can take a while and would make the user thread feel unresponsive. Examples of these are:

- Real-time sampling thread that moves data sampled by a 1401 into the Spike2 data filing system.
- Real-time thread to move data from Talker devices to the data file.
- Multiple threads used for the video capture and replay system.
- Thread used to calculate FIR filter coefficients in the filtering dialog. This can take a detectable time, which would make dragging items in the dialog feel unresponsive, so we calculate in a separate thread and update when calculations are done.

These operations can overlap with the user thread actions. In a single processor system, the processor swaps around between all available threads, assigning time to them based on their priority. If your system supports multiple processors or multi-core processors or hyper threading, then threads can run simultaneously, and you will feel the benefit of this in smoother and faster system operation, particularly when sampling data or recording video. Of course, we have to be careful that the threads do not mangle each others data.

Compatibility



This tab lets you disable new program features that have changed the way that Spike2 works and that might affect your results. We reserve the right to remove these options in future releases. If you need to check any of the boxes, please let us know so we have some idea of the number of users who depend on the old behaviour (no-one has ever done this!) These settings are saved in the system registry and are read when they are needed. You can modify the settings with the `Profile()` script command

Waveform measurements do not depend on drawing mode

Before version [5.12], measurements from waveform channels returned the nearest data point. Now they return what you see on screen. Check the box for the nearest data point.

Do not use the flicker-free drawing method

Version [5.12] implements a new buffered drawing method that reduces screen flicker on updates. However, this may impact the display speed. Check the box for the old method.

Do not overlap Power spectrum sweeps to match 7.09

From version [7.10], we overlap the Power spectrum sweeps. This will tend to weight the data more equally and can give more consistent results at the cost of more time spent transforming the data. Check this box for the old behaviour.

Use old default dialog font for backwards compatibility

Version [8.03] sets the equivalent of `DlgFont(1)` each time the script starts rather than the old default of `DlgFont(0)`. This generates better-looking text. However, it is possible that this will change the layout of user-defined dialogs, which can cause text to vanish, making a dialog difficult to use. Check the box to force the old behaviour.

Count() script command includes the end time to match 8.03

At version [8.04], the `Count()` command was changed so that the range did not include the end position. This ensured that if `t1<t2<t3`, `Count(chan%,t1,t2)+Count(chan%,t2,t3)` was the same as `Count(chan%,t1,t3)`. Check the box to restore the old behaviour.

Allow float script var to be passed as integer reference matching 8.04

For reasons of backwards compatibility, we allowed the following code to compile and run:

```
Proc OddBehavior(&integer%)
Message(integer%);
integer% := 23;
end;
```

```
var notInteger := 1.23;
OddBehavior(notInteger);  '8.05 onwards give an error here
```

From [8.05] onwards we no longer allow this and insist that reference arguments must match in type (as Nature intended). You can set this option to allow the old behaviour, but we urge you to modify the script to use the correct type of variable. We will likely remove this compatibility workaround at some future release.

Do not rescale saved templates to match target channel

From version [10.01] onwards, when a saved template does not match the channel scaling of the target channel (for example if the amplifier gains were adjusted), we now rescale the template (where plausible to do so) to match the target channel. To do this, the channels should have the same units, or at least recognisable units (`mV`, `uV` etc). Check this box to disable scaling, to behave the same as all previous versions of Spike2.

Use old format for Time view Text Output to match 10.11

Versions [10.12] onwards have changed the Time view Text Output SUMMARY section to show the same information for all channels, which should make it easier to write an importer for the format. We have also revised the output of Level event channels and WaveMark data. Check this option to revert to the old format (for example if you have an importing program that relies on the old format).

Allow script forward references to be unterminated to match 10.14

Versions [10.15] onwards flag the following script as incorrect:

```
Test()        'Call to procedure that has not yet been defined
halt;          'Error at start of this line, 'Missing ;'

Proc Test()
end;
```

However, previous versions of the compiler accepted this, so we have added an option to force the compiler to accept it. The fix is easy; add a semicolon to the first line:

```
Test();
halt;
```

No use of symbolic names in configuration files to match 10.14

Version [10.15] introduces the experimental use of symbolic names in sampling configuration files (`*.s2cx`). If you suspect a problem with this feature, check this box to disable it and report this to CED.

Removed compatibility options

The following options have been removed from the Compatibility dialog to make space for new options.

Use old-style colour mechanisms

Before version [5.04] in 2003, the colour palette was saved in sampling configuration files; loading a sampling configuration set the colour preferences. We now save the palette in the registry. Check this box to save the palette to configuration files and read colours from configuration files (if there are any), as before. This option is still supported and can be set using the `Profile()` script command with the "`Use old colours`" option

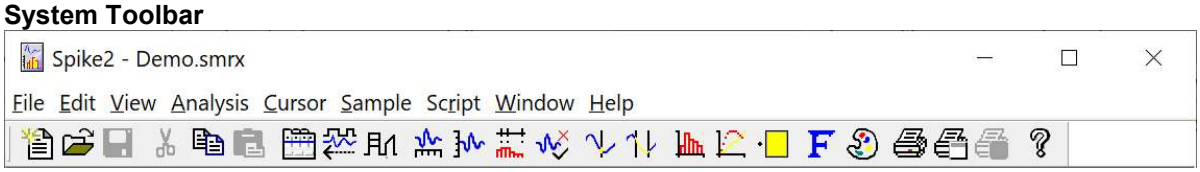
8: View menu

View menu

This menu controls what you see in Spike2 data views and how Spike2 displays the data. Views are generally divided into two basic categories: data-based views (time, result and XY views) and text-based views (script, output sequence, text and log views). The contents of the menu changes depending on the type of the view. The menu also has commands to control the Toolbar and the Status bar.

Toolbar and Status bar

System Toolbar



The system toolbar is normally docked below the Spike2 application menu, but can be dragged to become a floating window or docked to any side of the application window. The toolbar contains short-cuts to some of the commonly-used menu commands. When the Spike2 application is active, and you move the mouse pointer over one of the buttons, the Status bar will display a reminder of the buttons function. If you hover the mouse pointer over a button, a tool tip will appear with a short reminder. Buttons that are not currently enabled (because of the current view) are drawn in grey.

The Toolbar command in the View menu lets you show and hide the system toolbar bar. Script users can get the system toolbar handle with **`App(1)`** and show and hide the system toolbar with:

```
View(App(1)).WindowVisible(show%);  'Show%=1 to show, 0 to hide
```

To dock and position the toolbar, use the `Window()` script command. The `xHigh` argument sets the docking edge.

Status bar

The Status bar occupies the very bottom of the Spike2 application when it is enabled. The left hand end of the bar contains text that describes the currently selected menu command, or a reminder to use F1 to get help. The next field displays the line number and column number of the text cursor when a text view is current, or when a Time, Result or XY view is current it display the last known mouse position as [channel,] x position[, yposition] (items in square brackets are omitted if they do not apply). The remaining fields are:

CAP The `Caps Lock` key is active. Press `Caps Lock` on the keyboard to disable.

NUM The `Num Lock` key is active. Press `Num Lock` on the keyboard to disable.

OVR Over type mode is active in a text-based view so that typing replaces text. Press the `Ins` key to disable (this only has an effect when the current view is a text-based view).

REC Script recording is enabled. You can Turn Recording Off in the Script menu.

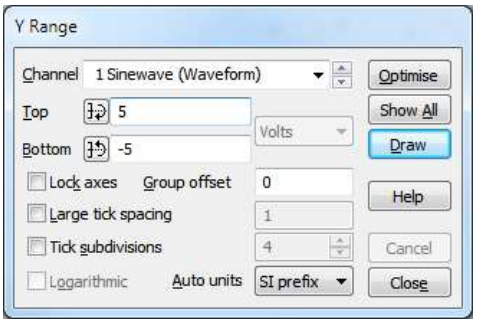
The Status bar command in the View menu lets you show and hide the status bar. Script users can get the status bar handle with **`App(2)`** and show or hide the bar with:

```
View(App(2)).WindowVisible(show%);  'Show%=1 to show, 0 to hide
```

Enlarge View Reduce View

These two commands duplicate the two buttons at the lower left of time and result windows. The enlarge command, short cut `Ctrl+E`, doubles the x axis data region and the reduce command, `Ctrl+R`, halves the region. The left hand window edge is fixed unless enlarging would display data beyond the end of the data, in which case the displayed area is moved backwards. If enlarging would display more data than exists, all the data is displayed. Short cut keys `Ctrl+U` and `Ctrl+I` zoom about the screen centre. You can also change the view size by clicking and dragging the x axis numbers.

Y Axis Range



This dialog sets the y axis range and style for visible time, result or XY view channels. The Channel field chooses one, all or selected channels or you can type a list of channels. If more than one channel is selected, the displayed settings are for the first visible channel that matches the displayed channel units.

Optimise

Click Optimise to draw the visible data scaled and offset to fill the available space. In most cases, we allow an extra 5% of the space at the top and bottom which avoids a 'cramped' look and allows a little headroom from channels drawn in cubic spline mode (where the peaks can be above the range of channel values). If the extra 5% of space crosses the value 0.0, the axis stops at the zero, for neatness. From version [10.02], if you draw grouped channels in Lock axes mode with a non-zero Group offset, we do not apply any extra space so that all available space is used, which looks better with a large number of grouped channels.

Show All

Click Show All to set the y axis to display the maximum possible range for waveform channels and from 0 to the estimated event rate for event channels drawn with a frequency axis. Both these buttons close the dialog. You can optimise without opening the dialog with the keyboard short-cut `Ctrl+Q` and by right-clicking on a channel and selecting the Optimise option from the context menu. In logarithmic mode, values of zero or less cannot be displayed.

Top and Bottom

The Top and Bottom fields set the values at the top and bottom of the axis. The buttons to the left of these fields make the axis symmetric about zero. The units of the values are displayed to the right of these fields. If there are multiple channels selected, and the channels have different units, you can select the units or <any> to match any units. Click Draw to apply changes to the Top, Bottom and Group offset fields. These changes are only applied to channels that match the displayed units; select <any> as the units to apply the values to all channels in the list.

Cancel undoes any changes and closes the dialog. The Close button closes the dialog; it does not apply changes to the Top and Bottom fields.

Lock Axes and Group offset/factor

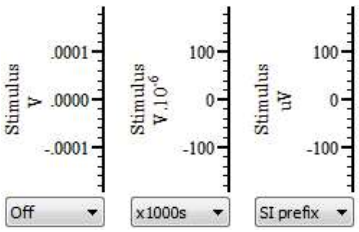
The Lock axes and Group offset/factor fields are visible when the current channel shares its y axis with other channels. The fields are enabled when the current channel is the first in the group. If you check the Lock axes box, the grouped channels not only share the same space, they also share the y axis of the first channel in the group. The Group offset field sets a per-channel vertical display offset to apply to each locked channel so you can space out channels with the same mean level. If the axis is logarithmic the field becomes Group factor as a constant shift becomes a multiplicative factor.

Tick spacing and subdivisions

When preparing data for publication you may wish to set the spacing between the major tick marks and the number of tick subdivisions. If you prefer a scale bar to an axis, you can select this in the Show/Hide channel dialog. You can control the Large tick spacing (this also sets the scale bar size) and the number of Tick subdivisions by checking the boxes. Your settings are ignored if they would produce an illegible axis. Changes made to these fields take effect immediately; there is no need to use the Draw button.

Auto units

The Auto units field is present when the Logarithmic box is not checked. There are three options:



Off The axis behaves normally, using the standard y axis units.

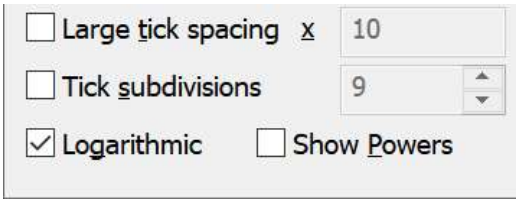
x1000s If the axis range becomes numerically very large or very small and scaling makes sense, the axis units are multiplied by a power of a thousand and the y axis units are displayed with the power of ten after the units.

SI prefix The same as x1000s, except that the y axis units are prefixed by a SI scaling prefix (M, k, m, u, ...). If the units already start with an SI prefix, this is removed (and taken into account) before display.

Any change made to an axis by this field is purely cosmetic; it has no effect on the values seen by Spike2 or by a script. It changes the visible axis values and the values displayed on screen by horizontal cursors and by vertical measurements made by holding down the `Alt` key and clicking and dragging in a channel.

Logarithmic and Show Powers

If the units are scaled, cursor labels and on screen measurements taken by holding down `Alt` and clicking and dragging are also changed. Apart from these cosmetic effects, all other measurements remain in the original axis units.



The Logarithmic check box can be used in Result and XY views to change the y axis to logarithmic mode. In logarithmic mode, the large tick spacing field becomes a multiplying factor between large ticks and is usually set to 10. The Group offset field also changes to Group factor. You can use all drawing modes with logarithmic axes; however, straight or cubic-splined lines between points will not pass through the same values as with linear axes. If either end of the axis is less than or equal to zero when you set logarithmic mode, the axis limits are adjusted.

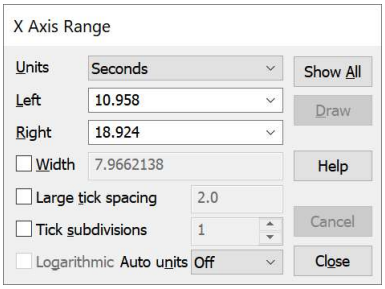
The Show Powers check box is present in logarithmic mode and replaces the Auto-adjust units field. With this box checked, the axis labels at major tick marks are displayed as powers of 10 (or of the value set in the Large tick spacing field).

Axis adjustments in logarithmic mode

See the Edit menu Preferences dialog Display tab to set the number of decades to display when an axis switches from linear to logarithmic mode or you Optimise in logarithmic mode and the low limit is less than or equal to 0.

X Axis Range

The X Axis Range dialog is opened from the View menu or by double-clicking on the x axis of a time, result or XY window. The dialog sets the x axis range to display and also controls the display units and the style of the axis. Not all features of the dialog are available in all view types; logarithmic axes are not allowed in time view, for example.



You can close the dialog in two ways: the Close button closes the dialog and accepts any pending changes. Cancel undoes any changes made with the dialog and closes it.

Units

The Units field in a time window selects Seconds, hh:mm:ss (hours, minutes, seconds), Time of day or milliseconds display mode. Time of day works for files created by Spike2 version 4.02 onwards; for older files with no saved creation time it is the same as hh:mm:ss. In result and XY views, the Units field display whatever the X axis units are set to. If the x axis units are "s" or "seconds", you can also choose milliseconds as your units.

The Time of day axis mode draws times as hh:mm:ss on the x axis that are the time of day at which the data was collected plus the time offset into the file. This is for display purposes only; all times used within Spike2 are times in seconds from the start of sampling. All times entered into dialogs are relative to the start of the file. From [10.11] you can type numbers that match the axis by following them with `tod`, for example `12:00:00tod`.

The milliseconds selection changes how the axis labels are displayed. Internally, Spike2 still uses seconds for all measurements and commands related to the axis. If you select milliseconds, all the time-based fields in the dialog (Left, Right, Width and Large tick spacing) change to milliseconds. When recording actions, if you use expressions (for example `Cursor(1)+1`) in milliseconds mode that would evaluate to a different result in seconds mode, these will record as the result of the expression in seconds.

The script language equivalent of this field is the `XAxisStyle()` command.

Left, Right and Width

The Left and Right fields set the window start and end. You can type in new positions or use the drop down lists next to each field. The drop down list contains the initial field value, cursor positions, the minimum and maximum allowed values and the left and right edges of the window (`XLow()` and `XHigh()`). The Width field sets the window width if the Width box is checked.

Click the Draw button to apply changes in these fields to the window. The Draw button is disabled if the dialog range matches the current range or if the dialog range is invalid (badly formatted number or the range exceeds the range of available data in the file).

If you type a number in these fields, it is interpreted in the units set for the Units field. In a time view, you can follow a typed number by `s`, `ms` or `us` to force the number to be interpreted as seconds, milliseconds or microseconds regardless of the settings in the Units field. In Time of Day mode, numbers typed in need to be expressed in time of day format or as the time of day expressed in seconds.

The script language equivalent of these fields are the `Draw()` and `XRange()` commands.

Show All

Show All in Time and Result views expands the x axis to display all the data the dialog. In an XY view, it sets the x axis range to show the full range of data channels that are not hidden plus a little extra space at the ends of the axis (unless one of the ends is at 0, when 0 is set). Show All closes the dialog. If all the points are at the same x position it centres the axis on the position.

Tick spacing

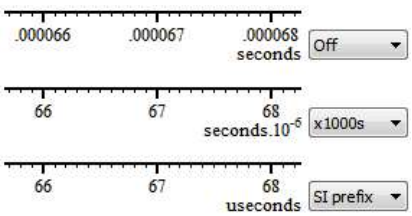
In normal use, you will let Spike2 organise the x axis style. However, when preparing data for publication you may wish to set the spacing between the major tick marks and the number of tick subdivisions. If you prefer a scale bar to an axis, you can select this in the Show/Hide channel dialog.

You can control the Large tick spacing (this also sets the scale bar size) and the number of Tick subdivisions by ticking the boxes. Your settings are ignored if they would produce an illegible axis. Changes made to these fields take effect immediately; there is no need to use the Draw button.

From a script, you can control the tick spacing with the `XAxisStyle()` command.

Auto units

The Auto units field is present when the Logarithmic box is not checked and the Units field is not set to milliseconds, hh:mm:ss or Time of day. There are three options:



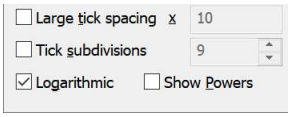
Off — The axis behaves normally, using the standard x axis units.

x1000s — If the entire axis in time or result views or the visible axis in an XY view becomes numerically very large or very small and scaling makes sense, the axis units are multiplied by a power of a thousand and the x axis units are displayed with the power of ten after the units.

SI prefix — The same as x1000s, except that the x axis units are prefixed by a SI scaling prefix (M, k, m, u, ...). If the units already start with an SI prefix, this is removed (and taken into account) before display.

If the units are scaled, cursor labels and on screen measurements (hold down `Alt` and click and drag) are also scaled. Apart from these cosmetic effects, all other measurements remain in the original axis units.

Logarithmic



The Logarithmic check box can be used in Result and XY views to change the x axis to logarithmic mode. In logarithmic mode, the large tick spacing field becomes a multiplying factor between large ticks and is usually set to 10. You can use all drawing modes with logarithmic axes; however, straight or cubic-splined lines between points will not pass through the same values as with linear axes. Furthermore, you can generate apparently blank axes in logarithmic mode. For example, if the data range of the axis was 61 to 69 units, changing this to logarithmic mode will generate a blank axis as the nearest minor ticks are at 60 and 70 and the nearest major ticks are at 10 and 100.

The Show Powers check box replaces the Auto-adjust units field in logarithmic mode. Check this box to display the axis labels at major tick marks as powers of 10 (or the value set in the Large tick spacing field).

Script language equivalents to this dialog

The `XAxisAttrib()` script command is the equivalent of the Logarithmic and Show Powers check boxes and the Auto units selector. The `XAxisStyle()` command sets the Units and the tick spacings and number of subdivisions of major ticks. The `XRange()` and `Draw()` commands set the displayed range. You can show and hide axis features with `XAxisMode()`. You can record your actions in this dialog by turning recording on, using the dialog, closing it and turning recording off.

Short cut keys and mouse wheel

Short cut keys that control the x axis are: `Home` and `End` move to the start and end of the data, `Left` and `Right` arrow scroll by one pixel, `Shift+Left` and `Shift+Right` scroll by several pixels and `Ctrl+Left` and `Ctrl+Right` move by half the screen width. The mouse wheel will also scroll the x axis one pixel at a time; add `Shift` or `Ctrl` to scroll by larger numbers of pixels.

Jump to event

In a time view you can jump to the next or previous event by selecting the event channel, then using `Alt+Right/Left` arrow. Spike2 searches for the nearest event to the centre of the screen to the right or left. If more than one event channel is selected, all channels are scanned for the nearest event. Spike2 beeps if none is found.

You can jump to any TextMark from the TextMark dialog. Double-click any TextMark to open the dialog, click the >> button, select the marker from the list and click Show.

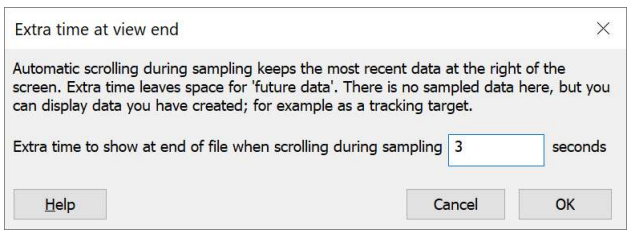
X Axis Extra Time

During sampling, the final length of the data file is unknown and we pretend that it is the maximum length possible. To make navigation through the file with the horizontal scroll bar work in a reasonable manner we 'pretend' that the file length is the current time into sampling. When the current sample time reaches the right-hand size of the screen the display starts to scroll to the left to maintain the current sample time.

This is usually what you want as there is no sampled data (yet) in the future.

Occasionally, however, you may want to write data into a non-sampled channel *in the future*. For example, in some training tasks you may want a subject match a template waveform with a force or breathing pattern. For them to do this, they need to see the waveform *ahead* of the current time. To allow for this, you can set extra time at the end of the X axis.

You can set this time with the View menu X Axis Extra Time... command, or by hovering the mouse over the X axis of a sampling Time view and selecting the X Axis Extra Time... command to open the *Extra time at view end* dialog:



The script language equivalent of this is the `ViewExtraTime()` command; it is likely that this feature will generally be used from a script as you will probably want to write a training waveform into the file. The following script generates a Sinusoidal waveform on channel 2 and places channel 1 on top of it:

```
SampleClear(); 'Set standard state (32 channels, 64-bit smrx file)
SampleOptimise(0, -1, 9); 'No optimise, set 1401 type, (do first)
SampleUsPerTime(10);
SampleTimePerAdc(1000);
SampleStartTrigger(-1); 'Set trigger to start sampling

'Channel recording definitions (adjust as required)
SampleWaveform(1, 0,100); ' chan, port, rate of 100 Hz
SampleTitle$(1,"User");
SampleTextMark(80);
SampleTitle$(30,"Title30");
SampleComment$(30,"Comment30");
SampleTitle$(31,"Title31");
SampleComment$(31,"Comment31");
SampleOptimise(1,1,9,1,50);  'Set optimise mode (do this last)
SampleMode(1); 'Continuous sampling

'Create data file for sampling
var fh:=FileNew(0, 1);       ' Create Data view
if fh <= 0 then Message("Failed to create view, error %d : %s", fh, Error$(fh)); halt endif;

ViewExtraTime(4);            ' Display 4 seconds into the future
PrintLog("ViewExtraTime() is %g\n", ViewExtraTime());
XRange(0, 6);                ' Set display width to 6 seconds
View(SampleHandle(1)).WindowVisible(1); 'Show sampling controls

var wave[100];               ' Space to hold a sinusoid
ArrConst(wave[1:], 2*_pi/20); '20 points per cycle
ArrIntgl(wave);
Sin(wave);                   ' generate sinusoid
ArrMul(wave, 4);             ' scale to +- 4 Volts
ChanNew(2, 9, 0, 0.1);       ' Make a channel sampled at 10 Hz
ChanTitle$(2, "Training");
ChanWriteWave(2, wave, 0);  ' Write 10 seconds of data at time 0

ChanPenWidth(2, 2);          ' Make channel 2 draw thicker
ChanColourSet(2, 1, 0.9, 0.9, 0.9);   ' Channel primary colour: light gray
ChanShow(2);                 ' Make channel 2 visible
ChanOrder(1, 0, 2);          ' Group with and under channel 1
YAxisLock(1, 1,0);           ' Set group y axis lock so they keep the same scale
Yield(3);                    ' Pause at the start

SampleStart();               ' Sample now (or omit for a manual start)
Yield(10);                   ' Wait 10 seconds
ChanWriteWave(2, wave, 15); ' Write next training wave at 15-25 seconds
Yield(15);                   ' Wait for 15 seconds
SampleStop();                ' Stop sampling
```

The script illustrates how the training waveform on channel 2 can be written before sampling starts and after during sampling. Using `Yield()` is a simplistic way to allow the system time to scroll the display. A real application script would probably use a `Toolbar()` command to allow user interaction and control.

Standard Display

This command sets the current time, result or XY view to a standard state. The script language equivalent is `ViewStandard()`. The x and y axes are displayed with all user options for grids, tick spacing and special axis modes turned off. In time and result views, duplicate channels are deleted and all channels are displayed in a standard mode and size and ordered as set in the Edit menu Preferences, all special channel colours are reset and any channel processing is removed. In an XY view, all channels are made visible, the point display mode is

set to dot at the standard size, the points are joined, the x and y axis range is set to span the range of the data and all the channel Z Order values are set to 0.

All cursors are removed regardless of values set by the `CursorFlags()` script command. This is the only non-script way to get rid of vertical cursors that are fixed in position and have the cursor flags set for no interactive delete.

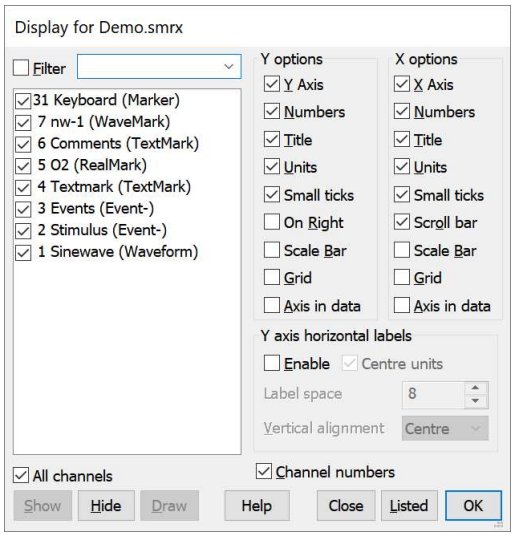
In a time view, Marker derived channels all display the first marker code. WaveMark channels are set to always display markers in hexadecimal format, all others will show printing characters. Any extra time set for the x axis is removed.

In a text-based view it removes any maximum line limit except in the Log view, where is applies the line limit set in the Edit menu Preferences option. It also hides line numbers, displays the gutter and the folding margin (for views that support folding). All the text styles are set to the default values (equivalent to opening the Font dialog and using Reset All) and any zooming is removed.

In a Grid view, the grid is set to a standard state as if it had just been created. The grid data is preserved.

Show/Hide Channel

This dialog sets the channel list to display in time, result and XY views. It also controls the display of axes, grids and the horizontal scroll bar in time and result windows. You can resize the dialog vertically to make room for long channels lists and you can filter the channel list to show a sub-set of channels.



Channel list and filtering

The left-hand half of the dialog lists the channels in the same order as they are displayed in the view. If there are more channels than will fit in the visible area, you can either scroll the channel list to see them all or resize the dialog by dragging the size box or the top or bottom edge of the dialog.

The channels are identified by a text string that includes the channel number, title and type. Views can hold hundreds of channels, so there is a filtering mechanism to reduce the displayed channels based on matching text in the channel list. To apply a filter, type text to match into the box to the right of the Filter check box. As you type, the channel list changes to only show channels where the filter text is found in the channel identification string. The matching is case sensitive and uses ECMAScript regular expressions. Clear the Filter check box to disable filtering.

There is a drop-down list of up to 20 previously used filters available. Filter strings are added to the list each time you change the state of a check box with an active filter. There are separate filters for Time, Result and XY views. The strings are stored in the registry (`HKEY_CURRENT_USER\Software\CED\Spike2\Settings` in the keys: `Time chan filter`, `Result chan filter` and `XY chan filter`). They are read each time the dialog opens and saved when the dialog closes. You could manipulate the keys with the script `Profile()` command.

You can `Ctrl`+click the Filter check box to add or remove the filter text from the drop-down list. If the text is not in the list it is added, if it is already in the list it is removed.

Check boxes, Show and Hide buttons

There is a check-box to the left of each channel identifier string. When you open the dialog, the box is checked if the channel is visible, unchecked if it is hidden. You can check or clear the boxes one at a time, or you can click and drag a range and then click a box to set the check state for all selected channels. You can also use the Show or Hide buttons to check or clear all the listed channels. These actions do not have any effect until you click the Draw or OK buttons.

All Channels

The All Channels check box gives you information about the overall channel list (including channels that are filtered out of the displayed list). If all channels are marked for display the box is checked, if none are marked for display the box is clear. If some are marked for display and some are not, the box is filled in (as in the image). You can click the box to set the state of all the channels (even with a filter set).

Draw

Most changes made in this dialog will require the associated data view to redraw, which can take a noticeable time. Because of this, changes are not applied until you close the dialog with the OK or Listed button. You can apply changes without closing the dialog with the Draw button. This button is disabled if the view settings match the dialog.

Listed

This button makes all the listed channels visible (hiding any channels that are not in the filter) regardless of the state of the check boxes. It is equivalent to clicking the All Channels check box to remove all channels, then clicking the Show button and then OK. The idea behind this is that if you have a saved filter setting to display channels that match a specific filter, you can open the dialog, select the filter string, then use Listed to display a new set of channels. If you prefer keyboard commands to using the mouse, you can perform this operation from the data view with:

`Ctrl+H`	Open the Channel Show/Hide dialog
`Tab`	Make the Filter text the current item
`Alt+Down`	Drop the list of filters
`Down`	Use the down arrow key to move to the desired filter text
`Alt+L`	Display only the channels in the list and close the dialog

Y Axis, X Axis, Grid

You can show or hide the axes and you have control over how the x and y axes are drawn. You can hide or display the grid, numbers on the axes, the big and small ticks and the axis title and axis units. You can also choose to show the y axis on the right of the data, rather than on the left and choose if the Scroll bar is visible for the x axis (not in an XY view). You can change the width of the lines used for the axis and ticks in the Edit menu Preferences Display tab.

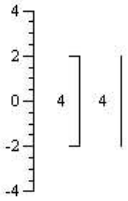
Title, Units, Small ticks

These check boxes allow you to show and hide various axis features. If you disable both the *Units* and the *Titles*, the vertical space used by the x axis reduces. The *Small ticks* are the sub-divisions between the axis labels.

In a Time view, the *Title* area is at the left-hand end of the x axis and is used in Time of Day mode to display the date at which the file started.

Scale bar

For publication purposes, it is sometimes preferable to display axes as a scale bar. If you check the Scale only box, a scale bar replaces the selected axis and any grid set for that axis will vanish. You can remove the end caps from the scale bar (leaving a line) by clearing the Small ticks check box. The size of the tick bar can set by the Large tick spacing option in the Y Axis Range or X Axis Range dialogs, or you can let Spike2 choose a suitable size for you.



A y axis will automatically switch to draw as a scale bar if there is not enough vertical space to display numbers for the tick marks.

Axis in data [10.07]

For publication purposes, you may prefer to display axes in the data area. If you check these boxes, the selected axis, or both, are drawn on top of any grid and under the data. The x axis is drawn along the line of 0 on the y axis. The y axis is drawn along the line of 0 on the x axis. Selecting these options does not turn off the standard x and y axes; you may wish to use these to adjust the display before turning them off. We expect these additional axes to be used with XY views. We do not expect these extra axes to be useful in a Time view; they may have a use in a Result view.

Y axis horizontal labels

Normally, the Title and Unit text for the y axis is presented vertically unless there is very little vertical space available. If you check Enable, the y axis text is presented horizontally and the other controls in this region become active. The channel title is left aligned when the axis is on the left and right aligned when the axis is on the right. The script language equivalent of this dialog section is `YAxisMode()`. The other controls are:

Centre units

If checked, the units are centred on the channel title. If clear, the units are aligned to match the title.

Label space

You can set the horizontal character space to reserve for the channel title and units. This is in terms of a representative character width, currently in the range 2-33. If you set more than 17 characters, any saved resource with this value read into a version of Spike2 before 8.12 will set a different width in the range 2-17.

Vertical alignment

You can choose to position the text at the top, centre or bottom of the available vertical space.

Channel numbers

The three check boxes at the top of the dialog control if channel numbers and the x and y axes are drawn. Channel numbers are never drawn in an XY view as all channels share the same y axis.

Short cut to display a single channel or channel group

In a time or result view, you can double-click a channel with a y axis to hide all the other channels; the channel expands to fill the entire window. A second double-click restores the display. `Ctrl`+double-click displays the channel plus all duplicates. If the clicked channel holds a group of channels, all channels in the group expand to use the display area and if `Ctrl` is held down, duplicates of all channels in the group are displayed.

Filter regular expressions

Regular expressions are too huge a topic to cover in detail here. When used with the channel dialog you will likely only use a small subset of the available possibilities. This following is just to get you started.

The first thing is that most characters stand for themselves and match themselves. So cat matches the middle letters of s**cat**ty, which is just what you would expect. However, there are several special characters that do not match themselves (unless you put a backslash before them to remove their specialness). You can also use `\t` to match a tab character. You can use `\d` to match a single decimal digit.

`.` Match any single character except a newline

`[`*chars*`]` Match a single character that is in the list of characters *chars*. The list can include a range as `a-z`, for example. To include - in the list put it first or last. `[0-9]` matches any numeric digit.

`[^`*chars*`]` Match a single characters that is not in the list of characters.

`(`*exp*`)` Parentheses enclose a "capture group" and gives it a number (starting with 1). You can then back-refer to the group: `(cat)\1` matches `catcat`. To use `(` and `)` as ordinary characters you must use `\(` and `\)`.

`*` Matches the preceding element 0 or more times. So `X*Y` matches `Y`, `XY`, `XXY`, `XXXY` and so on. Use `\*` to match a `*`

`+` Matches the preceding element 1 or more times. So `X+Y` matches `XY`, `XXY`, `XXXY` and so on. Use `\+` to match `+`.

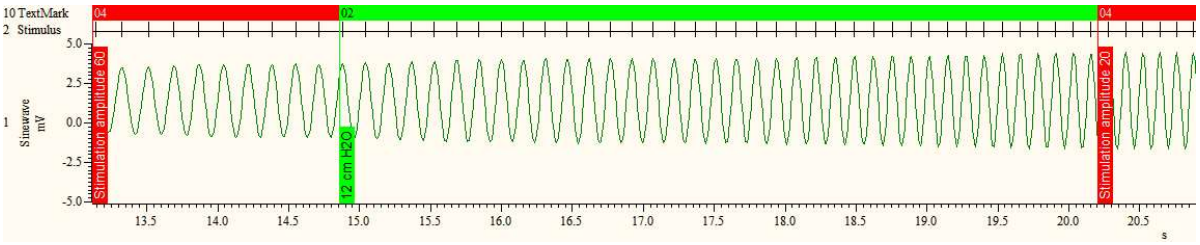
? Matches the preceding element 0 or 1 times.

| Matches either the regular expression to the left or the expression to the right. `cat|dog` will match either `cat` or `dog`. You can use parentheses to limit the range of matching: `gr(a|e)y` matches `grey` or `gray`.

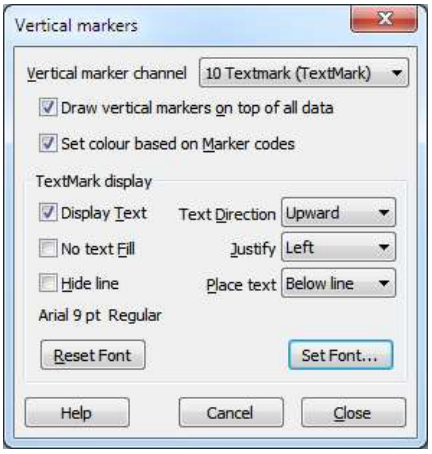
If you want the full details of regular expression searches, start here.

Vertical Markers

The View menu Vertical Marker command is available in a Time view and opens the Vertical Markers dialog. Vertical markers are vertical lines drawn under or on top of the channel displays, rather like vertical cursors, expect that their positions are set by the times of event or marker data points on a channel. If the channel is a TextMark, you have the option of displaying the text of the marker in a variety of styles. Vertical markers are not drawn if the display is in 3D overdraw mode, nor are they drawn over the Overdraw WM area of a Time view.



The picture shows an example of the use of vertical markers with a TextMark channel being drawn in both state mode and as a vertical marker. Vertical markers are often used in place of vertical cursors where you need a marker that cannot be moved by the user. If you use a memory channel as the vertical marker channel, you can edit the position by adding and removing items in the memory channel.



You can set the width of the vertical line in the Edit menu Preferences dialog and the default colour in the View menu Set Colours dialog in the Application Colours section. The dialog fields are:

Vertical marker channel

You can select any channel that holds event, Marker, RealMark, WaveMark or TextMark data. You can also select No channel to disable the display of vertical markers.

Draw vertical markers on top of all data

Check this box to draw the vertical markers after the channel data. Leave the box clear to draw it before the channel data. Leaving this box unchecked places the markers on top of any background (including channel background colours and grids) but under the data. Drawing is faster if this box is checked if you have grids or individual channel background colours defined.

Set colour based on Marker codes

There is a set of 9 colours in the colour palette that are used to display WaveMark data in colours based on the marker code. These colours can also be used for vertical markers if the marker channel is based on a Marker data type. Check the box to enable the use of WaveMark colours.

TextMark display

The remainder of the dialog fields are concerned with TextMark data. If the channel is not of TextMark type all the remaining fields are disabled unless No channel is selected, when all fields can be set.

Display Text

Check this box to display the text stored in a TextMark data channel.

No text Fill
Normally, the area behind the text is drawn in the colour set for the vertical marker and the text is drawn in white or black, whichever provides the most contrast. Check this box to draw the text in the vertical marker colour with no background fill.

Hide line
When text is drawn, you can choose to hide the vertical line by checking this box.

Text Direction
You can choose to run the text upwards (to match the text for channel labels on channels with a y axis) or downwards.

Justify
The text can be Left, Centre or Right justified within the scrollable area of the view.

Place text
The text can be aligned to be Below, On or Above the vertical marker line.

Set Font...
Click this button to open a Font select dialog where you can choose the font to use for the vertical marker text. The current font, size and style displays above the Reset Font button. The font size may be slightly different from the requested size as it displays the actual size of the font as rendered on screen.

Reset Font
Click this button to revert the vertical marker font to the font set for the view. The text above this button will show Use font set for view if the font has been reset.

Cancel
The view changes dynamically to show the effects of any changes. If you close the dialog with the Cancel button, all changes made are removed before the dialog closes.

Close
This closes the dialog, accepting any changes. If you have made a change and recording is enabled, script commands are saved that reflect the new state. You can use Undo to remove changes after the dialog closes.

TextMark List


The standard way to display TextMark data is in Dots mode, and comments are shown in the file as a small rectangle. The colour of the rectangle depends on the first marker code associated with the marker. If the code is 00, the rectangle is yellow. Otherwise, codes use the same colours as WaveMark data, set by the View menu Change Colours command. Other modes are available, such as Text and State. You can see the text associated with a TextMark by moving the mouse pointer over the rectangle and waiting a moment. The text appears as a tool tip. Any movement of the mouse pointer hides the text.

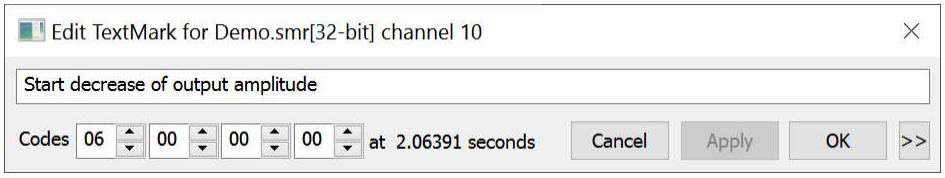
Opening the TextMark list dialog
You can open the TextMark dialog in several ways:

- With the View menu TextMark List command. If there are multiple TextMark channels this opens the lowest-numbered selected channel, otherwise the lowest-numbered channel.
- By double-clicking on a TextMark data item. The clicked-item is selected in the dialog. If you hold down the `Ctrl` key when you double-click, the dialog opens with the list area displayed, otherwise it opens with the list hidden.

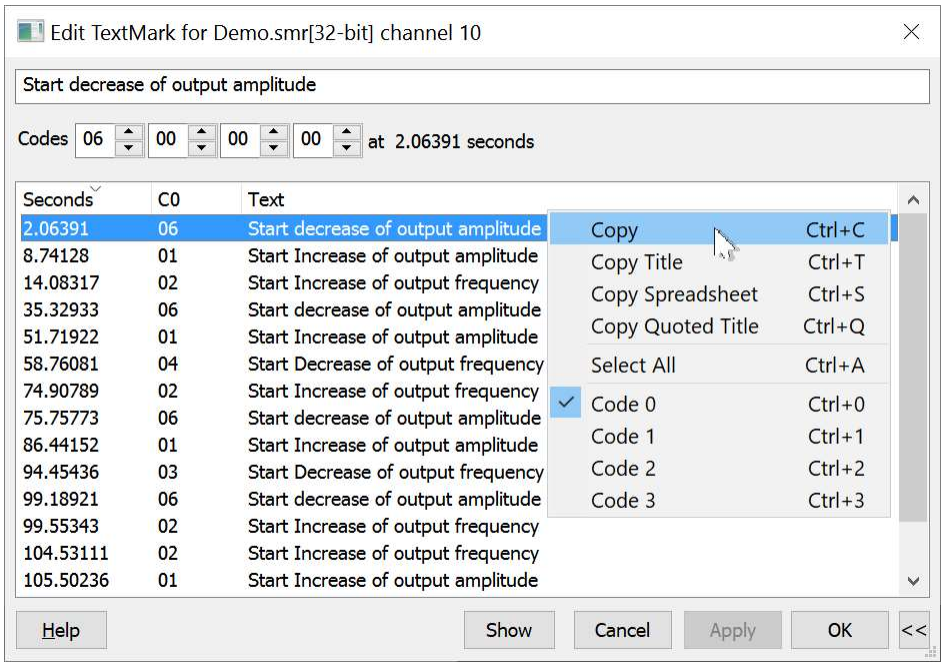
- By right-clicking on a TextMark channel and selecting View TextMarks... from the context menu. The nearest item is selected.
- By clicking on the system toolbar TextMark button or using the `Ctrl+T` keyboard shortcut when not sampling data. If you are sampling data, this opens the online Create a TextMark dialog.

Editing and reviewing the marker text

To edit the comment text and marker codes, use one of the methods above to open the Edit TextMark dialog. If your open method selected a marker or right-clicked on a TextMark channel, the selected marker or the nearest marker to the click is displayed. Otherwise, the first marker from the start of the displayed data is selected.



The >> and << buttons at the bottom right of the dialog show and hide the list area.



Double-click an item in the list to edit it and save any changes to the item that was previously displayed. The Show button sets the time view time range to span the currently selected items. If you have made multiple selections, the first marker is displayed at the left edge of the screen and the last one at the right edge.

The marker codes display in the format set for the source channel (Hex only or using printing characters).

Right-click in the list to control how many marker codes to display and to copy data to the clipboard. If you choose Copy for spreadsheet, text strings and marker codes are enclosed in quotation marks so that the data will import easily into spreadsheet programs. You can change the sort order of the data in the list by clicking the column titles. Initially, the items are sorted based on the first column (holding the time of each item).

The list area displays the same markers as the channel does, depending on the current settings of the Marker filter for the channel. You can open the Marker Filter dialog from this dialog by double-clicking on Codes (to the left of the four marker codes). The time of the items displays in the same format as is set for the x axis of the associated Time view. The number of decimal points displayed for the seconds depends on the tick spacing in the x axis at the time the list was last updated; the more you zoom in, the more decimal places. The list updates if the data changes or if you change the x axis units (seconds, milliseconds, hh:mm:ss, Time of day) or if you change the associated time view x axis width so that the number of decimal places changes.

Modifying the selected item

If you click on an item in the list, the text and codes are displayed in the upper part of the dialog. You can edit the text and codes, however nothing is saved until you click the Apply button or open the dialog for a different TextMark channel in the same file. Initially, the codes display in the format set for the channel. However, you

can change the code if it is in the 'printable character' range to two hexadecimal digits or one printing character by right-clicking on the marker code field and choosing the appropriate setting.

Navigating to TextMark data

You can move the associated Time view to the currently selected TextMark in the list by clicking on Show. If a single TextMark item is selected, the Time view stays the same width but scrolls to get the selected time as close to the centre of the window as possible. If multiple markers are selected, the display changes to start at the earliest marker and end at the latest. If you double click on an item in the list, this is equivalent to single-clicking it to select, then clicking the Show button.

If the dialog is open you can navigate to an item in the list by a single left-click on a TextMark data item in the time view window.

Context menu commands

Right-click on the list of comments to see the context menu. All items have a keyboard shortcut listed in the menu. The options are:

Copy, Copy Spreadsheet

These options copy the selected items in the list to the clipboard with each field (column) separated by a `Tab` character. If all items are selected this also copies the column headings. The *Spreadsheet* option places quotation marks around each field to group the field data (this should stop Spreadsheets attempting to identify marker codes as numbers, to avoid translating code `00` to `0`, for example.

Copy Title, Copy Quoted Title

These options copy the title line from the list, or a version with each column title in quote marks.

Select All

Selects all items in the list.

Code 0..3

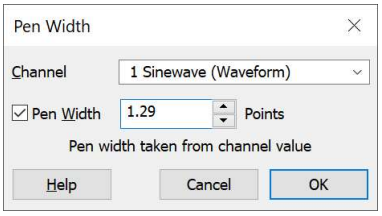
These options toggle the visible state of the columns holing marker codes 0 to 3. If you want to change the state of more than one item it is quicker to use the keyboard short-cuts.

Large numbers of TextMark items

The list box is limited to 12000 items, centred on the time of the item double clicked to open the dialog. If you work with a large number of TextMark items, avoid opening the file across a network; network access is typically much slower than local file access and some operations, such as sorting the TextMark data items (by clicking on the column titles in the dialog) can be slow on large, networked files.

Pen Width

The Pen Width dialog in a time or result view allows you to override the standard pen width set for data in all channels in the Edit menu Preferences dialog. The dialog can be opened as a context menu when right-clicking on a channel, or from the View menu. Changes made with this dialog can be undone and recorded as a script. The script language equivalent of this dialog is the `ChanPenWidth()` command. For XY views, use the XY Draw Mode dialog to set the pen width of a channel.



Channel

You can select a channel from the drop-down list or type in a channel or a list of channels. If more than one channel is set, the displayed information will be for the first channel in the list.

Pen Width

If you check the box, you can set a pen width in points; this will be applied to all the channels in the list when you click OK or when you use the spin control. If you clear the box, the standard pen width for data (set in the Edit menu Preferences Display tab) will be applied on OK and you cannot edit the value. A point is 1/72 of an inch, which used to be about same size as a screen pixel, but on a high-resolution display might be several

pixels. If you set a size of 0, the pen size will be 1 pixel on all devices (which can be a very thin on a printer or high-resolution screen). You can set a fractional point size, but the output pen will always be at least 1 pixel wide.

The spin control changes the points size in screen pixel increments and applies the changed value immediately. The text below the Pen Width field indicates the source of the pen width: *Pen width taken from Edit menu Preferences* or *Pen width taken from channel value*.

If you edit the pen width value, it is applied on OK.

File Information (Time View)

This displays information about the current time view, including five lines of comments and the time and date it was created (if available). You can edit the comments unless the file is open in read only mode. You can cause this window to open automatically when sampling ends from the Automation tab of the Sampling Configuration dialog.

New format .smrx files allow 8 comments internally, but we only display the first 5 to be compatible with the old format files. We are reserving use of the remaining comments for future extensions.

Originally, comments were limited to 100 characters. From version 8.09 we allowed much longer comments to be used in `smrx` files (up to 2000) characters. The intent is to allow more information to be stored, and this is likely to be useful to script users. This is an experimental change and the dialog has not been modified to accommodate huge comments. We may make provision for easy editing of multi-line comments in a future Spike2 version.

File Information (Result View)

This command displays information about the current result view window including the number of channels, bins and bin width. In particular, it displays the number of sweeps that have been added into a PSTH, correlation or waveform average, the number of data blocks in a power spectrum, the number of cycles in a phase histogram and the number of intervals that have been processed to build an interval histogram (including intervals that fell outside the histogram).

Channel Information...

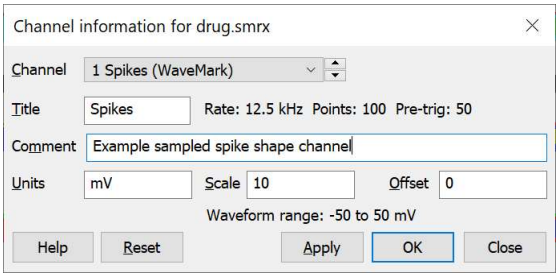
Channel Information (Time view)

Channel Information (Result view)

Channel Information (XY view)

Channel Information (Time view)

Use this dialog to view and edit time view channel information. You can open it with a double-click on a channel title, from the View menu or right click the channel to open the context menu. You can edit the Title and Comment of the channel set by the Channel field. The remaining fields are hidden or displayed depending on the channel type. The Reset, Apply and OK buttons are disabled until you make a change to one of the fields. The Close button closes the dialog and does not apply any changes.



Changes made in this dialog have no effect on your data until you click the Apply button or OK (which is the same as Apply then Close). Changes to all fields except Title apply to all duplicates of the channel and to the channel and its duplicates in any window duplicated from the current window. If you change channel without applying changes, any changes are lost.

Changes to the Title field are applied to duplicate channels if the duplicate has not been given its own title. If you set

the title of a duplicate channel, it has no effect on any other channel. If you clear the title of a duplicate channel, it takes the title of the channel from which it was duplicated.

The Reset button restores any changes you have made to the channel settings unless you have used Apply. To undo applied changes, close the dialog and use the Edit menu Undo command (`Ctrl+Z`).

From version 8.09 we have allowed much longer comments to be used (up to 2000) characters. The intent is to allow more information to be stored, and this is likely to be useful to script users. This is an experimental change and the dialog has not been modified to accommodate huge comments. We may make provision for easy editing of multi-line comments in a future Spike2 version.

Scale, Offset and Units

For waveform, RealWave or WaveMark channels, extra fields show the Scale and Offset values and the channel user Units. The scale and offset convert between the 16-bit integers used to store waveform and WaveMark data and user units. They also convert RealWave values to integers, when required.

Real value in user units = 16-bit value * Scale / 6553.6 + Offset
Integer value = (Real value in user units - Offset) * 6553.6 / Scale

When the Scale and Offset fields are present, the Waveform range field displays the range of values that a 16-bit waveform channel could span. You can read more about scaling in the documentation for the sampling configuration dialog. Both the scale and the offset must be smaller than 10 billion. The scale may not be set to 0 or very close to 0. If the channel scale or offset is edited into an illegal state, a warning message appears in the dialog and you cannot Apply the changes. If you want to calibrate a channel, it is often easier to use the Calibrate option of the Analysis menu.

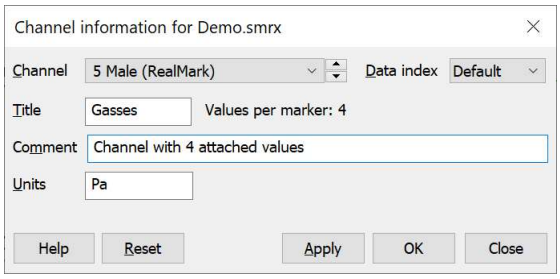
The Scale value is numerically the same as the value in the Sampling configuration when data is captured by a 1401 interface that has a ±5 Volt input range. If your unit has a ±10 Volt range, the scale value is double that in the Sampling configuration.

If the Scale and Offset fields are disabled and "No rescale" appears in the channel information to the right of the Title field, you have applied a channel process that has changed the calibration or is non-linear. At the time of writing, Rectify and Fill Gaps are both non-linear processes. To change the channel scaling, remove the process.

If you want to take advantage of SI prefixes for your channel units, see the Y axis dialog to enable this feature.

Data index (RealMark channels) [9.02]

RealMark channels have an extra field (Data index) to choose which set of titles and units are displayed. If there are n attached data items, you can choose from Default or Item 0 to n-1. The Default titles and units apply to all data indices that have no specific title or units set (are blank). When stored in the data file, the various titles and units are stored as: `"Default|Item 0|Item 1|..."`. This means you cannot use a vertical bar character in RealMark channel titles or units.



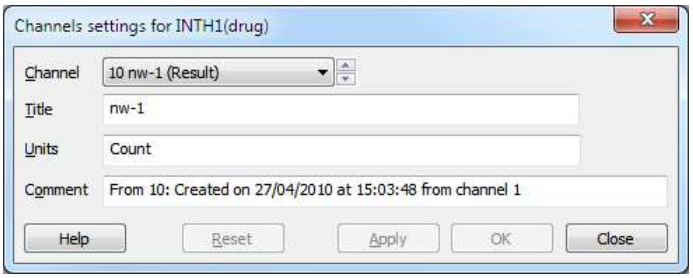
If this is a duplicate channel, you can also choose Duplicate (which allows you to edit the Title that belongs to the duplicate), otherwise the Title and Units fields are disabled. You must select the original channel to edit these fields. Once a duplicated channel has a title set, this over-rides any per item title until it is cleared. To clear the title, delete all characters in the field and click Apply.

This field does not change the displayed data index; use the Draw mode dialog Data index field to do this. The script language sets separate titles and units with the `ChanTitle$()` and `ChanUnits$()` commands with the extra `index%` argument.

Rescale

The Rescale button appears for RealWave channels. Click it to set the Scale and Offset fields so that the full range of data could be represented by 16-bit data. The offset is set to 0 if this does not lose too much precision. Some routines in Spike2 treat RealWave data as 16-bit integer, and the scale and offset you set determine how the conversion from 32-bit real to integer is done.

Channel Information (Result view)



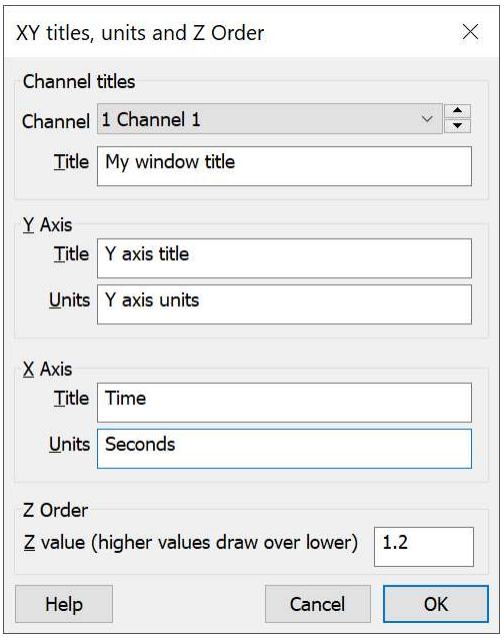
Use this dialog to view and edit result view channel information. You can open it with a double-click on a channel title, from the View menu or right click the channel to open the context menu. You can edit the Title, Units and Comment of the channel set by the Channel field. The Reset, Apply and OK buttons are disabled until you make a change to one of the fields. The Close button closes the dialog and does not apply any changes.

Changes made in this dialog have no effect on your data until you click the Apply button or OK (which is the same as Apply then Close). Changes to all fields except Title apply to all duplicates of the channel and to the channel and its duplicates in any window duplicated from the current window. If you change channel without applying changes, any changes are lost.

Changes to the Title field are applied to duplicate channels if the duplicate has not been given its own title. If you set the title of a duplicate channel, it has no effect on any other channel. If you clear the title of a duplicate channel, it takes the title of the channel from which it was duplicated.

The Reset button restores any changes you have made to the channel settings unless you have used Apply. To undo applied changes, close the dialog and use the Edit menu Undo command (`Ctrl+Z`).

Channel Information (XY view)



Use this dialog to modify the XY channel titles and the X and Y axis titles and units. You can open the dialog from the View menu, by double-clicking on the axis title and units area or from the XY view context menu. The OK button is disabled until you make a change. Changes are not applied until you click OK, at which point all the changes you have made are applied. If you click the Cancel button, no changes are applied. Changes you make in this dialog can be recorded and can be undone.

Channel titles

Each channel in the XY view can be given a title. If a channel does not have a title, it is listed as Channel n, where n is the channel number. You can select the channel from the drop down list, or by using the spinner control. All changes you make are saved in the dialog until you click OK or press the `Enter` key. We allow you to enter up to 50 characters. Channel titles are visible in the Key. The script language equivalent of this field is `ChanTitle$()`.

Y Axis

An XY view draws multiple traces using the same set of x and y axes. The displayed y axis title and units do not belong to any particular channel, so (unlike Time and Result views), the y axis title and units are independent of the channels. You can set titles and units of up to 50 characters, though no more than 10 will display. When a result view is saved, the text strings for the x and y axis titles and units have to fit in a restricted space (for reasons of backwards compatibility). There is sufficient space for 50 ASCII characters, but if you use non-ASCII characters you will have room for fewer. For example, there is typically space for 17 Japanese or Chinese characters. The script language equivalents are `ChanTitle$(0, ...)` to set and get the title and `ChanUnits$(0, ...)` for the units.

X Axis

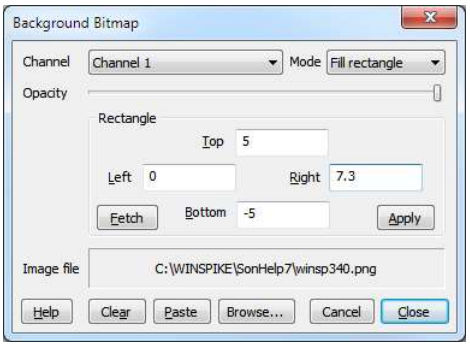
You have a free choice of text for the x axis title and units in an XY view. The same comments about the length of the text apply as for the Y Axis. The script language equivalents are `XTitle$()` and `XUnits$()`.

Z Order

From Spike2 version [10.17] you can control the order in which channels are drawn (not to be confused with the order that data is drawn within a channel, which is set by the XY Draw Mode dialog). Each channel has a Z value and channels are drawn starting with the lowest Z value. In the event of a tie, lower numbered channels draw before higher numbered channels. When a channel is created, it has a Z value of 0.0; you can set negative or positive values. This value has most effect when a channel is drawn in a filled mode, so channels drawn later can mask channels drawn earlier. The script language equivalent of this field is `XYZOrder()`.

The View menu Standard Display command resets all the Z values to 0.

Channel Image



You can set a bitmap as the background to any channel in a Time, Result or XY view. This can be very useful in an XY view when the XY view is being used to map activity. For example, when the view is showing where spiking activity occurred in a maze, or when tracking a target. This can also be used to display the result of the `ArrMapImage()` command. The menu command opens a dialog:

The dialog is used to select a channel and then link the channel to a bitmap file on disk. Any channel in the view can be associated with an image. The name of the file appears in the Image file section of the dialog. The image is rendered on top of the channel background colour, but below all other items that are drawn. When channels are overdrawn, the image appears when the background colour of the channel would be drawn. The dialog fields are:

Channel

You can select any channel in the current view. However, if you are in an XY view, all channels share the background, so the choice of channel makes no difference. When you change channel, any Rectangle region changes that have not been applied are lost and the current settings for the new channel appear in the dialog.

Mode

There are three mode settings: No display, Fill Background and Fill Rectangle.

No display — No image is displayed. This allows you to have an image loaded, ready to display. This will save the time needed to load the image from a disk file.

Fill Background — The image is scaled so that it fills the available channel rectangle. You would use this when you want to display the entire image as a background for the channel. Scrolling a view with an image in this mode will cause flickering as the image must be redrawn for every movement. We disable background images in Fill Background mode when sampling or rerunning.

Fill Rectangle — The image is scaled so that it fills a rectangle defined in x and y axis units. You would use this when the image must be aligned with the axes. For example, if the x and y axes represented co-ordinates in a maze, the image could be a picture of the maze. There is no problem with scrolled displays in this mode.

Changes made to the mode field are applied immediately, so you can see the effect.

Opacity

This field is a slider control that you can drag to the left to make the image transparent and to the right to make it opaque. Changes made by dragging are applied immediately.

Rectangle

The fields within this area are disabled unless the mode field is set to Fill Rectangle. The Left, Right, Top and Bottom fields refer to the x axis and y axis co-ordinates of the image. In the current implementation, you cannot invert or reflect the image; if you do it will not appear.

Fetch
This button sets the Left, Right, Top and Bottom fields to the values that would make the image fill the channel background.

Apply
This button is enabled when in Fill Rectangle mode and the current rectangle settings are legal number and do not match those of the channel. Click the button to apply the new values.

Browse...
Click this button to open a file dialog in which you can browse for a suitable image file. You can choose between windows bitmaps, JPEG and GIF images, PNG files and TIFF files.

Paste
This button is enabled if the clipboard holds a bitmap image. Click to paste the image to the current channel. The Image file name will display as `<CB>`. Setting the clipboard as the source can have unexpected results as each time a saved window opens it will copy whatever image happens to be on the clipboard. Clipboard images are more useful to scripts, for example to copy a multimedia view video frame using `EditCopy()`, then setting the image with `ChanImage()`, avoiding the need to write the image to a disk file.

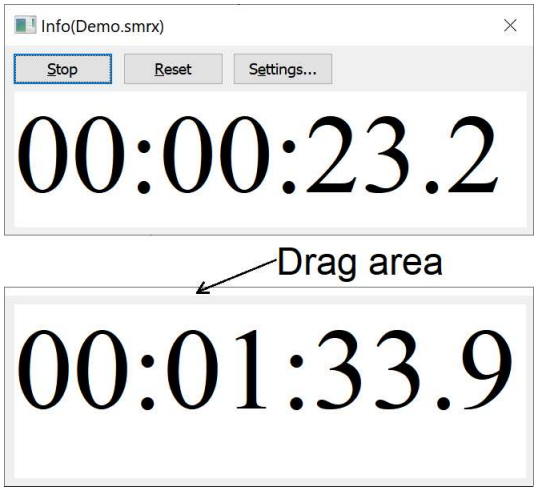
Clear
Click this button to clear any image set for the channel. This releases any resources used to hold the bitmap.

Cancel
Click this button to close the dialog and undo any changes made since the last channel change.

Close
Close the dialog, leaving the associated channel in the current state.

This command is experimental and is implemented in a simplistic manner. If you reuse the same image for multiple channels we do not notice, and store the image multiple times, which could be a problem if you use huge images on many channels.

Info windows



The View menu Info windows command display a pop-up menu that allows you to create a new Info window attached to the current Time, Result or XY view, or to select an existing Info window belonging to the current view and bring it to the front. Info windows were added to Spike2 at version [10.00].

Each Info window owns a timer, a user-defined text string, an optional background image and has configurable text and background colours and font. You can hide the window title bar and/or the buttons to reduce screen clutter. With the title bar hidden, a small area remains at the top of the window for dragging.

The text string (see the Settings dialog) usually contains one or more special fields that are replaced by a timer value or by other data extracted from the associated data view and updated at user-defined intervals. You can resize the window, and the window contents automatically scale to fill the available space. This makes Info windows useful for real-time work in busy experiments by providing easily read (from a distance) experimental information.

You can set the Info window to speak the displayed text. This can be useful in online situations where you are required to use a microscope or cannot see the screen while capturing data.

You can attach up to 10 Info windows to each data view.

If you close a view that owns Info window(s), the settings are preserved in the view resource file. When you open a view from a disk file we also restore any Info windows that were saved.

In addition to the interactive use described here, there are script language commands to create, configure, control and delete Info windows.

Create New Window

Use the Create New Window sub-command to this command to create a new information window. The newly created information window is attached to the current time, result or XY view. Other sub-commands can be used to select an already existing information window.

Using the information window

You can resize the information window by dragging the bottom-right corner of the window or any edge; the font size adjusts according to the window size so that the window text fills the window unless the Lock font size option is active.

Button bar commands

These commands are also available from the right-click context menu.

Start	Visible and enabled when the timer expression (`%t`) is part of the window text and the time is not running. Click to start the timer.
Stop	Replaces the Start button when the timer runs. Click to stop the timer.
Reset	Resets the timer. The timer starts if you have set the Settings dialog Start on reset option.
Settings	Opens the Info Settings dialog where you set the text string and configure the timer. You can hide this button with an option in the Info Settings dialog.

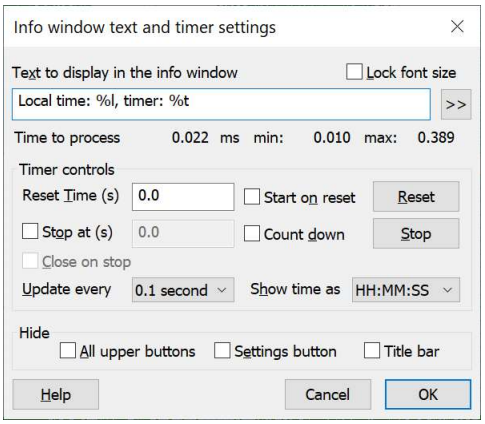
Context menu commands

Other options are provided by a menu obtained by right-clicking on the window. The context menu commands are:

Copy	Copy the displayed information window text to the clipboard.
Close	Close the information window.
Start/Stop	Start or stop the clock or timer. This is the same action as the Start/Stop button at the top of the window.
Reset	Resets the clock or timer. This is the same action as the Reset button at the top of the window.
Deblock	Re-enable timed-out expressions (fields that display {time}).
Settings...	Open the information window settings dialog.
Colours...	Open the colour selection dialog to change the colours for this information window. The main colour selection dialog can also be used to set the colours for all information windows or all those associated with the current parent view.
Font...	Open the font selection dialog to change the information window font. Note that the font size is set automatically according to the size of the window.
Image...	Open the Info Image dialog allowing you to set an image that will be displayed in the information window.
Speak...	Opens the Info Speak settings dialog to configure speech output of the Info window contents.
Title visible	Check the box to display a title bar at the top of the information window.
Buttons visible	Check the box to display the Command buttons. The same commands are available from this pop-up menu.
Lock font size	Check the box to fix the Info window display font at the current size.

See also: Toolbar and Status bar, Enlarge View Reduce View, Y Axis Range, X Axis Range, X Axis Extra Time, Standard Display, Show/Hide Channel, Vertical Markers, TextMark List, Pen Width, File Information (Time View), File Information (Result View), Channel Information..., Channel Image, XY Key Options, XY Autoscale, Trigger/Overdraw, Channel Draw Mode, Multimedia files, Spike Monitor, Font, Use Colour and Use Black And White, Change Colours, Colour scale dialog (Sonogram, Density map colours), Folding, Show Gutter, Show Line Numbers, ReRun, Annotate, Grid view commands

Info Settings dialog



The Info window settings dialog configures the text to process to generate the display, controls the Info timer and can hide unwanted items from the window. Any changes you make in this dialog are applied immediately. The script language equivalents of this dialog is the `InfoSettings()` command or the `InfoOpen()` command to create a new window. The window is divided into several regions:

Text to display in the info window

At the top you type in the text that generates the Info window display. You can make this text span multiple lines by including the sequence `\n` to add a line break. This text is usually scaled to fit into the Info window rectangle; the bigger the window, the larger the text. You can defeat this scaling by checking the Lock font size box.

You can embed special sequences in the text that are replaced with values based on the current time of day, a controllable timer and measurements made from the associated Time, Result or XY view using Dialog expressions. You can find the full details of the Text to display... field below. The >> button opens a menu of commonly-used special sequences.

Timer controls

In addition there are a number of settings to control how the timer value is displayed and how the timer behaves. The timer can be reset manually (with the Reset button) or automatically during sampling when a new sweep starts (for a file view parent) or when new data is processed into the memory view (for memory or XY view parents). The timer can be started and stopped using the Start/Stop button, or automatically on a reset. The timer controls available in the dialog are:

Reset time (s)	Sets the time in seconds to load into the timer on a reset.
Start on reset	Check the box to start timers automatically (if stopped) on a reset.
Count time downwards	Check the box to run timers backwards.
Stop at (s)	Check the box to stop the timer at the time set in the field to the right of the check box. The time is in seconds. This box enables the Close on stop check box. If this box is not set, the timer will continue past the Stop at time.
Close on stop	If this box and the Stop at check box is set, the Info dialog closes when the counter stop condition is reached (as long as no dialogs opened from the Info window are open). This is not expected to have any use from the Settings dialog; it is intended for use when the Info window is opened from a script and you want it to display for a fixed period, then close (for example to display a message that disappears after a short period). This field was added at [10.15].
Update every	This selector sets how often the displayed time is updated and the smallest unit of time that is shown. You can choose between every 0.1 second, Second, Minute or Hour. For example, if you update every 0.1 seconds then the displayed time show tenths of a second. If you update every minute, the seconds value is not shown.
Show time as	Sets how times are displayed. You can choose from 0.1 seconds (a count of 100 millisecond intervals), Seconds (a count of seconds), `MM:SS` and `HH:MM:SS`. The

Update every selector also affects how the time is displayed - parts of the time display that are shorter than the Update every interval will not be shown.

The dialog displays error messages if any of these values are contradictory (e.g. counting down from 0 but want to stop at 1) or illegal and disables the OK button.

Hide

The Hide region allows you to conceal features of the Info window that you do not want the user to see. You can hide:

All upper buttons	Check this box to conceal the button bar in the Info window holding the Start/Stop, Reset and Settings... buttons.
Settings button	Check this to conceal the Settings... button. Checking the All... button disables this control.
Title bar	Check this to remove the Title bar from the Info window (usually used when displaying a pop-up message from a script).

OK and Cancel

The OK button is disabled if there is a problem with the current dialog settings. An error message drops down from the bottom of the dialog after a couple of seconds to explain what caused the error. Some errors, such as a syntax error in a dialog expression, are marked in the information window output as they are not detected by the parser in this dialog. The associated Info window is kept up to date with the contents of this dialog; clicking `OK` accepts the current state and closes the dialog.

The Cancel button undoes any changes you have made in the dialog and closes it.

See also: Info Image dialog, Info Speak dialog

Info window Text field

The text entry field is the template for the Info window display. You can embed special text fields that are replaced by measured data values. You can split the line of text into multiple lines for display by inserting `\n` into the text. Drawing text is relatively slow, so you will want to minimise the text in the window, particularly if you are updating often.

Special fields in the text are introduced by a `%` character, or enclosed in curly braces, for example `%t` or `{xC0}`. These fields are evaluated each time the window updates and the field is replaced by whatever value the evaluation generates or by `<reason?>` if it cannot be evaluated where `reason` indicates what failed when attempting to evaluate the special field. The two type of special field are:

% special fields

Fields introduced by a % are more efficient (take less time to process) than curly brace fields, but less flexible in use. All values in them (channel and point numbers) are provided as decimal numbers and are fixed. You cannot use symbolic channel names (`v1`, `m1` and so on) or refer to cursor numbers.

To make it easy to generate these fields, you can click on the >> button to build them interactively.

These fields are parsed (checked for validity) when you type them, but the check only verifies that the text matches the pattern of each command. View types, channel numbers and point indices are verified when the field is evaluated. You can find the full details of the fields below.

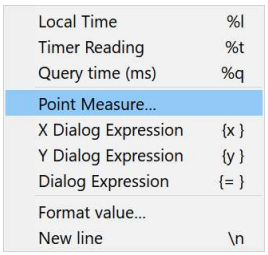
{x...}, {y...} and {=...} special fields

The text within these fields are passed to the dialog expression evaluator for the current view. These fields are parsed every time they are evaluated, which makes them slower to evaluate than % special fields. However, they have more data processing options and deal with x axis co-ordinates in x axis units, not in data points. To make it easy to generate these fields, you can click on the >> button to build them interactively. However, we have not automated the contents of the fields; you have to type in the dialog expressions yourself. You can find the full details of the fields below.

Format control

Unless you specify otherwise, Spike2 displays the measured values in a default format. You can change the format for the items marked with a 'y' in the Fmt column of the table by following the item immediately (no white space) with `"{flags}{width}{.precision}format"` where items in curly braces are optional and `format` is one of `f`, `F`, `g` or `G`. This is a restricted version of the script language format specifier for real values with the leading `%` omitted. If you do not specify a format, you get the equivalent of `"f"`, which is fixed point with 6 decimal places. To make this easy, the >> button leads to an option to insert a format interactively.

>> (insert special field)



The button opens a pop-up menu in which you can select items to insert into the text box, replacing the current selection. The items are grouped into three section: time related insertion, data measurement insertions and data formatting insertions. The insertions are either % special fields, { dialog expression } special fields, or format fields.

Format fields surrounded by double quotes (for example ".3f") and are recognised only when they immediately follow a special field that supports formatting with no white space between the final character of the special field and the " character that introduces the format field. If you insert a format field in a position where it is not recognised, or if the format text is not recognised as a valid format string, the text of the format field appears in the Info window output.

You can insert the following:

Local Time	The current time of day, formatted as `HH:MM:SS`.
Timer Reading	The current timer value, formatted as set in the Timer controls section.
Sweeps	Valid for Result views, the sweep count.
Query time (ms)	Time (in ms to 3 decimal places) to calculate the previous update.
Point Measure...	Opens the Measurement dialog to select a channel-related measurement.
X Dialog Expression	Inserts `{x }` and sets the text caret ready for you to type an expression.
Y Dialog Expression	Inserts `{y }` and sets the text caret ready for you to type an expression.
Dialog Expression	Inserts `{= }` and sets the text caret ready for you to type an expression.
Format value...	For use immediately after a special field. Opens the format dialog.
New line	Inserts the new line sequence `\n`

Time to process

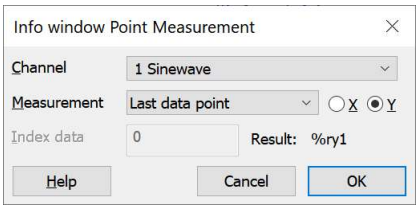
This line in the dialog displays the time taken, in milliseconds, to process the last update of the Info window and the minimum and maximum times taken since the last change to the text. The update time is the same value as displayed by the `%q` expression.

Each display update recalculates all the `%` and `{...}` expressions, which can take substantial time. For example, if you include:

```
{y Mean(0,Maxtime(),1)}
```

in a sampling time view to calculate the mean value of all the data in channel 1, this will take longer and longer to compute as time passes. Eventually it will take so long that the program will feel sluggish and unresponsive. We combat this by timing how long the computation of each dialog expression takes (these are usually the slowest items). If any item exceed 100 milliseconds, it is disabled and the output from that expression is replaced by {time}. You can re-enable the fields with the Info window right-click context menu Deblock command.

Info window Point Measurement



This Info Settings dialog >> button Point Measure... command opens this dialog which allows you to generate a % special field interactively. The current result of your dialog settings can be seen and the dialog will not let you generate syntactically incorrect fields. However, you can generate references to data points that do not exist. These will cause the text `<point?>` to appear in the output.

Channel

You can select any channel in the data file from the drop-down list of channels. If you want to use a symbolic channel number, such as v1, this cannot be done with % special fields. You must use the {...} dialog expression fields to do this.

Measurement

This field chooses the data item that you want to measure. You can choose from the following:

Last data point	The last data point on a channel. Usually used with an XY or a Time view.
Previous data point	The second to last data point in the channel.
Difference of last points	Use the final two points in the channel and subtract the value from the second to last from the value for the last.
Indexed point	The value at a particular indexed into the channel. The first point on a channel is index 1. Indices from 0 downwards are backwards from the end. In a time view you can only use indices from 0 downwards to -1999. In Result or XY views you can use forward or backward indices.
Last point - Indexed point	Subtract the value at a particular index from the last value on a channel.
Number of channel points	Valid and useful for an XY view channel. Valid for a Result view. This is not implemented for a Time view and generates `<view?>` if you use it.

Index data

Select the Indexed point or Last point - Indexed point measurements to enable this field. Positive values, from 1 upwards, index the channel point from the start of the channel. The index value 0 is the last point, -1 the one before the last and so on, indexing backwards from the end. In a time view, you may only index backwards from the end, and only up to 1999 points backwards (to limit the length of time spent searching through data). If you want to access your data by x axis value rather than by point number, you can do this using the { dialog expression } special fields.

X or Y

You can choose to measure either the X or Y value associated with a particular point on a channel. This field is not valid for the Number of channel points measure, and is disabled when you select this measure.

Result

This field displays the text that would be inserted if you clicked the OK button.

Cancel and OK

The Cancel button closes the dialog without inserting any text. The OK button closes the dialog and accepts the current value, which is inserted in the settings dialog to replace the current text selection. Invalid entries in the Index data field will disable OK.

% special field details

In the table, items in <angle> brackets stand for channel <ch> and point <pt> numbers. Channel numbers are the channel numbers displayed when you view the channels have values from 1 upwards. Fields with a numeric result and a yes in the Fmt column can be followed (with no intervening white space) by "<format>" to set a specific numeric format when they are expanded.

Point numbers can be used in some commands. The first point in a channel is 1, the second 2 and so on. Point numbers greater than 0 are generally only used with XY views. Point number 0 means the last point of a channel, -1 is the point before it and so on. Point numbers of 0 or less can be used with both XY views and Time views to access the most recently added data. Using point numbers much greater than 0 or with large negative values in a time view can be very slow to evaluate.

We detect the end of a number by finding a character that cannot be part of the number. This can lead to ambiguity if you want to follow a field by a numeric character. We allow you to enclose numbers is brackets, for example (10) to avoid any problem. So, %rx3 and %rx(3) are both acceptable as the rear x value of channel 3.

The fields you can set are (<ch> stands for a channel number, <pt> is a point index as described below):

Coded as	Fmt	Replaced by
%l		The local system time, displayed as HH:MM:SS.
%t		The current value of the view timer in the format set by the dialog.
%n<ch>		Replaced by the number of items in the channel (for use with XY views, also works with Result views).
%w		Result views only. Replaced by the sweep count for the result view.
%q		Query how long it took in milliseconds to evaluate the previous window update. If an update takes too long, we may reduce the update rate or even disable the display.
%rx<ch>	yes	The most recent/rear x value from the channel. Example: %rx1 get the last x position of channel 1.
%ry<ch>	yes	The most recent/rear y value from the channel. Example: %ry3 gets most recent channel 3 value. From version [11.00] if the channel is a Marker, the y value is the current marker code set for the channel, if the channel is a TextMark, the data is the marker text.
%px<ch>	yes	The penultimate x value from the channel. %px2 gets second last x position on channel 2.
%py<ch>	yes	The penultimate y value from the channel. %py20 gets the second to last value from channel 20. From version [11.00] if the channel is a Marker, the y value is the current marker code set for the channel, if the channel is a TextMark, the data is the marker text.
%dx<ch>	yes	The difference of the last two points x co-ordinates; last value - previous.
%dy<ch>	yes	The difference of the last two points y co-ordinates; last - previous.
%c<ch>px<pt>	yes	The x data at index <pt> on channel <ch>.
%c<ch>py<pt>	yes	The y data at index <pt> on channel <ch>.
%c<ch>rx<pt>	yes	The x position difference between the last point and the data at index <pt> on channel <ch>. That is (last x position - x position of <pt>).
%c<ch>ry<pt>	yes	The y value difference between the last point and the data at index <pt> on channel <ch>. That is (last y value - y value of <pt>).

You must use lower-case letters for the codes; if you use upper-case, the codes will not be recognised.

The <pt> field

The <pt> field is a 1-based point index into the channel. You can use index 0 to mean the last point, and negative points to refer to points before the last point. To stop the command taking too long, we limit the backwards indexes to -1999.

In an XY view, you can index into any of the points held in a channel. In a Result view, you can index to any of the data bins in the channel. In a time view, we currently allow only backwards indexing from the end, so <pt> values greater than 0 do not work. You can use dialog expressions (see below) to display values at any x value in a channel.

The <ch> field

The <ch> field is a numeric channel number. If you want to refer to virtual or memory channels you must use the numeric channel number, or use a dialog expression (below), which does understand channel numbers such as v1.

Errors

It is possible to type in % expressions that are syntactically correct, but not valid. If you do this, the result of evaluating the expression can be:

<view?> The expression is not valid in this owning view. For example, requesting the number of points in a Time view channel is not supported.

<channel?> The requested channel number does not exist.

<point?> The requested point does not exist or is not supported in the current view.

Syntax errors in the text results in a display up to the point where the error was detected.

Dialog expression fields

Dialog expression fields have the format:

```
{<type><optional spaces><dialog expression>}
```

The <type> is one of x, y or = and determines the type of the dialog expression that can be used. The text within the curly braces is not modified, other than replacing % fields, before being passed to the expression evaluator. Leading white space before the dialog expression is ignored. You can select between:

type	Fmt	Replaced by
{x...}	yes	The ... stands for an x-axis dialog expression. For example {x C1} is replaced by the position of cursor 1, expressed as a number.
{y...}	yes	The ... stands for a y-axis dialog expression. For example {y At(C1,2)} is replaced by the displayed y value of channel 2 at cursor 1.
{=...}	yes	The ... stands for a non-view numeric expression. You might use this to perform arithmetic on the results of % special fields. For example: {=Abs(%ry2)} is replaced by the absolute (rectified) value of the final point of channel 2.

The difference between these types comes down to the list of functions that are recognised at the top level of parsing. For example, C1 or Cursor(1) is allowed as an x-axis dialog expression, but not as a y-axis expression. However, At(C1,1) is allowed as a y axis expression as the cursor 1 reference is in a context where an x-axis value is required. Both x-axis and y-axis expressions can use any of the features of the non-view based numeric expressions.

The dialog expression evaluator parses the ... expression each time it is evaluated, which makes it slower than % expressions. The {y...} can be slow to evaluate, especially if the measurement requires scanning a range of data. Spike2 may disable an Info window if the evaluation takes a significant time as this might compromise data acquisition.

Dialog expression can be combined with operators, for example:

```
{y abs(At(C1,v1) - At(C2,v2))}
```

This computes the absolute value of the difference between the y values of the first virtual channel at cursor 1 and cursor 2.

You can enclose special fields introduced by % within curly brace special fields and these will be evaluated and converted to text before passing the curly brace field contents to the dialog expression evaluator. If you do this, you must consider the format you choose for the values. In a time view, x positions should be expressed to the time resolution of the time window, which is typically or order 1 microsecond, so the default format (equivalent to .6f) is probably OK. In other view types, you only need enough x axis resolution to locate the data items, so a lesser resolution may be suitable. The resolution for y positions can often be very different, and depends on the channel and measurement type.

```
{= 1/%dx1}
```

This calculates the reciprocal of the difference in x values of the last two points on channel 1.

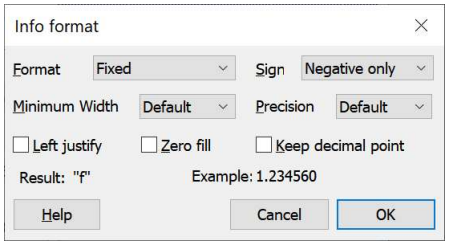
Dialog expressions can be followed by a format specifier.

Errors

If we fail to parse a dialog expression, the output in the Info windows shows the Dialog expression that failed and parsing stops after the failure. The effect of this is that as you type in a dialog expression, the output displays what you type until you type the terminating } at which point the expression is passed to the dialog expression evaluator. This will either succeed, in which case the result is the value, or it can fail and the output displays:

{=?}	The dialog expression evaluator could not evaluate the expression.
{time}	The expression was evaluated, but took too long be allowed to run repeatedly. You can allow it to run again with the context menu Deblock command or by changing the Info window source text.

Info format dialog



This dialog opens from the >> button of the Info Settings dialog. It is used to build a format expression of the form:

"<flags><width><.precision><format>"

where all the fields in angle brackets are optional except the format. This the Spike2 script language

Format expression are valid when they follow immediately after a % expression or a {...} dialog expression field (with no spaces between them) and they are syntactically correct. This dialog will not generate a bad format. The Result field displays the output with the current settings. The Example field shows how the number 1.23456 would be displayed with the current format. The OK button accepts the current state and inserts it, the Cancel button closes the dialog and does not insert it.

If you do not supply a format, the default is equivalent to a format specifier of "f".

Format

This sets the <format> part of the format expression and is a choice of:

	code	Description
Engineering	`e`	The value is expressed as a number with one digit in the range 1-9 before the decimal point followed by the number of decimal places set by the Precision field, followed by an exponent, such as e+03, being the power of ten to multiply the result by to obtain the value. Example: `1.234560e+00`
Fixed	`f`	A fixed number of decimal places, set by the Precision field. Example: `1.234560`
General	`g`	Either the `e` format or the `f` format, whichever is shorter. The Precision field sets the number of significant digits. Trailing 0's after the decimal point are removed unless the # flag (Keep trailing Zeros) option is used.

Sign

Negative values always start with a minus sign, but you have a choice of how we deal with positive numbers. The choice you make here adds a <flags> field character unless you choose Negative only:

	flag	Prefix for positive numbers
Negative only	`none`	None
Always	`+`	Plus sign, for example +1.234560

space or - `space` A space character. This is often used in Fixed format to align the decimal point for columns of numbers.

Minimum width

This sets the optional <width> part of the format. The default state (with no <width> field) is to use the minimum field width possible to satisfy the requirements of the other format fields, equivalent to setting a minimum width of 0. If this field sets a width that is more than that required by the rest of the format, the output is padded on the front (after any sign characters) with either space characters or 0's until the minimum width is achieved. We allow you to set a minimum with of up to 20 characters.

Left justify (-)

Check this box to add a - to the <flags> field. This forces the output of the format, before any space or 0 padding, to be left justified in the width field.

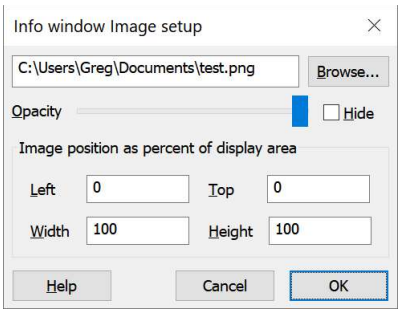
Zero fill (0)

When the Minimum width exceeds the required space, the extra space on the left is usually filled with spaces. Check this box to fill with 0s (after any sign character). This adds a 0 to the <flags> field.

Keep decimal point/Keep trailing zeros (#)

The name of this check box changes according to the selected format. In `e` and `f` format it preserves the decimal point, even when Precision is set to 0 (which normally removes it). In `g` format, it preserves trailing 0s. When checked, this adds # to the <flags> field.

Info Image dialog



You can display an image on top of the Info window background, behind the text that scales with the Info window. To make it easier to display text over the image we allow you to change the image opacity with a slider. There is also a control to hide the image. The image can be positioned anywhere within the Info window and scales with it, so you may need to add several new lines (`\n`) to the text window to make the output rectangle have a reasonable aspect ratio.

Dialog changes affect the display as you make them, so you can evaluate the effect.

The script language equivalent of this dialog is the `InfoImage()` command.

Hide image

Check this box for no image display. When script-controlled, this can be useful to show an image on an error condition.

Image file name and Browse...

You must select an image for display. The Browse... button opens a file selection dialog in which you can select a suitable image. This causes an immediate Info window update. We support the common bitmap file types: `BMP`, `PNG`, `JPEG`, and `TIFF`. If you edit the file name, the change happens when you click elsewhere in the dialog.

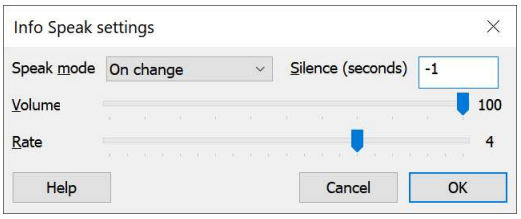
Opacity

The image draws under the text and can make the text difficult to read. You can make the text more or less transparent with the Opacity slider. Setting the slider control to the right sets an opaque image, moving it to the left merges the image into the background colour and setting the slider to the left results in an invisible image.

Image position
The values for the image left and top corners and for the width and height set the position of the image within the background area of the info window. All of these values are a percentage of the information window height and width, from 0.0 to 100.0.

OK and Cancel
The OK button accepts the current dialog state and closes the dialog. The Cancel button undoes all the changes made by the dialog and closes it.

Info Speak dialog



Open the Info Speak dialog from the Info window Speak... context menu (right-click). This dialog configures the Info window to output the displayed text as speech (if supported by your computer and sound card). You can choose when (and if) to output the text, the minimum time gap between speech utterances and the volume and rate of the output.

The `InfoSpeak()` script command has the same capabilities as this dialog.

Speak mode
You can choose one of the following settings from the drop-down list to choose when speech output should occur:

Off	Never. This is the default setting.
On change	When the output text changes.
Continuous	Start reading as soon as the previous output ends. Negative Silence values are treated as 0.
On change, repeat	When the output changes, and repeat after the period of silence.

Silence
Left to itself, the Microsoft speech output engine inserts around 1 second of silence after each utterance with the standard speech rate. This field lets you insert additional silence. When this field is set to a positive value, we wait for the speech output engine to tell us that the previous speech output has ended (including the built-in silence). We then insert the number of seconds of silence you have requested before looking for the next speech trigger.

You can set this field to -1 to mean: *do not wait for the previous utterance to end*. For example, if you want a 10 second countdown to be spoken (set the output text to `%t`, Update every to Second and Show time as to Seconds in the Settings dialog) you will find that with Silence set to 0, the speech plus built in gap takes too long to speak every number (10... 8... 6...). Set Speak mode to On change and Silence to -1 to achieve the full count.

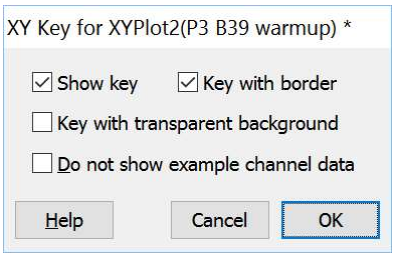
Note that in Continuous mode, negative values of Silence are treated as 0 (otherwise there is no output as the speech is continuously restarted).

Rate
This slider controls the speed of the speech. The central position is the standard speed (0), fully to the right (10) is approximately 4 times faster (just about understandable) and fully to the left (-10) is approximately 4 times slower (painfully slow). The Rate also affects the duration of the gap between utterances. On my Windows 10 PC, a Rate of 4 allows a ten second count down to work with a Silence value of 0, rather than -1, which is required for all Rate values less than 4.

Volume

This slider controls the output volume. Fully to the right (100) sets the loudest output, fully to the left (0) is no output. Of course, the actual volume will depend on your sound card (assuming you have one) and how it is connected to the loudspeakers. Please consult the documentation for your computer to connect up your sound card and to select a voice for speech.

XY Key Options



This command is for XY windows only and opens the XY Key options dialog. The command is available from the View menu and also by right-clicking in the XY view or by double-clicking on the XY Key. The dialog has a check box that controls the automatic expansion of the axes when new data is added. It also controls the XY window "key". The key is a small region to identify the data channels that you can drag around within the XY window. For each visible channel it display the channel name and draws the line and point style for the channel.

The key is intended for use with a relatively small number of channels.

Auto expand axis range

If this is checked, each time new data points are added to the XY view that lie outside the currently displayed area, the axis ranges are changed to display all the data and the view is marked invalid so that it updates at the next opportunity. The script language equivalent of this is `XYDrawMode(-1, 5, autoExpand%);`.

Show Key

Check this box to display the XY view key, clear it to hide the key. The script equivalent of this is the `ChanKey()` command using `flags%` value of 1 to show the key. You can also control the key visibility with `XYKey()`.

Key with border

Check the box to have a rectangular border drawn around the key, clear the box to remove the border. The script equivalent of this is the `ChanKey()` command using `flags%` value of 4 to show the border. You can also control the border visibility with `XYKey()`.

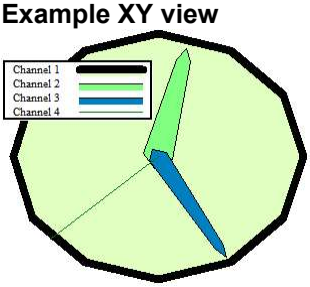
Key with transparent background

Check this box to show the contents of the XY view through the key. Clear the box for an opaque background. The script equivalent of this is the `ChanKey()` command using `flags%` value of 2 to for a transparent key. You can also control the key transparency with `XYKey()`.

Do not show example channel data

Check this box to remove the example data traces from the key. These examples are drawn to the right of the channel titles and are usually needed to identify the traces. However, some users like to set descriptive text for the entire view in the channel titles and do not want the example channel data. The script equivalent of this is `ChanKey(0, 1, 1)` to hide the example channel data.

Example XY view



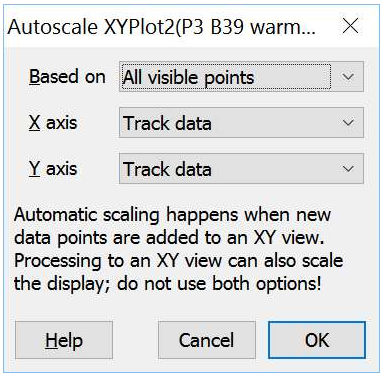
Scripts\clock.s2s

This example XY view, (generated by the `clock.s2s` script in the `Scripts` folder in the *User data folder*) shows the key. You can make the key visible by opening the XY Options dialog as described above.

In this example, several channels use filled mode. The channel fill colour displays below the line in the key. You can set the key background transparent or opaque and choose to draw a border around the key. You can experiment with the XY Draw Mode dialog to see the effect this has on the key.

If you move the mouse pointer over the key, the pointer changes to an open hand. Hold down the mouse button and the pointer changes to a closed hand to show that you can drag the key around the XY view.

XY Autoscale



This option is for XY views only and opens the XY view Autoscale dialog. This dialog determines how the x and y axes of an XY view behave when new data points are added to the XY view. You can use this to have the XY view automatically track your data, displaying either all of it, or concentrating on the most recently added points.

There are two common methods used to add data points to XY views:

1. Using the Analysis menu Measurements->XY view command.
2. Adding points using the script language.

If you are adding data using the Measurements to XY view command, the associated Process dialog has its own method for Optimising the axes, which will take precedence over any automatic scaling you set in this dialog. If you enable both methods, the Process dialog optimise will win, and the time spent calculating the Autoscale will be wasted.

What does Autoscale do?

An XY view holds multiple channels of data. Each channel is a list of (x,y) data points that can be drawn in a variety of ways. From the script language you can both add data to a channel and delete data points from a channel. The Measurements to XY view process can also add (and remove) data points.

Each time data is added to an XY view from whatever source, Spike2 gives the XY view the opportunity to rescale the x and y axes. This is done based on three screen rectangles that enclose:

3. all the visible data points (including those just added)
4. only the newly-added data points
5. The current visible drawing area

The settings you make in this dialog determine which rectangles are used and how they are used to rescale the x and y axes (which can have different scaling modes).

The script equivalent of this dialog is `XYDrawMode(-1, 5, autoScale%)`; you can record any changes you make in this dialog as a script.

Based on

This field determines which set of data points are passed to the x and y axis optimise routines. You have the following options:

No automatic scaling	This turns off automatic scaling. The x and y axes are not changed by adding data to the XY view.
New points + visible Y	The screen rectangle used for optimising holds the full range of visible data points in the y direction, but only the newly added points in the x direction.

New points + visible X	The screen rectangle used for optimising holds the full range of visible points in the x direction, but only the newly added points in the y direction.
All visible points	All the visible points in the XY view (added and existing points).
New points only	Only the points that were added this time.

The most common settings will be No automatic scaling and All visible points. To use the other settings you will usually have an axis setting that has a fixed size, or that has a minimum size.

X axis, Y axis

These two fields have identical options and determine how the x and y axes change using the x and y dimensions of the rectangle generated by the Based on field. You can choose from:

Expand only	If any point in the data passed is not visible, the axis range is increased so that it is visible.
Track data	The axis is set to display the entire range of the data selected by the Based on field. If the range has no size (for example there is only one point), the axis scale is unchanged and the axis is centred on the new point. You would normally use this with All visible points mode.
Fixed size following data	The length of the displayed axis is saved the very first time data is added and the axis is kept at this length. The axis is shifted so that the mid point of the data selected by Based on is in the middle of the axis. You can use this with the New points options to track the most recently added data.
Track with zoom in limit	The length of the displayed axis is saved the very first time data is added. This then behaves as the Track data option except that the range of the axis is not allowed to be less than the saved size.
No automatic scaling	The axis is not changed by the data.

Cancel and OK

Changes made in this dialog are applied immediately you make them so that you can see the result. If you use the Cancel button, any changes made are removed. If you use OK and have made any change, the original state is added to the Undo queue and if you are recording your actions, the changes are recorded.

Backwards compatibility

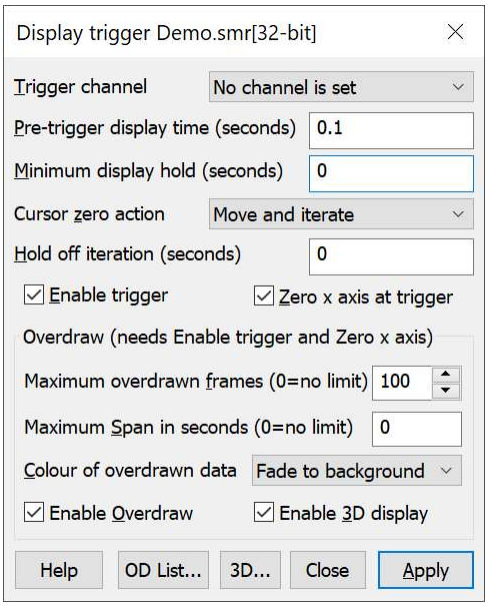
Up to Spike2 version 8.09b, the autoscale option was much simpler, being either off (equivalent to No automatic scaling) or on (equivalent to the New points + visible Y and both axes set to Expand only). If the resources of a previous data file are read, off works as you would expect and on is read as the exact equivalent. If an old version of Spike2 reads the resources created by this version, it will treat anything being set as on, otherwise as off.

Trigger/Overdraw

This View menu command is for time views only and opens a pop-up menu that leads to commands for controlling triggered displays, display overdrawing and 3D (three dimensional) displays of sweeps of data identified by trigger events:

Display trigger	This is the overall set up dialog for triggered and overdrawn displays. Overdrawn displays use a stored list of trigger times.
Overdraw List	A quick way to overdraw using an event channel as a source of multiple trigger times.
Overdraw 3D	Controls how frames are shifted and scaled to give a 3D effect.
Clear List	Empty the stored list of trigger times.

Display Trigger



The display trigger is used with Time views to provide an oscilloscope style trigger, paged display on-line, display overdraw, a 3D display and a means of easily moving back and forward to the next or previous trigger both on-line and off-line. The script language equivalent of this command is `ViewTrigger()`. This dialog holds items that define how to select the sections of data to display, what to do on each trigger event and controls overdraw.

Note that the width of each triggered section is controlled by the standard view X Axis Range controls, you do not set it in this dialog. When using this dialog, you will most likely want to enable the Width control and leave the Left and Right values alone. Alternatively, use the Enlarge/Reduce View controls.

Trigger channel

The Trigger channel field sets an event, Marker, WaveMark, TextMark or RealMark channel to be used as the trigger. You can select Paged display for a permanent trigger (for on-line paged displays rather than a scrolled display). This field can also display No channel is selected if the overdraw trigger list contains times from a mix of data channels.

Pre-trigger display time

The Pre-trigger display time field sets the time before the trigger to show each time a trigger occurs. This dialog does not set the width of the time view; that is set by the normal time view mechanisms. If you set the pre-trigger display time larger than the displayed time view width or negative, the trigger point will not be visible. Negative pre-trigger times move the trigger point off the screen to the left of the display.

Minimum display hold time

The Minimum display hold field is used on-line and sets the minimum time that data is displayed after the current time passes the right-hand edge of the screen. This allows you to see individual data frames with a high frequency trigger. A value of 0 means wait until the current screen is displayed before looking for new triggers.

Cursor zero action

The Cursor zero action field has three setting that control what happens to cursor 0 and any active cursors that depend on it when the view triggers:

No action	Cursor 0 state is unchanged.
Move to trigger	Cursor 0 moves to mark the trigger point, active cursors do not move.
Move and iterate	Cursor 0 moves to mark the trigger point, active cursors 1-9 move. The Hold off iteration field is made visible.

Hold off iteration

This field appear when Cursor zero action is set to Move and iterate. It is used online and sets a time, in seconds, to delay the active cursor iteration after the trigger. This allows you to search for data features after the trigger point when sampling or rerunning (when data after the trigger has not yet been sampled).

Zero x axis at trigger

The Zero x axis at trigger check box changes the x axis so that 0 lies at the trigger time. This is purely a visual convenience; all measurements are still in the original x axis units. You must check this box if you want to use display overdraw or 3D display modes. Script users can find the current zero position with the `ViewTrigger(4)` command.

Enable trigger

Check the Enable trigger check box and click OK/Apply to allow display triggering. Two extra  buttons appear in the area to the left of the x axis scroll bar. These buttons can be used on-line and off-line to move to the previous and next trigger. With Paged display set, they move by the time between the trigger point and the right hand screen edge. If you use these buttons on-line to move to a trigger, this disables

the automatic tracking of new triggers. You can re-enable automatic tracking by scrolling to the end of the view or pressing the `End` key.

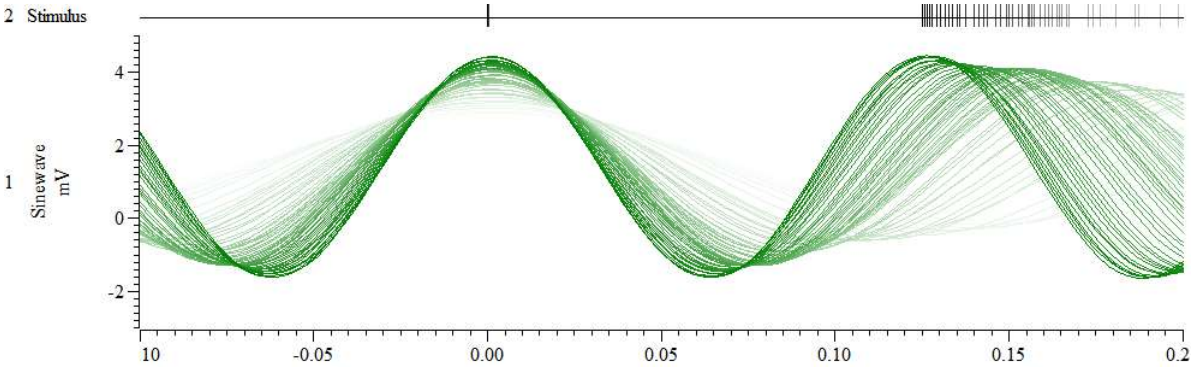
OD List... and 3D...

These buttons open the Overdraw list dialog and the Overdraw 3D dialogs.

Keyboard control

In addition to clicking the buttons, you can also step to the next and previous trigger point with the `Alt+Shift+Right` and `Alt+Shift+Left` key combinations.

Overdraw



An example of overdrawn data

The overdraw section of the dialog is enabled when both Enable trigger and the Zero x axis at trigger are checked. In overdraw mode, data sections identified by triggers are drawn over each other in time order, oldest first up to the current time. We call each overdrawn section a *frame*. You can change the displayed time range as normal, but trigger times are ignored if time before 0 or time after the end of the file would be displayed. Each new trigger time is added to a list of previous times. If you step backwards, all trigger times after the time you step to are forgotten. You can also add trigger times with the Overdraw List dialog. Accepting the dialog settings with OK clears the list if the new settings are incompatible with the old settings (for example if the channel is changed).

Overdraw is just a display mode. All measurements, cursor positions and the like behave as if the overdrawn frames are not present.

Maximum overdraw frames

You can limit the number of overdrawn frames of data. Values up to 4000 are allowed, or 0 meaning no limit on the number of frames. It takes time to draw each frame, and you will experience significant screen update delays if you display huge numbers of overdrawn frames of data over long time periods. You can break out of such long displays with the `Ctrl+Break` key combination. If you set a limit, this can be used in 3D mode to set a z axis of constant length. We suggest that you set a limit to the number of frames, particularly for on-line use, to limit the drawing time.

Maximum span in seconds

You can also limit the time range of the overdrawn frames with this field. Each time you add a trigger time, all trigger times after the new time and any trigger time that is more than the Maximum span before are discarded. You can set the value 0 for no time limit. If you set a limit, this can be used in 3D mode to set a z axis of constant length.

Colour of overdrawn data

The last frame drawn (which corresponds with the current trigger time, that is the time for which the x axis is showing 0) is always in the standard colour. You can choose how the remaining frames are drawn from:

No change	Draw in the normal colours.
Half intensity	A equal mix of the normal colour and the background colour.
Fade to background	A gradual fade from the normal to the background colour. In 3D mode, the colour depends on the z axis value, otherwise it depends on the frame number.
Fade to secondary	A gradual fade from the normal to the secondary colour. If no secondary colour is set, this is the same as Fade to background.

Enable Overdraw
Check this box to enable overdraw mode. You must also have checked Enable trigger and Zero x axis at trigger. Frames are added to the overdraw list by stepping to the next trigger event or with the Overdraw List dialog or the `ViewOverdraw()` script command.

Enable 3D display
Instead of overdrawing the frames of data exactly on top of each other, you can choose to draw them offset both vertically and horizontally to create a three-dimensional effect. Check this box to enable this drawing method. The 3D drawing only occurs if you have also checked Enable trigger, Zero x axis at trigger and Enable Overdraw. The screen arrangement of channels and frames in 3D drawing mode is controlled by the Overdraw 3D dialog.

On-line and Rerun use of triggered displays

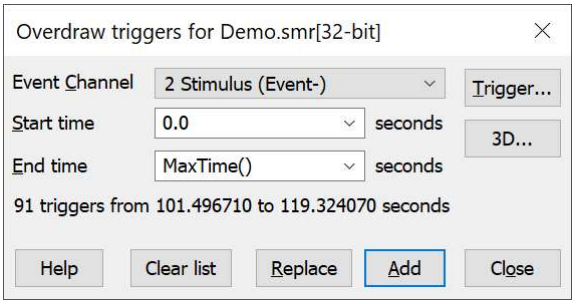
When enabled, incoming trigger channel data is searched until a trigger is found when the display will hold with the pre-trigger time shown before the trigger until another trigger occurs. The hold time will be at least as long as the Minimum display hold field from the point where the display reaches the right-hand edge of the screen. In Overdraw mode, the display hold time limits the number of frames that will be overdraw as if you trigger at time t, the earliest time the next trigger will be used is: $t + w - p + h$ where w is the screen width, p is the pre-trigger time and h is the display hold time.

If you use the buttons at the bottom of the window to move to a previous trigger, automatic display updates on new triggers are suspended to allow review of previous triggers. You can re-enable automatic trigger updates by scrolling to the end of the time view. The easier way to scroll to the end is to use the `End` key.

With Paged display selected, sampling begins as normal, then each time the right-hand screen edge is reached and the hold time has passed, a new sweep starts. The pre-trigger time sets the overlap between the sweeps. In overdraw mode, each new sweep adds a new trigger time.

Overdraw List

In a time view you use this View menu command to add a list of trigger times to a time view based on the settings in the Display Trigger dialog. If the time view is not in overdraw mode, the dialog displays a warning message and a button to open the Display Trigger dialog where you can set overdraw mode. Dialog controls are:



Event Channel

An Event, Marker or Marker-derived channel in the time view to use as a source of trigger times to add to the list held by the time view. You can also select No channel to add trigger times by typing them in.

Start time, End time

The time range to search for events to add to the list. All events found in the time range are used as trigger times. If you select No channel as the source, the End time field is hidden and the Start time field sets a trigger time.

Clear list

Delete all the items in the list.

Replace

Click this button to clear any existing times from the list before adding new times. If the Event Channel is not the same as the channel set in the Display Trigger dialog, the Display Trigger dialog channel is changed to match.

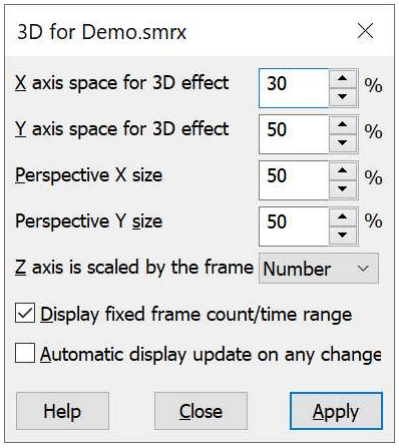
Add
Click this to add times identified by the channel and the time range to the list. Unlike manually adding times by stepping to the next or previous event when the last added time causes all later times to be deleted, added times are merged into the list in ascending time order. The current time display is set to be the last time in the list.

Trigger... and 3D...
These buttons open the Display trigger and Overdraw 3D dialogs.

Undo
Actions in this dialog are added to the Undo list of the view. To Undo changes made in this dialog, click in the view and use Edit->Undo or `Ctrl+Z`.

Display Trigger dialog interaction
If you add events to the trigger list such that the event times come from a mix of channels (or if you add times not associated with a channel using a script), the Display Trigger dialog channel will change to No channel is set. If you replace events such that the trigger list holds times from a single channel, the Display Trigger dialog will show that as the current channel.

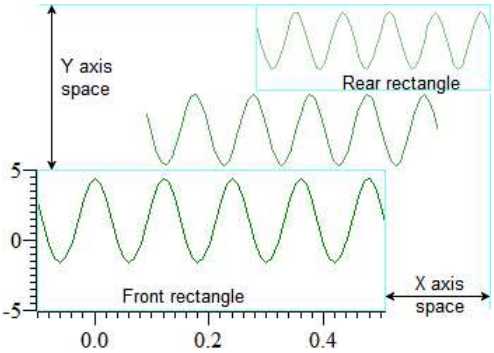
Overdraw 3D



In a time view you use this View menu command to open the 3D dialog to control the 3D drawing effect. The script language equivalent is `ViewOverdraw3D()`. This dialog does not enable 3D drawing; use the Display Trigger dialog to do that.

To create the 3D effect, each frame is drawn inside a rectangle and the position and size of the rectangle depends on a notional z axis. The z axis is based on the frame number or the trigger time of the frame. The z axis can be set to a constant frame count or time range, or it can vary depending on the current list of frames to display.

In example, below, there are three frames of data. The *Front* rectangle, used for the most recent trigger (the trigger with the maximum time), is always positioned at the bottom left of the channel area.



3D drawing rectangles

The front rectangle always holds the x and y axes (if they are displayed).

The *Rear* rectangle, used for the oldest trigger time (trigger with the minimum time), is always positioned at the top right of the channel area. The middle frame (and all other displayed frames) also has a rectangle that is calculated by a linear interpolation between the front and rear rectangles based on the z axis position of the frame.

When this dialog is open, the top and right edges of the *Front* rectangle and the left and bottom edges of the *Rear* rectangle are drawn and can be clicked on and dragged with the mouse.

The dialog fields control the positioning of the front and rear rectangles:

X axis space for 3D effect
This field sets the percentage of the width of the channel area to use for the 3D effect. The larger the value, the smaller the width of the front rectangle. Set this and the Perspective X size fields to 0 to make all frames draw vertically aligned.

Y axis space for 3D effect
This field sets the percentage of the entire view vertical space to share between all the channels to generate the 3D effect. All channels are given exactly the same space so that the 3D effect is the same for all channels. The larger the value, the smaller the height of the front rectangle.

Perspective X size
This sets the width of the rear rectangle as a percentage of the width of the front rectangle in the range 0 to 100. Setting a value less than 100 gives a perspective effect.

Perspective Y size
This sets the height of the rear rectangle as a percentage of the height of the front rectangle in the range 0 to 100. Setting a value less than 100 gives a perspective effect.

Z axis is scaled by the frame...
The position of each frame of data (between the *Front* rectangle and the *Rear* rectangle) can be set either by the frame number (giving equally spaced frames), or by the frame trigger times. Choose from Number or Time.

Display fixed frame count/time range
If you check this field and you have a set a maximum number of overdrawn frames (for z axis scaled by Number) or a maximum time span (for z axis scaled by Time) in the Display Trigger dialog, the z axis will be of a fixed size. If you do not check this box, or there is no maximum number of frames or the time range set, the length of the z axis is taken from the number of frames in the list or the time span of frames in the list.

Automatic display update on any change
If you check this box, any change made to the dialog will cause the display to update. If your display does not take long to update you probably want to check the box, but if you have a lot of data to draw you may prefer to update with the Apply button. If this box is checked, dragging the *Front* or *Rear* rectangles will repaint the view data during the drag, otherwise the repaint will wait until you release the mouse button.

Apply
This button is enabled when *Automatic display update* has been disabled and the dialog state differs from the view state.

Undo
Actions in this dialog are added to the Undo list of the view. To Undo changes made in this dialog, click in the view and use Edit->Undo or `Ctrl+Z`.

Notes
The 3D display mode behaves in exactly the same way as the Overdraw mode except that the overdrawn frames are offset and scaled. There is one other difference; if there are no frame trigger times in the list, then nothing is displayed. There is no restriction on the drawing modes you can use (other than your own common sense). Overdraw WaveMark and Sonogram modes are unlikely to be useful.

The 3D display mode makes no difference to any measurements you may make. There is still a current time range that is displayed (this corresponds to the frame of data that is at the front of the display).

Clear List

This command from the View menu Overdraw/Trigger pop-up menu removes all stored times from the list used for overdrawing.

Time view overdraw details

Each time view maintains a list of overdraw times. The list of times is always held in sorted time order with the smallest time first and the largest time last. There are no duplicated items. The size of the list can be limited to a maximum number of items and to a maximum time span between the first and the last item. If you do not limit the size or time range and the list becomes very long, it can take a long time to draw the data. You can break out of drawing with the `Ctrl+Break` keyboard command.

Adding items to the list

Items can only be added to the list when the time view is in overdraw mode. There are two modes used when adding times:

Normal — This is the mode used when you step interactively between trigger times using the keyboard commands `Alt+Shift+Left/Right` or use the trigger step buttons at the bottom of the time view. The new time is added to the list (if it is not already present), and all later times in the list are deleted. The added time becomes the current time, the x axis is set to display 0.0 at this time and the display is updated to show the pre-trigger time before the new event. If there is a time range set or the list is full, the first (the oldest) time in the list is deleted to make room for the new event.

Merge — This is the mode used when you use the Overdraw List menu command to add a list of times from an event channel. New times are merged into the list in time order. If a time range is set and the new time causes other times to be out of the time range or if the list is full, the event in the list furthest away in time from the new time is deleted to make room. The last item in the list sets the current time for drawing as described for Normal mode.

Users of the script language `ViewOverlay()` command can choose the mode to add times.

Source channel changes

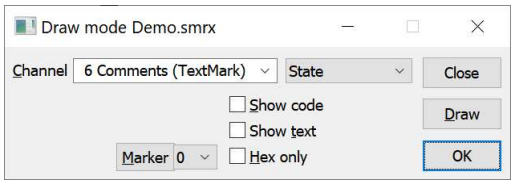
Normally, the trigger times are taken from a single channel. If this is the case, the commands to step to the next and previous event (`Alt+Shift+Left/Right`) work as normal; each step adds the time of the event to the list as described for Normal mode. However, if you add events to the overdraw list from a mix of channels, or use the script language to add a list of times (so there is no channel), these commands step backwards and forwards through the times in the list without adding or deleting any times and any previously set channel is ignored. The `ViewTrigger(-1)` command returns -1 if there is a list, but no channel.

Channel Draw Mode

This menu item is available when the current window holds a Spike2 data document or result view and sets the data channels display mode. You can set the mode for a single channel, all channels, or selected channels. The dialog changes, depending on the channel type and display mode. The Draw button is common to all modes and updates the display without closing the dialog box.

There are different controls for Time views, Result views and XY views.

Time view drawing modes



The top line of the dialog is always the same and holds controls to select the channel and the drawing mode to apply. You can select multiple channels by typing in a channel list, selecting All channels or by selecting channels in the time view and choosing Selected. If you select multiple channels, the displayed settings are from the first channel in the list.

The Close button shuts the dialog without making any change, Draw applies the current settings and leaves the dialog open. OK accepts the current settings and closes the dialog. The remaining controls vary with both the channel type and the drawing mode.

Edges

In some modes the Edges field appears if you choose a channel of event level data. You can select Rising, Falling or Both edges of the data. If you select both edges, then the display modes that show frequency count both the rising and falling edges of the event signal in their rate calculations. You would normally count only one edge, so select the edge you prefer.

For Marker, RealMark, TextMark and WaveMark channels, the Marker and Hex only fields may appear.

Marker

The Marker button/field appears if the marker code is displayed or sets the colour of the displayed items. Normally the first code (0) is used, but you can choose any of the four codes. If you select a code other than 0, the channel number field turns red as a warning of a non-standard display mode. Click the Marker button to open the Marker Filter dialog.

Hex only

The Hex only field appears if the marker code is displayed. Check the box to display all codes as two hexadecimal digits, unchecked to display codes `0x20` to `0x7e` as single characters (the ASCII character with that code). The View menu Standard Display command sets Marker code 0 and the appropriate Hex only setting for the channel type. If this is a TextMark channel, this setting also applies to the Edit TextMark dialog.

Dot size

If the display will show the data as dots, the Dot size field appears and you can set any dot size from 0 (the smallest mark possible) to 1 to 9 times the thickness of the lines set for drawing data in the Display tab of the Edit menu Preferences. If you set a size to match or exceed the Round dot minimum size field in Edit menu Preferences, the dots will be drawn as circles. Round dots take longer to draw that square dots.

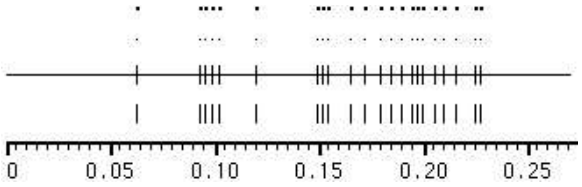
Data index

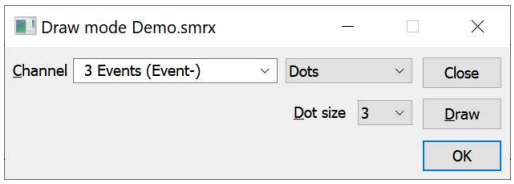
This field appears for RealMark channels to set which of the data items to display. This is a 0-based index to the item. The script language equivalent of this field is the `ChanIndex()` command. You can set separate channel titles and channel units for each data index in the Channel Information dialog. You can also set the Default channel title and units for all data indexes that do not have specific titles and/or units set.

Decoration

The Decoration button appears for RealMark data drawn as a waveform. Click the Decoration button to open the Channel Decoration dialog.

Dots and Lines mode





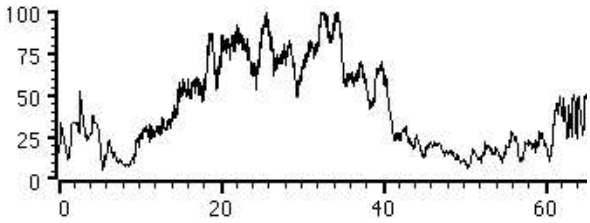
The simplest event channel draw method is dots. You can choose large or small dots (small dots can be very difficult to see). You can also select Lines in place of Dots. The picture shows the result of both types of display on an event channel. The dots are drawn in the Application colour set for Events as dots or in the channel Primary colour if it is set.

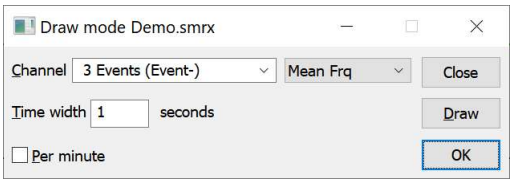
Marker channels displayed as dots also show the currently selected marker code. In this case the dots or lines are drawn in the Application colour set for Markers or overridden with the channel Primary colour. The text is drawn in the Application Marker colour or overridden with the channel Secondary colour.

If you select lines for a marker channel, the display of marker codes is suppressed. In Lines mode you have the option to draw a central horizontal line with the Centre line check box. The line colour can be overridden with the channel Secondary colour.

You can also select Dots mode for a waveform channel.

Mean frequency





The mean frequency is calculated at each event by counting the number of events in the previous period set by Time width. The result is measured in units of events per second with channel units of Hz unless the Per minute box is checked for events per minute (which changes the channel units to BPM). The mean frequency at the current event time is given by:

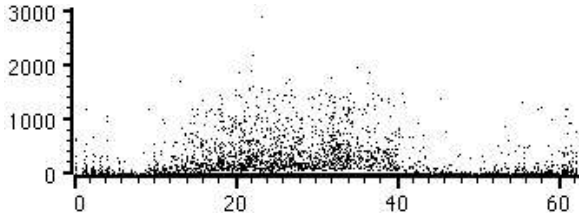
$(n\text{-}1)/(te\text{-}tl)$ if $(te\text{-}t1) > tb/2$
n/tb if $(te\text{-}t1) <= tb/2$

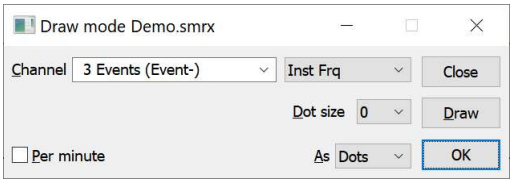
where:

tb is the bin size set, *te* is the time of the current event, *t1* time of the first event in the time range and *n* is the events in the time range

A constant input event rate produces a constant output until there are less than two events per time period. You should set a time period that would normally hold several events.

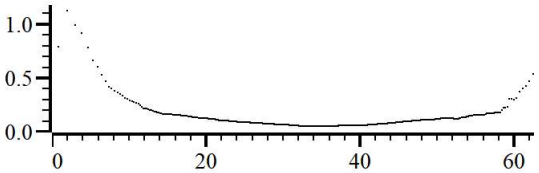
Instantaneous frequency

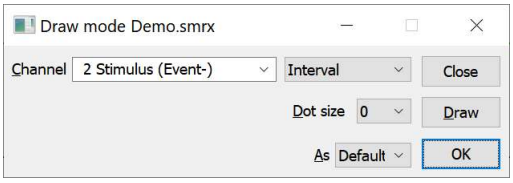




Instantaneous frequency is the inverse of the time interval between an event and the previous one on the same channel. Check the Per minute box for a rate per minute rather than per second. You can display the result as Dots, Line (linking the dots) or Skyline (horizontal lines between dots). In Dots mode you can choose the dot size. You can display the frequency as events per minute rather than per second.

Interval

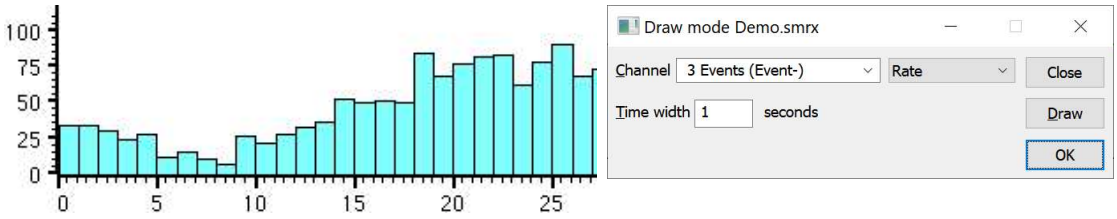




This mode draws the time interval between an event and the previous one on the same channel. The interval is measured in seconds. If you are dealing in short intervals, set the Y axis for the channel to display the units in SI units or factors of 1000. You can display the result as Dots, Line (linking the dots) or Skyline (horizontal lines between dots). In Dots mode you can choose the dot size. This mode was added at version [10.05].

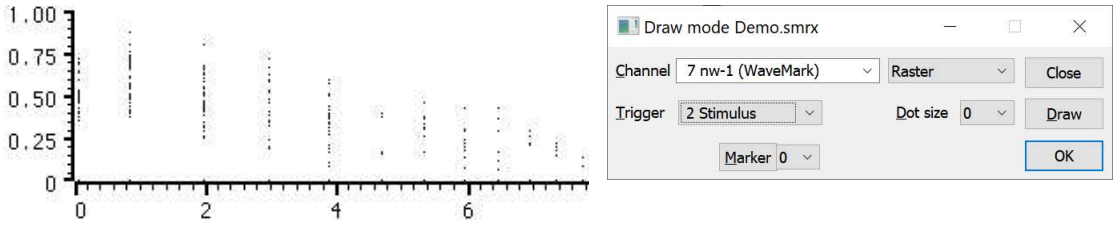
The `DrawMode()` script command mode value is 17.

Rate histograms



Rate mode counts how many events fall in each time period set by the Time width field, and displays the result as a histogram. The result is not divided by the bin width. This form of display is especially useful when the event rate before an operation is to be compared with the event rate afterwards.

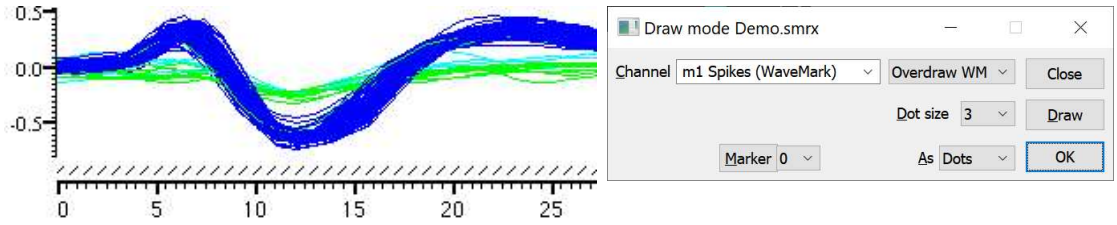
Raster display



Raster mode shows the event positions relative to trigger times. Each trigger event defines time 0 in the y direction for the sweep. The Y Range dialog sets the time range to display in the y direction; negative times show pre-trigger events. For each trigger, events are drawn no further back in time than the previous trigger and no further forward in time than the next. The trigger events are also drawn at y axis time 0.

If the channel is marker-based, events are drawn in the colours set for the selected WaveMark code otherwise they are drawn in the channel primary colour. The trigger events are drawn in the channel secondary colour.

Overdraw WM



Overdraw WaveMark mode draws WaveMark data as superimposed waveforms. Channels drawn in this mode are moved to the top of the window and separated from the x axis (which does not apply to them) by a hatched bar. If this mode is used during data capture and the screen is scrolling to show the latest data, new WaveMark events are added, but old events are not erased (to stop the display flickering). Click on the x axis scrollbar thumb to force an update.

As and Dot size

From version [10.05] you can draw WaveMark data in Dots mode. Setting any mode other than Dots draws the data as a cubic spline.

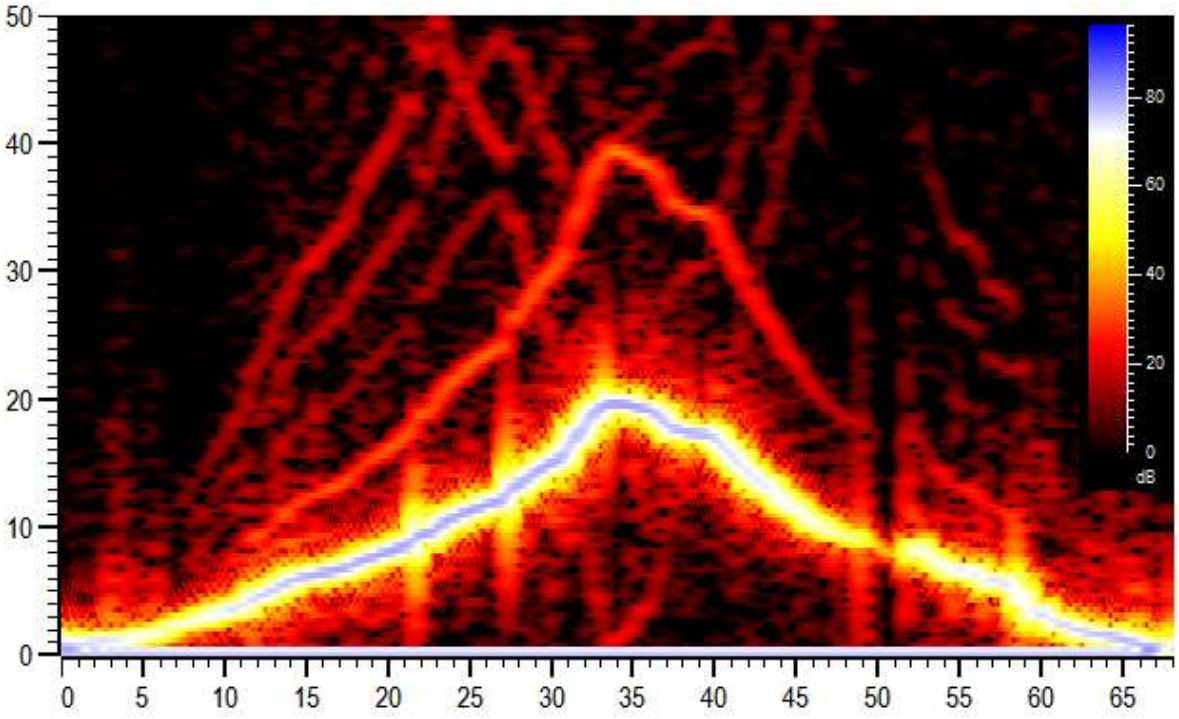
Find with cursor 0

To locate an event, right-click on its waveform at a point where it is clear of all other waveforms and select Find with cursor 0 in the pop-up menu. This locates the event nearest to the clicked position and moves cursor 0 to it. If the Edit WaveMark dialog is open, this will display the event. Any active cursors will iterate.

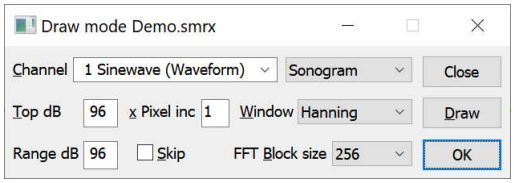
Set WaveMark codes

Hold down `Alt+Ctrl` and click and drag a line over the events you want to identify. Once you have started to drag you can release the `Alt+Ctrl` keys. If you continue to drag and hold down the `Shift` key, the line is constrained to be horizontal. On mouse up, a dialog opens in which you can set codes for the intersected events. If the channel has multiple traces the line must only extend over a single trace. This also works when the spikes are drawn in Waveform and WaveMark drawing modes.

Sonogram display



Sonogram mode shows how the frequency content of a waveform channel changes with time. The y axis units are Hz (frequency) and useful results are available for the frequency range 0 to one half of the sampling rate for the waveform channel. You can control the colour scale used for the sonogram and display a key to indicate how the colours map to intensity.



By default, intensity of the frequency content is indicated by a grey scale, the darker the image, the more intense the signal. However, you can choose a colour scale, or create your own scale in the Edit menu Preferences Display tab. You can set:

Top dB — The signal intensity that maps to the top of the sonogram colour scale. dB means decibels, which is a logarithmic measure of ratio, usually of amplitudes or power. 20 dB is a factor of 10 in amplitude. Spike2 stores waveform data as 16-bit integers, and we measure the amplitude with respect to 1 bit, so 96 dB is the maximum possible level. For RealWave channels, (stored as 32-bit floating point values) the range is much greater, but the 0 dB is still taken as the value that corresponds to 1-bit if the data were converted to 16-bit integer using the current scale factor. That is the 0 dB level is `scale/6553.6` where `scale` is the channel scale factor.

If you display the sonogram key you can choose the signal amplitude that is drawn as 0 dB in the key. This is purely a convenience for interpreting the data and has no effect on how colours map to the intensity of the signals.

Range dB The range of data to display as a colour map. Signals with an intensity of Top dB - Range dB (or less) are displayed as the colour at the bottom of the sonogram colour scale. If you are unsure what dB values to set for a new signal, setting values of 96 dB for both Top dB and Range dB will display something in almost all cases!

You can be caught out with RealWave data channels. If you are not getting sensible results, double click on the channel y axis to open the Channel information dialog, then click the Rescale button. This will set an appropriate channel scale factor for almost any channel.

x Pixel inc You can speed up drawing, at the expense of resolution, by setting this field to values greater than 1. It sets the number of screen pixels in the x direction to calculate at a time. A value of 1 gives the best visual resolution (and the slowest calculation and drawing time).

Window The sonogram is calculated using a Fast Fourier Transform (FFT). As explained in the analysis section for the Power spectrum, it is important to apply a "data window" to the signal before taking the power spectrum, otherwise the results are difficult to interpret. We provide several different types of window: None, Hamming, Hanning, Kaiser 30dB, Kaiser 40dB, Kaiser 50dB, Kaiser 60dB, Kaiser 70dB, Kaiser 80dB and Kaiser 90dB.

All windows are a compromise between increasing the apparent width of a spectral peak and the ability to see small signal peaks in the presence of large ones. If you apply no window, you will get the sharpest resolution of a single peak. However, you will not be able to see any small peaks around it due to the "side lobes" of the window. If you are not familiar with the use of windows, the Hanning window is a reasonable compromise. The Kaiser xxdB family of windows has the property that the largest side lobe is xx dB below the peak. Of course, the larger the xx dB, the wider the peaks become.

Special calculations are done where there are discontinuities (gaps) in the data and at these points we ignore the selected window to reduce the computational effort. This effect persists up to half the FFT block size around each discontinuity.

Block size This determines the number of data points used in the FFT, and thereby it determines the frequency resolution (y axis) and time resolution (x axis). Like the choice of data windows, this is also a compromise. The larger the block size, the better the frequency resolution, but the worse the time resolution. If you are looking for a short, localised burst of changing frequency, you will need to use a block size that is smaller than the duration of the episode you are looking for. Alternatively, you could consider the Continuous Wavelet Transform, available through the `ArrCWT()` script command.

Skip If you are analysing a lot of data, there can be many thousand data points for each screen pixel. If you check this box, the sonogram will only analyse the first Block size points for each pixel (normally the sonogram analyses all the points). This can save time if you have a really large file. Of course, the result will only represent a "sampling" of the correct response.

You can export Sonograms as enhanced metafiles and as bitmaps. Printing is supported to Postscript compatible printers. With other printer types, the intensity scale output may be quite coarse. You may obtain better output by saving the sonogram as a bitmap and printing from specialist bitmap editing programs.

Sonogram key

To display the sonogram key, right-click on a channel displayed in sonogram mode and select Show Key from the context menu. The key is a rectangular window that can be dragged around the time view. It holds a colour bar with an axis, mapping the colours in the sonogram display to dB. Interactive key actions are both recordable and can be undone with `Ctrl+Z`. The key can be controlled from a script with `ChanKey()`.

Key position

The key is normally positioned on top of the channel that it relates to, but you can drag it wherever you like in the data area of the time view. When dragging the key, you will notice that if you drag it out of the channel rectangle, it is slightly "sticky" at the upper and lowed edges of the channel rectangle. This is to let you align it more easily with the top or bottom of the channel.

The key position is remembered either relative to the channel rectangle, or relative to the view data area. Which it is relative to determines how the key positions itself if the view is resized or the number of displayed channels changes. Normally, if you drag the key so the top left corner lies within the channel rectangle, the position is remembered relative to the channel, otherwise it is relative to the view. If you hold down the `Ctrl` key and

drag, there is no "stickiness" when you cross a channel edge. If the Ctrl key is down when you release the mouse button, this reverses what the position is relative to.

Context menu

If you right-click on the key a context menu appears with several key-specific options.

Sonogram colours
Opens the Colour scale dialog. Currently all sonograms share the same colour scale.

Set dB offset
This opens a dialog that lets you set the signal value that is used as the 0 dB reference level for the axis.

Rotate Key
The sonogram key can be displayed with a horizontal or a vertically axis. This context menu command switches between the two possible states.

View background
The key can be displayed with the background set to the sonogram colour for the lowest intensity signal, or you can select this option to set the background to the background colour set for the data view.

Mirror axis
When the key is vertical, the axis is usually displayed on side that is closer to the edge of the window. This option displays it on the side nearer to the centre of the window. This has no effect for a horizontal key.

Hide Sonogram Key
This removes the key from the display. You can restore the key with all its settings unchanged by using the channel Show Key context menu command.

Draw Border
Check this option for a rectangular border.

Default position
Moves the key to align with the top-right of the channel rectangle.

Sonogram key dB offset

This dialog lets you set the 0 dB reference point for the axis in the sonogram key. It has no effect on how the sonogram data is displayed (this is determined by the Draw Mode dialog).

When you display the sonogram key, the axis is graduated in dB, and you will probably find that the 0 dB level is somewhat arbitrary (being relative to 1-bit of the equivalent 16-bit waveform). This is especially the case if your source data is a RealWave where the concept of 1 bit does not apply (but see below). This dialog allows you to choose a reference level in terms of the channel data. For example, if your data is calibrated in Volts, you may want the 0 dB point to be referenced to 1 Volt rather than 1 bit. To do this, just type in 1 in the edit box and hit OK.

To set the key back to reference 1 bit you can type 0 in the edit box and click OK or click the Reset button, which resets the scale and closes the dialog.

Sonogram details

This section contains more technical information about the sonogram and how it is implemented.

Db scale

Sonograms show a representation of the power in a signal using a map of colour to intensity. The dB (decibel) scale is used for this purpose. This scale describes a quantity (in our case power) with respect to a reference value. It is calculated as:

$dB = 20*\log(A/Ar)$

where log(x) means log to the base 10 of x, `A` is the amplitude of a quantity and `Ar` is the reference amplitude. An alternative representation is:

dB = 10*Log(`P`/`Pr`)

where `P` is a power and `Pr` is a reference power. As power is proportional to amplitude squared, it is easy to see that these definitions are equivalent.

The 1-bit reference

Spike2 originally supported 16-bit integer waveform data only (in the range -32768 to 32767). Scale and offset values convert the integer data into user values. The sonogram was set, by default, to display data with respect to an amplitude of 1 data bit. Unless you use the Draw Mode dialog to adjust things, the top of the scale is mapped to 96 dB (relative to 1 bit) and the range of the display is set to 96 dB. With 16-bit integer input the maximum input amplitude for an unclipped signal relative to 1 bit is 90.3 dB (`20*log(32768)`). If there is random noise present, you can visualise amplitudes smaller than 1 bit, but for most purposes, the default settings span the useful 16-bit waveform range. However, then we added RealWave data channels.

RealWave data

RealWave channels store data as 32-bit floating point. These channels have a scale and offset value that is used (when required) to convert the data to integer values. Thus, there is still the concept of 1-bit of integer data, and this is used as the reference. However, if the scale value is poorly chosen, (either far too big or far too small), the sonogram display will not be useful (all the data at the same end of the colour range). To set an appropriate scale value, open the Channel Information dialog for the RealWave channel and click Rescale. This adjusts the scaling so that the data maps reasonably into integer range.

Top db and range

In most cases, the useful dynamic range of the signal (the ratio between the largest and smallest data) is less than 96 dB and you may prefer to have a smaller range displayed in the sonogram. You will probably want to set the Top dB field so that the biggest signal in your data is represented. For a waveform channel (or a RealWave channel after a Rescale), a top level of 91 dB is probably sufficient for all cases; a smaller value may be appropriate if the data does not reach full scale.

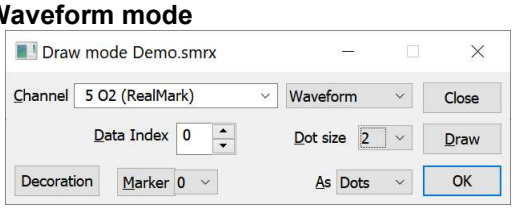
The bottom end of the displayed dB range is determined by Top db - Range dB. The smaller you set the range, the more detail you will see, but signals below the bottom of the range will all be off the bottom of the scale.

Waveform SkyLine WaveMark and Cubic spline

These modes apply to waveform, RealMark and WaveMark channels. Waveform mode joins the data points with straight lines. Skyline joints points with horizontal and vertical lines (not WaveMark channels). Cubic spline mode joins the points with smooth cubic curves based on the assumption that the first and second derivatives of the data are continuous at the data points. Cubic spline mode becomes waveform mode in Windows metafile output (use enhanced metafile format to preserve cubic splines).

WaveMark is for WaveMark channels only and is the same as waveform mode but also draws the selected marker code. The marker code is drawn in the Application Marker colour and overridden with the channel Secondary colour. Both Waveform and WaveMark modes can now use the As field to draw the spike shape waveform as dots.

Extra fields for RealMark and WaveMark channels in Waveform mode



RealMark channels have extra fields in Waveform mode. A RealMark channel can have multiple data values attached to each point; Data Index selects the value to display (indices start at 0). The As field specifies how to connect the waveform points. If Dots is selected, the Dot size field can be used; the dots are drawn in the colour set for the selected Marker code. From [11.00] you can override this by setting the channel Primary colour. The other As options are Line, Default (same as Line) and Skyline (which is the same as selecting Skyline as the draw mode).

The Decoration button opens the Channel Decoration dialog for the display of error bars and low and high ranges. WaveMark channels treat all As modes other than Dots as line mode.

Single trace

With a WaveMark channel that has multiple traces, an extra check box, Single trace, appears for Waveform, Cubic Spline and WaveMark modes. Check this to display only a single trace, which will be the one set in the Marker Filter dialog (click Marker to open it) or by the `MarkTrace()` script command. This can be useful when you want to use the `Ctrl+Alt+`mouse drag method to intersect a line with spike traces to set marker codes.

Set WaveMark codes

If the channel holds WaveMark data you can identify events that cross a line. Hold down `Alt+Ctrl` and click and drag a line over the events you want to identify. Once you have started to drag you can release the `Alt+Ctrl` keys. If you continue to drag and hold down the `Shift` key, the line is constrained to be horizontal. On mouse up, a dialog opens in which you can set codes for the intersected events. If the channel has multiple traces the line must only extend over a single trace.

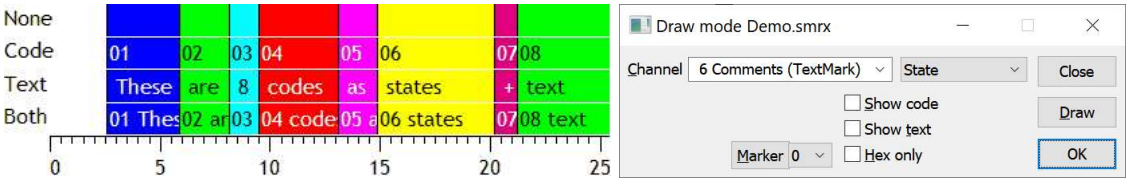
How we draw waveforms quickly

We will sometimes ignore the drawing mode you have chosen to speed up the drawing process. If the ratio of waveform data points to screen (or printer) pixels in the x (horizontal) direction exceeds 2.1, instead of drawing every single data point, we draw vertical lines that span the range of the data points. In most cases, this will make no discernible difference, apart from speeding up drawing. If you notice a sudden slowing of drawing when reducing the displayed x range, the reason is likely to be that you have just crossed from drawing vertical lines to drawing every data point.

If you are drawing in cubic spline mode, this change may make the data look smaller in the vertical direction.

We also apply this optimisation when drawing waveforms as dots when there are 20 or more dots per screen pixel.

State

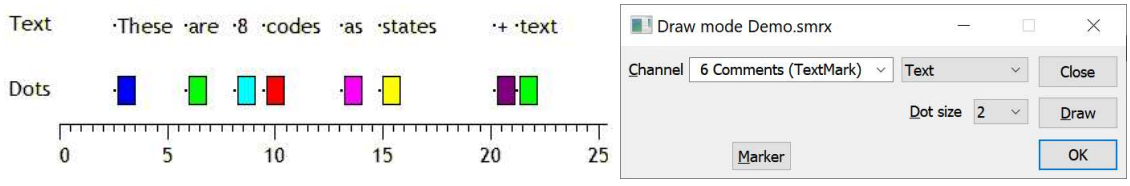


This drawing mode can be applied to a marker, TextMark, RealMark or WaveMark channel. The state is set by the selected marker code and persists up to the next marker on the channel. States are drawn in the same colours as set for the WaveMark codes except that code 00 is drawn in the channel background colour. The initial state at the start of the file and before any markers are found is assumed to be 00. You can choose to display the marker code and if this is a TextMark channel, the text.

If you set a marker filter on a channel drawn in state mode, this extends states through the filtered out markers, which may not be desirable. If a channel is in this mode, the Cursor Values dialog reports the displayed state.

From version [10.07] you can group a marker or derived type channel and use State mode to colour the background of the group. To do this put the State channel at the bottom of the list. When grouped, any code and text display is shown at the top of the channel area.

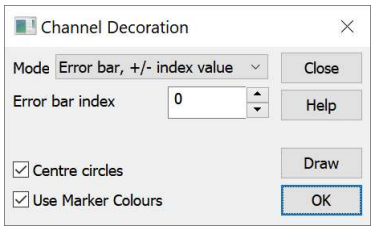
Text



This display mode applies only to TextMark channels. Each item draws as a dot at the TextMark time followed by the text. You can override the dot and text colours with the channel primary and secondary colours in the colour palette. In Text mode the Marker code is not displayed as it has no effect on the displayed text. You can still use the Marker button to open the Marker Filter dialog. From version [10.07], if you group a TextMark channel, any text displays at the top of the group channel area.

Channel Decoration dialog

The Decoration button appears in the draw mode dialog for RealMark channels that are displayed as a waveform (Waveform, Cubic Spline and Skyline modes). Click the button to open this dialog.



A RealMark channel holds events that each have one or more values associated with them. In a waveform display mode, Spike2 draws the one of the associated values as a wave (selected by the Draw Mode dialog Data index field). The Decoration dialog allows you to use other associated values as error bars or to indicate a data range. There are currently three channel decoration modes:

No decoration
The channel is displayed without any additional decoration.

Error bar, +- index value
You choose one of the associated data values to set the size of an error bar with the Error bar index field. The bar is drawn from the data value plus the index value to the data value minus the index value.

Low-high range
You choose two associated data values to be linked by a vertical bar with the High index and Low index fields. These are normally chosen as values that span the low-high range of data, so the waveform data will usually lie between the low and high values, but this is not required.

Centre circles
If you check this, a circle (filled with the channel background colour) is drawn on the channel data value.

Use Marker Colours
If you check this box, the bar and any centre circle is drawn in the Marker colour, otherwise the channel secondary colour is used.

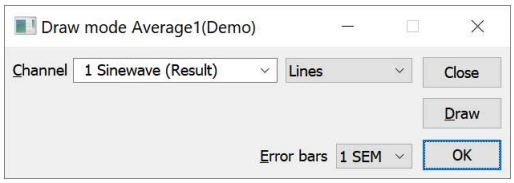
Draw
Update the display to match the current settings.

Close
Close the dialog.

Result view drawing modes

There are seven drawing modes for result views: Histogram, Line, Dots, SkyLine, Cubic Spline, Raster and Raster lines. The last two can be used if you checked the Raster data box when you created the result view. In Dots mode you can also choose to display large dots.

Error bars



If your result view has associated error information, for example a waveform average with error bars enabled, you have an extra control. You can choose from None, 1 SEM, 2 SEM or SD. It should be emphasised that error bars only have meaning if the data points that contribute to the average have a normal distribution about the mean. Given this, then 1 SEM shows ±1 standard error of the mean, 2 SEM is ±2 standard errors of the mean and SD is ± 1 standard deviation.

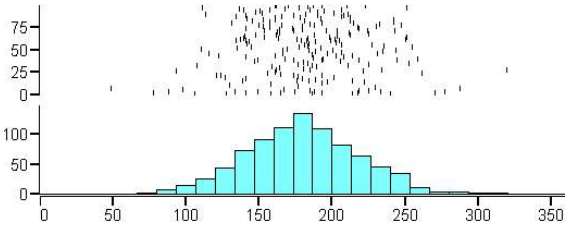
If each point of your data can be modelled as a constant "real" value to which is added normally distributed noise with zero mean, then you would expect the measured mean value to lie within 1 standard error of the mean (SEM) 68% of the time, or within 2 SEM 95% of the time. The standard deviation represents the width of the normal distribution of the underlying data at each data point.

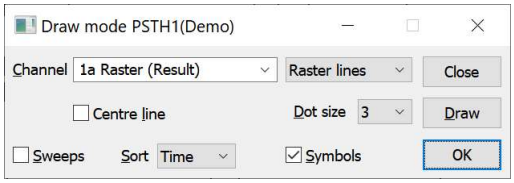
Dot Size

If the display will show the data as dots, the Dot size field appears and you can set any dot size from 0 (the smallest mark possible) to 1 to 9 times the thickness of the lines set for drawing data in the Display tab of the Edit menu Preferences. If you set a size to match or exceed the Round dot minimum size field in Edit menu Preferences, the dots will be drawn as circles. Round dots take longer to draw that square dots.

Result view raster

If a result view channel has associated raster data, there are two more drawing modes: Raster and Raster lines.





Raster mode shows each event as a dot, Raster lines mode shows each event as a short vertical line. The Dot size field controls the dot size and vertical line length. In Raster lines mode you can check the Centre line box for a horizontal line through each sweep.

Extra features with auxiliary values

If you set an auxiliary channel in the peri-stimulus, event correlation or phase histograms set up dialog or set auxiliary values from the script language, you can choose the sweep sort order. The Sort field can be set to Time or Sort 1 to Sort 4. Time presents the sweeps in the order that they were recorded. Sort n sorts the sweeps based on the sort value n. Sort value 1 is set if you set an auxiliary channel in the result view set up dialog. The script language can set all the sort values with the `RasterSort()` command.

If you check Sweeps, the y axis is a sweep count and all sweeps are evenly spaced in the y direction, otherwise it is the value of the item selected by the Sort field and the sweeps are spaced out based on the value set in the Sort field. If you sort by Time, the sweep number is the order in which sweeps of data were added into the result view.

Each raster can store 8 times and display them as symbols (circle, cross, square, up triangle, plus, diamond, down triangle, filled square) if the times lie within the x axis range. Symbol time 1 (circle) is set if you select an

event channel as the auxiliary channel in the result view set up dialog. Use `RasterSymbol()` from the script language.

XY Draw Mode

This command sets the drawing style of XY window data channels. OK makes changes and closes the dialog, Apply makes changes without closing the dialog. The Cancel button closes the window and ignores any changes made since the last Apply. The Channel field sets the channel to edit, or you can select All channels. If you change channel, the dialog remembers any alterations so there is no need to use the Apply button before changing channel unless you want an immediate update. The script language equivalent of this dialog is `XYDrawMode()`, other commands (listed with the fields) control individual features.

Join style

Style	Effect
Not Joined	No lines drawn between data points.
Joined	Each point is linked to the next by a straight line, drawn in the channel primary colour.
Looped	Like Joined except that the last point is linked to the first.
Filled	The points are not joined, but the shape made by joining is filled with the XY channel fill colour (the channel secondary colour).
Fill and Frame	Equivalent to Filled followed by Looped.
Histogram	The points are treated as a histogram. Bins are defined by each consecutive pair of points. The start and end of each bin are set by the x co-ordinates of consecutive points. The bin amplitude is set by the first point. You need one more data point that there are bins to define the final bin width. Although it is normal for data points to be set in order from left to right with equal width bins, this is not a requirement.

The line and fill colour is set by the Change Colours dialog. The script equivalent of this field is `XYJoin()`.

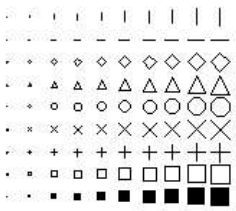
Line type and Width

These two fields set the type of line joining data points. The Line type is one of: Solid, Dotted or Dashed. The Width field determines how wide the line is in units of half the data line width set by the Edit menu Preferences dialog. Set 2 for the normal line width, 1 for half this width.

Up to Spike2 version 8.09b, if the width in pixels was greater than 1, a solid line was always drawn, regardless of the Line type field. In subsequent versions we draw dotted and dashed for wider lines. However, drawing wide dotted and dashed lines is *very* slow. On my computer, dashed lines draw 5 times slower, dotted lines draw 12 times slower.

Marker and Size

The Size field sets the marker size in units of Points (a point is 1/72nd of an inch or approximately one screen pixel). A size of 0 makes the markers invisible. You can set sizes up to 100 points. There is a wide range of marker styles to choose from. The picture shows a screen dump of all the marker styles in sizes 1 to 10. If you need to tell them apart on screen, sizes below 3 should be avoided. If you have excellent eyesight and a high-resolution printer, size 1 is viable in printed output. The Marker and Size settings have no effect in histogram mode as markers are not drawn.



Sort by

Each XY view channel holds data in the order that it was added to the channel. However, you can choose to display it sorted by the x co-ordinate or by the y co-ordinate. This is only important if the data is joined by a line, as the sort order determines how the points are joined. You have always been able to change the sort order from a script. The ability to change it interactively was added at version [10.07]. In the dialog you can choose from: Unsorted, X value, Y value. The script equivalent of this field is `XYSort()`.

XYKey

If the XY key is visible, changes made to the XY Draw Mode for visible channels are reflected in the example channel data drawn in the key. The script equivalent of this field is `ChanKey()`; the older (deprecated) `XYKey()` is preserved for compatibility with old scripts.

Multimedia files

This View menu command opens multimedia files associated with the current time view; it is not enabled unless there are files with names matching the data file. If your data file is called `fname.smrx` or `fName.smr`, the associated multimedia files are in the same folder and have names of the form: `fname-N.avi`, where the `N` stands for 1 to 4. Files do not open if they use an unrecognised format, are corrupt, use a video or audio codec that is not installed or if your computer is not equipped to replay the file.

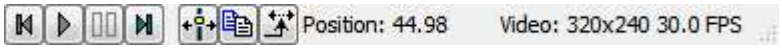
If a multimedia file contains only audio, it folds up the video display area and you can only size the window horizontally. The multimedia window displays images as close to the time of the right-hand edge of the time view as it can. If you scroll the time view, the multimedia window scrolls to match. If you use the View menu Rerun command, any multimedia windows track the rerun and replays any audio through your sound card. You can choose to link the multimedia position to the position of cursor 0 (see Cursor 0 link, below).

From version [10.01], we have made the Cursor 0 link independent of the replay command, allowing you to replay a multimedia file and drive cursor 0 in the associated time view, scrolling the time view to keep the cursor visible.

Script users have additional controls over the multimedia window. For example, they can replay individual multimedia windows independently of the associated time view. They can also grab the video image in a variety of formats (RGB, monochrome, HLS) for further processing. See the `MMOpen()` script command.

A multimedia file contains audio and/or video information. In most cases, the files will have been recorded by the s2video application. However, there is nothing to stop you renaming any `mp4` (or `avi`) file to match your data document. Click here for more information on system requirements.

Control bar



The control bar at the bottom of each multimedia window seven buttons followed by the current multimedia file position in seconds and information about the multimedia file. If you hover the mouse pointer over the buttons tool-tips appear to remind you of the button function. There is a *Status* field between the position and information fields that displays the state of the DirectShow replay process. This is blank if DirectShow is paused, R if running, S if stopped and B if busy (when stepping by frame).

Note that stepping by a frame can be slow in a compressed file as the system finds the previous Key Frame, then runs forward to the desired position. Heavily compressed files may have a Key Frame every 200 (or more) frames... The process of decoding may display intermediate images.

Replay controls

The first four buttons control replay. The arrow button replays the file from the current position to the end. This works in two modes, depending on the Cursor 0 link button. If there is no cursor 0 link, this causes the time view to rerun without opening the View menu Rerun dialog in an attempt to simulate the data sampling. If there is a cursor 0 link, this causes the video to rerun, but does not trigger the rerun option. Instead, it drives the position of Cursor 0 in the associated time view, which will scroll to keep cursor 0 in the centre of the view, if possible. The cursor 0 link behaviour is new in version [10.01]. In whichever cursor 0 link mode, the next button stops playing.

If the multimedia file contains video, the first and fourth button will attempt to step the video to the previous and next frame in the MP4 or AVI file, otherwise they are hidden. From version [10.07] stepping in AVI and MP4 files attempts to be frame accurate. Before [10.07], MP4 file stepping requested a time that was one average frame period different from the current position.

Cursor 0 link

The fifth button cancels any replay and links/unlinks the displayed multimedia position to cursor 0 in the associated time view. Linking makes Cursor 0 visible if it is hidden. When the link is made, dragging cursor 0

in the time view will also change the current position in the multimedia window. Using the step buttons or running the video will also move cursor 0 in the time view.

Displaying non-sequential video (when dragging the cursor) is not as efficient as continuous replay. In particular, displaying highly compressed files with few key frames can be slow. This is because for every move, the system must search for a key frame at or before the desired target time, then decode forwards from that point to locate the desired frame. If your video has key frames every 100 or more frames (which can happen with images that are mostly unchanging), the system may need to decode several seconds of video for each cursor move. If you use video 'scrubbing' using cursor 0 to review your images and you have control over the compression process, you may wish to limit the number of frames between key frames when you record (if your codec supports this option).

Cursor 0 can also be driven by the Spike shape dialogs, by the Measurement process and by Offline Waveform Output. From version [10.01], each time a cursor 0 driver tries to take control of cursor 0, other drivers are notified and stop driving it. In the case of multimedia replay windows, these drive cursor 0 when the Cursor 0 link is active and the Run button is clicked. If another window starts to driver cursor 0, the Run command is terminated. The Cursor 0 link will remain, and the multimedia window will attempt to track Cursor 0.

Copy
The sixth button copies the video image at the current multimedia file position to the clipboard. This button is hidden if there is no video in the file. The image is copied at the native resolution of the multimedia file, not at the current screen size.

If you hold down the `Ctrl` key and click the Copy button, this copies diagnostic text to the clipboard. This text describes the underlying DirectShow video graph and can be helpful to CED engineers when diagnosing video replay problems.

Video time offset
The right-hand button opens a dialog in which you can set the time offset between the multimedia time and Spike2 time. This button is hidden if the file does not contain video data.

Multimedia Offset

This dialog lets you view and change the time offset in the Spike2 file that corresponds with the start of the multimedia data. Ideally, the offset will be 0. However, there can be a significant fraction of a second time offset (in either direction) between Spike2 time and the multimedia time. Negative offsets move the video data earlier (in Spike2 terms), positive offsets move it later.

Alignment of video in MP4 files

Each frame of the MP4 file holds a time stamp. If the file was created by `s2video`, this time stamp should track the sampling time of the associated data file. Missing frames and varying frame rates should have no effect on the timing. There will usually be a small (but fixed for any file) time offset between the frame times and the Spike2 time. This is the time it takes for the frame to be formatted in the camera and transferred to the computer where it receives the a time stamp. On my computer, this is of order 0.1 seconds. However, when sampling a sequence of files, there can be much larger delays while Spike2 and S2Video shut down the previous file and open the next.

From S2Video 2.03, we adjust the frame timing to allow for the time delay between Spike2 starting to sample data and the video stream generating frames. The first frame in a MP4 file is always at time 0. The second and subsequent frames should be positioned in the 'correct' place. We would hope that you can leave the offset value set at 0 for such files.

Alignment of video in AVI files

An AVI file holds a list of video frames (some of which may have no data if the frame was dropped), and the only timing information is the frame rate, which is specified as the ratio of two 32-bit numbers. The only controls we have to line this up with the Spike2 time is by adjusting the frame rate and by adjusting the offset of the first frame. There is an option in s2video to calculate the frame rate based on number of frames (and skipped frames) in the AVI file divided by the elapsed time (as seen by Spike2).

As long as the video recording process was not stymied by frames being lost due to insufficient bandwidth in the system, and the frame rate stays constant, it should be possible to align the video with the Spike2 data. Sadly, it is difficult to guarantee either of these conditions and this can lead to AVI files having poor timing.

We attempt to store the time offset between the start of the multimedia data and Spike2 time in the source `.avi` file so that you only need set this value once per file. The s2video application can be configured to set an offset for AVI files if your hardware generally has a constant time offset.

Display frame times as an event channel

The following script will display the frame times in the first MP4 or AVI file associated with the current time window:

```
if ViewKind() then Message("Must have time view current to run this"); halt endif;
PrintLog("Data file: %s, Maximum time: %.2f\n", FileName$(), Maxtime());
var mm% := MMOpen(1, 1);        'Open associated video
if (mm% <= 0) then Message("No multi=media file located"); halt endif;
var hPix%, vPix%, fps;
if MMVideo(hPix%, vPix%, fps) < 0 then Message("No video in the multi-media file"); halt endif;

var n% := MMFrame(0, View(-1).Maxtime());
if (n% > 0) then
    PrintLog("%s\nVideo is %d x %d at %.8f FPS. %d frames, duration %.2f seconds\n",
             FileName$(), hPix%, vPix%, fps, n%, (n%-1) / fps);
    var frames[n%];
    MMFrame(0, View(-1).Maxtime(), frames); ' get frame times
    View(-1);
    var mc% := MemChan(2);  'Make event channel
    var i%;
    for i% := 0 to n%-1 do
        MemSetItem(mc%, 0, frames[i%]);
    next;
    ChanShow(mc%);
    var nMax% := Min(n%, 20);
    PrintLog("%12.8f\n", frames[0]);

    for i% := 1 to nMax%-1 do
        PrintLog("%12.8f, %.8f\n", frames[i%], frames[i%] - frames[i%-1]);
    next;
    Message("Found %d frames at %.8f FPS, first few times listed in Log view", n%, fps);
else
    Message("No video frames");
endif
```

Spike Monitor

The View menu Spike Monitor command opens a new window that displays all the WaveMark channels in the current time view in a grid. The command is disabled if there are no WaveMark channels in the current time view. See the Spike Sorting chapter for more details of the Spike2 Monitor.

Font

You can select the font that is used for each window in Spike2. In time, result or XY views the font size changes the space allocated to data channels. Smaller fonts give more space to the channels, however fonts need to be large enough to read! The data view dialog is a standard operating system font dialog. In a text-based view, this command opens the editor settings dialog where you can set fonts for all the text styles supported by the view type. This is described in the Edit menu Preferences under the General tab.

There are other places in Spike2 where you can set fonts for specific items, for instance the Vertical markers dialog and the Printed Header and Footer dialog. These open standard operating system font dialogs.

ClearType fonts

If you are working with a flat panel display, you should check that you have enabled smoothing the edges of fonts and have ClearType enabled as this can greatly enhance the appearance of text, especially small font sizes.

In Windows XP, font smoothing is enabled in My Computer->Properties->Advanced->Settings->Visual Effects and make sure that Smooth edges of screen fonts is checked. In Windows XP, ClearType is enabled by right clicking on the desktop and choosing Properties->Appearance->Effects... then check the box Use the following method to smooth edges of screen fonts and select ClearType.

Dialog font

The size of the font used for all Spike2 dialogs is fixed (but see the `DlgFont()` script command). At the current time, the only way to change the size is to change the Windows system display settings to alter the DPI value (Dots Per Inch). This will affect all programs, not just Spike2.

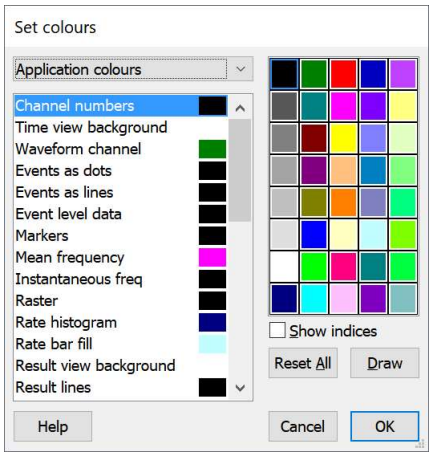
Use Colour and Use Black And White

This option is here to make it easy to print without using a lot of ink. The Use Black and White command displays all backgrounds in white and everything else in black. This does not make any changes to the application colour maps. This option saves you from the tedious task of changing every colour when you print to a monochrome printer. The Use Colour command (which replaces the Use Black and White command in monochrome mode) switches back to colour displays.

Change Colours

The Change Colours dialog controls time, result, XY and Grid view display colours (the Font dialog to controls colours in text-based views). To open the Change Colour dialog, use the View menu Change Colours command or click on the palette icon in the main toolbar. You can choose the colours that are used for almost everything in Spike2 (sonogram colours are changed by a different dialog).

If you open the dialog with an active time, result or XY view, the dialog has multiple pages. Select a page with the drop-list at the top left. Pages are: Application colours, Marker colours, View colours, Channel primary colour, Channel secondary colour, Channel background colour. The Application and Marker colour pages are always available, the remainder are available when a time, result or XY view is active.



Application colours dialog

A restricted version of the dialog can be opened from information window context menus and from the Grid view context menu.

To accommodate long lists, the dialog is resizeable. Click on the bottom edge and drag down to increase the dialog length.

Application colours

To change colours, select one or more items in the list on the left, and then click a colour in the palette on the right. You can check the result of your action with the Draw button. Cancel removes the window and undoes any changes. OK accepts (and records, if enabled) changes and closes the dialog. You can set the following (bracketed text means it can be overridden):

Channel numbers	The channel number, used to identify and select data channels and drag them to new positions.
Time view background	Background colour of the entire time view (view background). This can be overridden by the View colour, and that in turn can be overridden by the channel background colour.
Waveform channel	Waveform data in time views (primary). Also used for waveforms in the spike shape and digital filter dialogs.
Events as dots	Event data drawn as dots (primary).
Events as lines	Event data drawn as vertical lines (primary) and horizontal line (secondary).
Event level data	Level event data (primary).
Markers	Marker codes and associated dot, plus RealMark waveform (primary).

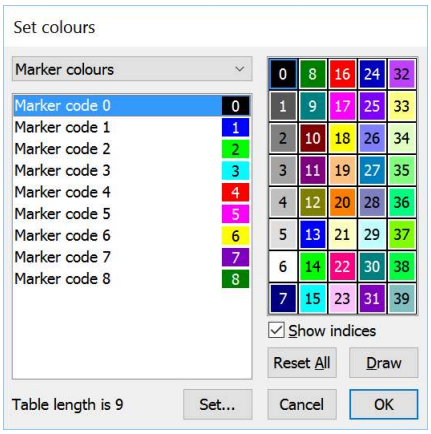
Mean frequency	Events draw as mean frequency (primary).
Instantaneous freq	Events drawn as instantaneous frequency (primary).
Raster	Time view and Result view raster responses (primary), Time view trigger event and Result raster centre line (secondary).
Rate histogram	Border colour for events drawn as a rate histogram (primary).
Rate bar fill	Fill colour for events drawn as a rate histogram (secondary).
Result background	Result view background (view background).
Result lines	Result view data as lines (primary).
Result dots	Result view data as dots (primary).
Result skyline	Result view data as a skyline(primary).
Result histogram	Border of result view data as a histogram (primary).
Result bar fill	Fill of result view data as histogram (secondary).
TextMark text	TextMark data drawn in Text mode
Cursors	The colour to use for horizontal and vertical cursors and cursor labels.
Controls	This is currently unused.
Grid	All axis grids.
Axes	All axis lines, ticks and labels.
XY view background	The XY view background colour (view background) and XY key background colour.
Not saving to disk	Used to draw data displayed in a sampling window that is not being written to disk to warn you that the data will not be saved. Also used for non-triggered data in the spike shape dialog. Set this colour the same as Time view background to use the normal colours. The not saving colour works for just about everything except spike shapes, which are drawn in the sorted code. State drawing mode shows the original colour and the not saving colour on top.
Errors (SD/SEM)	All result view error bars and envelopes (secondary).
Curve fit	Fitted curves.
Cluster background	The background colour for the clustering dialog.
WaveMark background	The background colour for all the spike shape dialogs.
Vertical markers	Vertical marker colours if marker colours are not used.
XYChannel data	XY view data channels (primary).
XY Channel fill	XY view fill colour (secondary).
XY key	XY view key
Info window text	The text colour to use for the view information windows
Info window background	The background colour for the view information windows
Pulse edit background	The background colour for the graphical sequence editor.
Pulse edit foreground	The foreground colour for thegraphical sequence editor.

If you set a channel or text colour that is too similar to the background, Spike2 will change the colour to keep the item visible. There is an option to defeat this in the Edit menu Display Preferences. Script users can set application colours with the `ColourSet()` command.

Marker colours

Several drawing modes draw Markers and marker-derived data (TextMark, RealMark, WaveMark channels) in different colours, depending on a marker code. Marker codes can take values from 0 to 255. Prior to version 7.07, there were 9 marker colours, numbered 0 to 8. Marker code 00 always uses marker colour 0. All other marker codes used colour `(c mod 8) + 1`, where `c` is the marker code and `mod` is the modulus (remainder) operator. Marker codes 1 to 8 used marker colours 1 to 8, marker codes 9 through 16 also used marker colours 1 to 8, and so on.

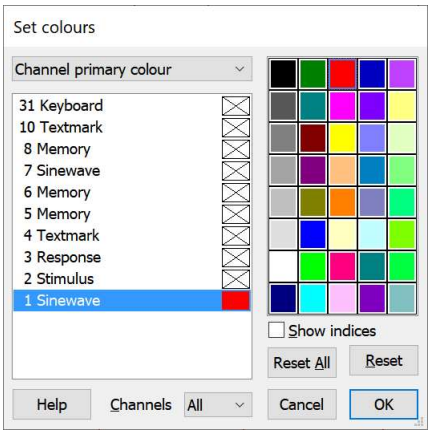
Click Set... to change the number of marker colours in the range 3 to 255. With `m` colours, marker code `c` is assigned marker colour `(c mod (m-1)) + 1`. If you increase the marker colours, new colours start set to black. Script users can set marker colours with `ColourSet()`. For WaveMark data, the colour is for the first trace, the rest fade to the background.



Marker colours dialog

Show indices
Check this box to display the palette number for each colour in the palette and the item number (where appropriate) in the list to the left. This is a convenience for script programmers.

Channel primary, secondary, background colour
These three pages assign colours to data channels in the current time, result or XY view and override any Application colour or View colour.



Channel primary colour dialog

The primary colour is used as the drawing colour for lines, waveforms, events, and histogram outlines and as the channel number colour when the channel is grouped.

The secondary colour is for filling histograms and XY channels, the raster display trigger point, event codes for Markers, horizontal centre lines for events and result view raster line mode, the text colour in Text drawing mode and for drawing the SEM and SD in result views. The channel background colour overrides the view background for the area occupied by the channel data. An X in a box marks a channel with no colour override.

Changes made on this page are applied immediately, so there is no Draw button. You can Reset the selected channels back to the colours set in the Application colours page. Channel colour overrides are stored in the resource file associated with the data file and in the sampling configuration file for new data files. Script users can set channel colours with the `ChanColourSet()` command.

The Channels field (new at [10.03]) allows you to change the list of displayed channels; choose one of: All, Visible, Selected. This can be useful in cases where you have a large number of channels, or you have hidden channels you are not interested in. Using Selected in an XY view results in an empty channel list.

View colours
This page allows you to override the application colours set for the current Time, result or XY view. At the moment you can override the view background colour and the associated Info window colours. The equivalent script command for this is `ViewColourSet()`. View colour overrides are stored in the resource file associated with the data file and in the sampling configuration file for new data files.

Grid view (new at [10.05])
This page allows you to override the current Grid view background and text (foreground) colours. If you do not override these, new Grid views use the Application colours: Grid background and Text Label. Currently, you cannot change the colour of the border around each cell, which is set to a light grey.

Grid view cell colours
You can modify the colours of selected cells by right-clicking in the grid area and selecting either Colours... to open the Grid Cell colour dialog or Reset Colours from the context menu. Both commands apply to the selected cells. The script language allows you to set and get colours of columns, cells, rectangular and selected areas of the grid with the `GrdColourSet()` and `GrdColourGet()` commands.

Assign a colour
There are two ways to assign a colour to the selected item in the list.

1. Click on the palette to the right to assign a palette colour to the list item.
2. Double-click the list item to open a colour picker to choose a colour.

The Reset button restores the selected item to its default state (if there is one) or cancels a colour override.

Changing the Palette
To change a palette colour, double click it to open the colour picker and select a replacement colour. The first seven palette colours form a grey scale from black to white and cannot be changed. You can replace the palette colour with any standard colour, or you can click the Define Custom Colours... button to select an arbitrary colour. Click OK or Cancel to exit.

The colour palette is owned by the Spike2 application, not by the data file and is stored in the registry. If the compatibility option to use old colour mechanisms is set, we also write the colour palette to the Spike2 configuration file `default.s2cx`. When Spike2 starts, the palette and Application and Marker colours are read from the registry; if there are no colours to read or the compatibility options is set, the colours in the configuration file are used (if any were saved).

Reset All (and Dark mode)

When the dialog opens from the View menu Change Colours... command or from the system toolbar, Reset All sets the Application colours, the palette and the Marker colours to a standard state. If you hold down the `Ctrl` key and click Reset All, a set of Application colours with black backgrounds is selected (Dark mode) and Marker code 00 is drawn in white. Without the `Ctrl` key, a set of Application colours with a white background is selected and Marker code 00 is drawn in black.

When the dialog opens from the Grid view or Info view context menu, Reset All returns the colours to the default values. The Application and Marker colours are not changed and you cannot set Dark mode.

Recording your actions

When recording is enabled in the Script menu, any changes you have made with the dialog are recorded as the equivalent script at the point when the dialog closes.

Undo

Each view (Time, Result, XY and Grid) has its own list of undoable actions. Changes made to View colours can be undone. Changes made to Application colours cannot be undone.

Script language

The script language function `ColourSet()` controls the palette, Application and Marker colours. The `ChanColourSet()` function controls the Channel primary, secondary and background colours and The `ViewColourSet()` command controls the view colours. From version [10.05], additional control over the colouring of individual grid columns and cells is available using the `GrdColourSet()` and `GrdColourGet()` script commands.

RGB colours

All items in Spike2 that can be assigned a colour set the colour in terms of the RGB value (Red, Green, Blue). The current graphics system stores each component as a number in the range 0 to 255 with 0 meaning no colour and 255 meaning the maximum. However, the underlying representation may change with more advanced graphics systems, so in the script language we use three values in the range 0.0 to 1.00, shown in commands as `r, g, b` when setting the values and `&r, &g, &b` when reading them back.

For example, a RGB value of (1,0,0) is red, (0,0,0) is black, (1, 1, 1) is white and (0.5, 0.5, 0.5) displays as a mid grey.

The Palette

Although all colours are set by RGB values, it is useful to have a set of colours that are easy to apply interactively, and these are stored in the Palette. This is displayed in the Colour dialog as a grid of colours. Each colour in the Palette has an index, which is used by the old (and now deprecated) colour commands that set colours by palette numbers.

In the Colour dialog, a click in the palette sets currently selected item to the RGB value of the clicked palette colour. You can double-click palette colours (apart from the fixed grey-scale colours on the left) to modify the palette colours.

Changes at version 7.07

Prior to version [7.07], the application, view and channel colours were all controlled by the colour palette. All colour items had an index into the palette and were displayed with the colour at that index. This made sense when Spike2 was originally written as most systems had 16 colours, some had 4 and some had only 2. It was

possible to generate intermediate colours by mixing dots of different hues, but this led to unpleasant screen effects. Apart from limiting the colour choice to the colours in the palette, using palette indices had the additional effect that a change to a palette colour might change items other than those that were intended. From version 7.07, all items have a free choice of colour, defined by RGB (Red, Green and Blue) values in the range 0 to 1. For example, a RGB value of (1,0,0) is red, (0,0,0) is black, (1, 1, 1) is white and 0.5, 0.5, 0.5 displays as a mid grey. We have preserved the palette as it is useful to have a set of colours that are easy to apply. A click in the palette sets the RGB value of the palette colour; the palette index is of no consequence.

Effects of the change

This change is transparent to most users. The only difference is that if you double click in the palette to change the colour, the change applies to the currently selected item in the list on the left of the dialog. Previously, if you changed palette item `n`, this would change the colour of any Application, View or Channel colour that happened to be pointing at palette index `n`.

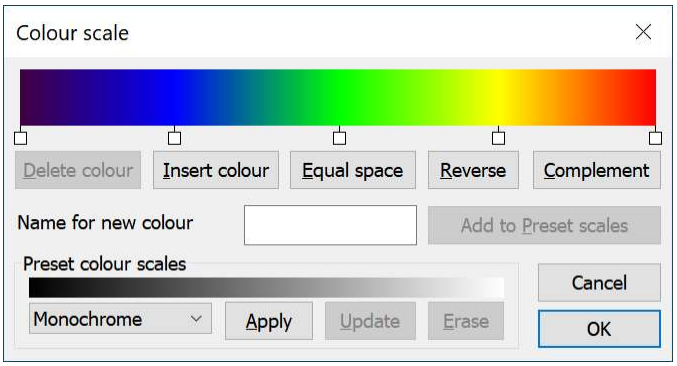
The changes should make it much easier to set colours using a script. However, we have preserved the old script commands so that old scripts will continue to work. We very much encourage you to avoid the old script commands in new scripts (unless they must also run in older versions of Spike2). The new script commands that allow you to set and read back colours using RBG values are: `ColourGet()`, `ColourSet()`, `ChanColourGet()`, `ChanColourSet()`, `ViewColourGet()` and `ViewColourSet()`.

You can still use the following commands, but we suggest that your avoid them in new code: `Colour()`, `ChanColour()`, `ViewColour()`, `XYColour()`, `PaletteGet()` and `PaletteSet()`.

We have also separated out the Marker colours (previously called WaveMark colours). This also allows us to provide more marker colours, if needed.

The new system is more open-ended than the old, and allows us to add further colours and colour overrides in the future.

Colour scale dialog (Sonogram, Density map colours)



The Colour Scale dialog can be opened by the Sonogram Colours command in the View menu, from the clustering dialog View menu Density Colour Map command and from the Edit menu, Preferences command Display tab. You can also open this dialog by right-clicking on a time view sonogram display and selecting Sonogram Colours from the context menu.

The dialog sets the colour scale for the time view sonogram display mode and for the clustering density plots. The script language commands `ColourGet()`, `ColourSet()`, `ArrMapImage()` and `Spline2D()` access the colour scales.

You can choose from a range of built-in and user-defined colour scales and create user-defined colour scales. The dialog is in three section: an upper part where you edit the current colour scale, the centre line, where you name and save user-defined colour scales, and the Preset colour scales section where you select and manage named colour scales.

The colour scale

The colour scale is the band of changing colour across the upper part of the dialog. The small squares below the colour scale mark fixed colour points. Single click a small square to select it. Double-click a small square to change its colour. The colours between the fixed colour points are formed by linear interpolation. Click and drag the squares sideways to change their position. You cannot move the first and last fixed points. You can add a new fixed point by double-clicking on the colour scale away from any small square. The buttons immediately below the colour scale are:

Delete colour — This deletes the selected fixed point; it is disabled if there is no selection.

Insert colour — If there is a selected fixed point, this adds a new point next to the selected point on the side with the larger gap. If there is no selected point, this adds a new fixed point in the largest gap

between existing fixed points. You can also add a fixed point by double-clicking the colour map clear of any fixed point. You can have a maximum of 10 fixed points.

Equal space	This spaces the fixed points evenly across the colour bar.
Reverse	This reverses the order of colours. Reversing twice restores the original.
Complement	This replaces each colour by its complement, formed by subtracting each colour from white. Complementing twice restores the original.

Name for new colour map

You can save up to 10 user-defined maps in the registry. Set a name and click the Add to Preset maps button to add a new colour map to the list. If the list is full you must use the Erase button to remove a colour map before adding a new one. If you type a name that cannot be used, the Add to Preset maps button is hidden and a text message explaining the problem replaces it. Names of user-defined maps must be unique and may not include the vertical bar character.

Preset colour maps

The drop-down list in this section holds the names of colour scales that you can copy into the current colour scale and displays the current list selection. If you have set colour scales for sonograms or clustering, these items start the list. The list continues with built-in maps, named to suggest their contents, for example Rainbow, Thermal and Geography. The list ends with the names of colour scales that you have created and saved. The buttons to the right of the list are:

Apply	Copy the current preset selection to the colour scale at the top of the dialog.
Update	Update the preset user-defined colour scale with the colour scale at the top of the dialog. You can also update the Sonogram and Clustering items, but only when this dialog is displayed from the Edit menu Preferences dialog.
Erase	Delete a user-defined preset colour scale. You cannot delete the Sonogram, Clustering or built-in maps.

OK and Cancel

When opened from the Edit menu Preferences dialog Display tab, OK closes the dialog.

When used from the Sonogram display or the Clustering dialog, the OK button accepts the current scale, closes the dialog, saves the current colour scale for future Sonogram or Clustering use and applies it to the sonogram or clustering display.

Cancel closes the dialog. It does not undo changes made to user-defined colour scales.

Script access to colour maps

From Spike2 9.01 onwards, you can access the colour maps with the `ColourGet()` and `ColourSet()` script commands. The `ArrMapImage()` and `Spline2D()` script commands make use of the colour scales to generate images. You can use `ChanKey()` to add a colour scale index key to any view. You can record changes to user-defined colour scales; recording is not smart enough to detect multiple changes to the same item, so it can result in duplicated code.

In addition to recording changes, you can copy the information to the clipboard by using `Ctrl`+mouse click. A click in the colour scale copies suitable code to the clipboard to define a colour scale for use with the `ColourSet()` script command, for example:

```
const colScale[4][5] :=
{
    {0.251, 0.000, 0.251, 0.000},
    {0.000, 0.000, 1.000, 0.243},
    {0.000, 1.000, 0.000, 0.502},
    {1.000, 1.000, 0.000, 0.753},
    {1.000, 0.000, 0.000, 1.000}
};
```

A `Ctrl`+click over one of the tabs copies information about the tab, being the position and colour, for example:

```
0.243, RGB(0.000, 0.000, 1.000)
```

Storage of colour scale information

The Sonogram, Clustering and user-defined colour scales are stored in the registry in the folder:

HKEY_CURRENT_USE\Software\CED\Spike2\Settings\ColourMap

with the names ColourMap0 to ColourMap19. The sonogram scale uses the name ColourMap0. The clustering scale uses the name ColourMap1. The user-defined colour scales are saved as ColourMap10 to ColourMap19. The names ColourMap2-9 are reserved.

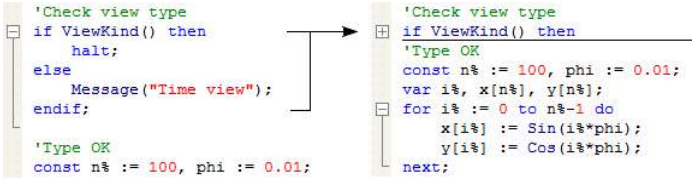
The built-in colour scales are (currently) hard-coded into the Spike2 program and cannot be changed by users. As we are finding more uses for these maps we may modify or extend the built-in scales.

Folding

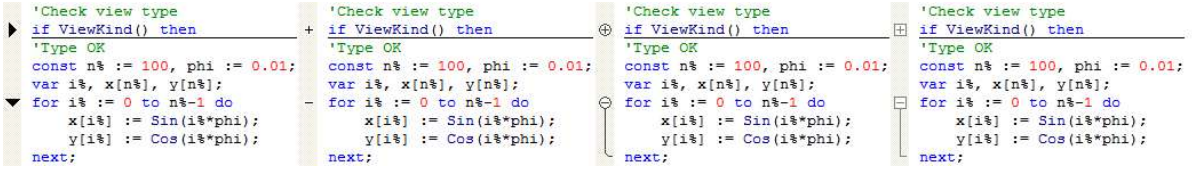
Folding is supported in the Script language editor and in the Output sequencer editor. In a script, folding is based on script language keywords for flow of control and functions and procedures; folds can be nested within other folds. In the sequencer it is based on the keyboard codes used to jump interactively to blocks of code and are not be nested.

Folding allows you to hide sections of your text that are not relevant to the current edit. It is particularly useful in large scripts and sequences.

This menu item is available for Output sequencer and Script views to control the code folding options. The folding options are covered in the Edit menu Preferences command in the General tab for these view types. This View menu command gives you a quick way to change the folding style and to toggle all the folds. The folding styles (illustrated for a script file) are Arrows, Plus and Minus, Circles and Lines, Squares and Lines:



Example of folding in a script



The four folding styles

The remaining folding commands are:

Toggle all folds

This searches the text from the start to find the first fold, then sets all the folds in the file to the opposite state. The short-cut is Ctrl+Shift+T. To toggle an individual fold, click on the fold mark.

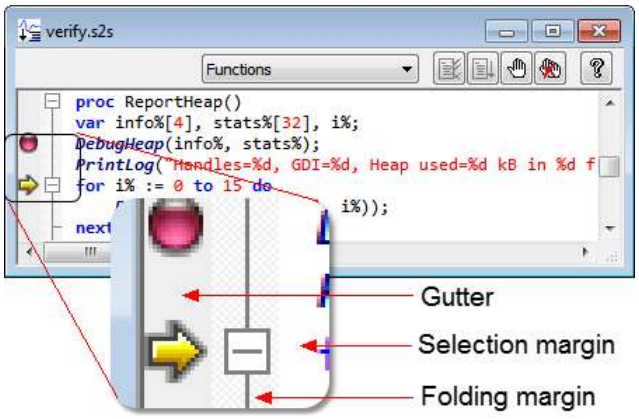
No folding

This hides the fold margin. If this option is selected and text is folded, it is unfolded.

Show Gutter

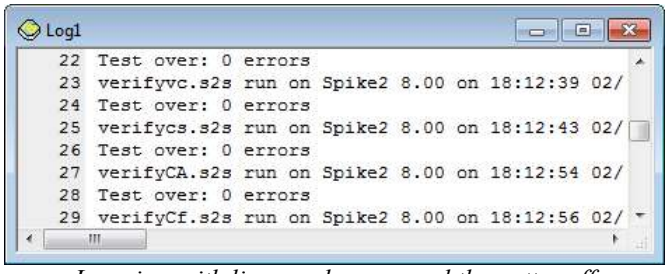
The gutter is an area to the left of the text in a text view, usually with a grey background, that can be used to select lines of text, except in a script view where clicking on it sets break points. The gutter is also used in script views to display the current line pointer. This menu command shows and hides the gutter, equivalent to the `Gutter()` script command. The gutter shares the same display background colour as line numbers. The gutter is normally visible.

If you turn the gutter off, there is a selection margin to the left of the text that can be used to select lines of text by clicking and dragging.



Show Line Numbers

You can choose to display line numbers in any text view. Line numbers usually have space for up to 5 digits, but for more than 99,999 lines of text you can use the `ViewLineNumbers()` script command to increase the displayed digits. This menu command makes the line number area visible if it is invisible and hides it if it is visible. You can change the appearance of the line number region with the View menu Font command. Select the Line number style, and a line number will appear in the example window so you can preview any changes.

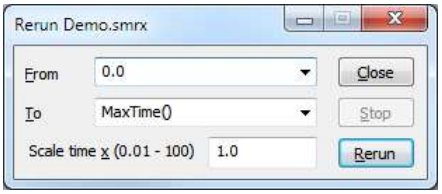


Log view with line numbers on and the gutter off

With line numbers on, you will usually want the gutter off, except in a script where the gutter is used for break points and to mark the current line. You can use the line number margin to select complete lines of text.

ReRun

This option can be used in a Time window to simulate data sampling for demonstration and training purposes. From and To define the time range to use. Scale time sets the number of seconds of data to rerun every second. Rerun starts the simulation and Stop cancels it. Unlike real sampling, there is no sample control window and the output sequencer and on-line waveform replay are not available.



When the update point reaches the right-hand window edge, the window scrolls to keep the update point at the right-hand edge. You can use the scroll bar to move through the document, but only up to the current end. If you move the scroll bar fully to the right, the window will scroll automatically, if the scroll bar is not at the right, the window will not scroll, but the scroll bar position will change as time passes to show its relative position in the document. The document will keep running until it reaches the end or you use the Stop button.

Any open multimedia windows associated with the time window also rerun and replay the audio signal to stay aligned with the current replay point.

In place of scrolled displays, you can have a paged update. Open the View menu Display Trigger dialog, select None as the trigger channel, check Enable trigger and click OK.

Clicking Rerun while re-running

If you have the default setting to run from 0 to MaxTime() and you click the Rerun button while re-running, the value of MaxTime() will be wherever the re-run operation has got to, not the end of the file. If you want to

interrupt a re-run operation and replay the entire file, click Stop first (so that MaxTime() returns the full file length), then Rerun.

Link to Offline waveform output

To rerun to match a waveform playing through the 1401 DACs or through a sound card in your computer, check the Rerun the file to Match the waveform output box in the Offline waveform output dialog.

Annotate

This menu item is visible if the current view is a script and is enabled if the script is compiled and has been saved. Select the option to display the compiled script code below each script line. Select it again to hide the compiled code. We use this annotated view to diagnose script compiler problems. Most users never need to use this option. The following is for interest only; it gives a flavour of the inner workings of the script system. We do not document the innards of the script system as there is no plausible use for it.

Example

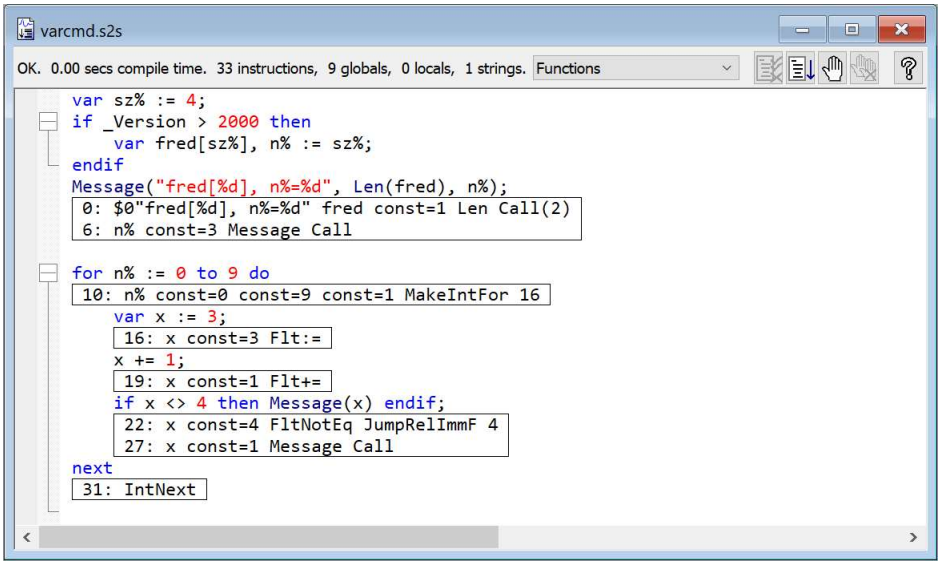
The following script was written to illustrate the quirks of the `var` statement. If you run it, the output is a message box containing: `fred[0], n%=0`:

```
var sz% := 4;
if _Version > 2000 then 'Equivalent to if 0 then
    var fred[sz%], n% := sz%;
endif
Message("fred[%d], n%=%d", Len(fred), n%);

for n% := 0 to 9 do
    var x := 3;         'Declare and initialize in for loop
    x += 1;
    if x <> 4 then Message(x) endif;
next
```

The first line creates a variable `sz%` initialized to the constant 4. The following `if` statement (equivalent to `if false then` as we are not yet beyond Spike2 version 20) includes a declaration of two variables `fred` and `n%`. This is followed by a call to a built-in function, `Message()` to display the size of the array `fred` and the value of `n%`.

The following `for` loop illustrates the effect of declaring a variable and initializing it in a loop.

If you compile this and check the View menu Annotate option you get:



The text enclosed in each box represents the code generated by the previous script line. This is the code that is generated by the script compiler and that is interpreted by the script runtime system. Unless you are really interested in how this works, you can safely skip the remainder of this topic!

What is a compiled script?

Spike2 compiled script code is simply a list of numbers. Each number (currently in the range 0-65535) is the index of an object in an object table that is maintained by the script system. The object numbers 0-20 refer to the constant integer numbers 0-20. Object number 21 to 132 (at version 11.00) are instructions (for example `Int+` : *Pop the top two items off the stack and add them as integers and push the result back on the stack*). This is followed by objects for all the built-in script commands. Up to this point, the object list is of permanent objects. More objects are added when a script runs. There are objects for all the global variables and for user-defined `Proc`, `Func` and `Object` items. When a user-defined function runs, objects are added to the object table for local variables. There are also temporary objects created and destroyed to hold intermediate results, such as the result of adding two values; these are not part of the object table.

Each object has a set of actions that include an *Execute* action that defines what the object does when the object number is executed by the script. The standard Execute action for everything except the instruction objects is to push their object number onto the stack. The instruction object Execute action depends on the instruction.

When you compile a script, in addition to building the list of numbers the script compiler also generates a symbol table that holds entries for all user-defined names in the script. This symbol table is used during compilation to identify all the user-defined names and is used at run time to create temporary local variables for user-defined functions and by the script debugger.

How does a script run?

Unlike the Spike2 program, which is compiled into machine code and executed by the computer CPU directly, the script language is interpreted. It runs in a virtual machine that has a program counter (`pc`) and a stack pointer (`sp`). The program counter is an index into the list of compiled numbers that represent the script. The stack holds pointers to objects. The stack pointer is an index into the available stack space.

When you push an item onto the stack it is stored at the current `sp` index and `sp` increases by 1. When you pop an item off the stack the `sp` is decremented by 1 and the item at that index is retrieved. Some objects are temporary and have to be created when required and destroyed when no longer required. To manage this, all objects have a reference count. Permanent objects, such as global variables, are created with a reference count of 1. Temporary objects, such as the result of adding two number, are created with a reference count of 0. Each time an object is pushed onto the stack, or a references to it is taken (for example making a sub-array from an array), the object reference count is incremented. Each time an object is popped from the stack or a references is removed, the object reference count is decremented. If the reference count reaches 0, the object is deleted.

When you run a program, the program counter is set to 0. The process of running a script is basically:

1. Fetch the number from the compiled script at index `pc`; this is an object index. Increment `pc`.
2. Look up the object at the index, run the Execute action of the object, go to step 1.

In reality, the script loop is more complicated as it has to make provision for debugging operations and also allow periodic checks for the user trying to break out of a script that would otherwise never stop running.

Explanation of the illustrated code

Each source line that generates script output is followed by text in a box holding a list of mnemonics for the generated instruction. The number followed by a colon at the start of each annotation line is the instruction number in the compiled script.

The first thing to notice is that the first 4 lines of script generate no code at all.

```
var sz% := 4;
```

This creates the global variable `sz%`, initialized to the constant value 4. This is created with this value when the script starts up and is not in any kind of loop, so there is no need for any runtime code to set its value.

```
if _Version > 2000 then
    var fred[sz%], n% := sz%;
endif
```

The `if` statement condition is a constant expression (known at compile time), so the compiler knows that the following code does not execute at runtime. But, it contains a `var` statement, so the variables `fred` and `n%` are created. As the size of `fred` is set by a variable, it is considered unknown at compile time. Similarly, `n%` is initialized by a variable, so it is also considered unknown, so the size of `fred` and the value of `n%` are both initialized to the default value of 0. No runtime code is required. This seems odd, but the compiler sees the `var` statement in two distinct parts: firstly declaration of a vector and an integer, secondly setting the vector size

from a variable and the integer value from a variable. The second part requires runtime code, but we have placed this inside an `if` statement that is never true, so the runtime code is omitted and the code is equivalent to `var fred[], n%;` and no code generation.

```
Message("fred[%d], n%=%d", Len(fred), n%);
```

This is the first line that generates any code. This is a call to the built-in function `Message()` with three arguments (a literal text string, the result of a call to the built-in `Len()` function and the variable `n%`. To make sense of this you have to know that the script language has a *stack*, which is a list of items held in order of addition. You can push items onto the stack and pop them off, you can also refer to items relative to the current end of the stack.

The code for this looks complicated, but we can break it down into two parts: the call to `Len()` and the call to `Message()`. All calls generate the same basic code, so a function defined as `Func fName(arg1, arg2, ..., argn)` would generate:

```
arg1 arg2 ... argn arg_count fName Call(variant)
```

The arguments are identified by their object number. The `(variant)` is present if this is a built-in function with multiple possible arguments lists and is used to identify which argument list is in use. The `arg_count` is the number of arguments present in this call. When these instructions are executed they push the arguments on the stack followed by the argument count and the object that identifies the function. The `Call` instruction pops the function off the stack and runs it. The function is expected to pop all the arguments off the stack, and if this is a `Func` (rather than a `Proc`), it pushes the result on the stack.

The second argument to `Message()` is `Len(fred)`. This generates the code:

```
fred const=1 Len Call(2)
```

This pushes the array object `fred` on the stack, followed by the argument count (1) and calls the second variant of the built-in function `Len()`. This function pops the arguments off the stack and pushes the result of the call to `Len(fred)` on the stack. The `Message()` call is then:

```
$0"fred[%d], n%=%d" expr n% const=3 Message Call
```

Where `expr` stands for the call to `Len(fred)` that will push the length onto the stack. The `$n"text"` is how we write literal strings . The `n` value is the index of the string in the internal script compiler symbol table. The first string gets the index 0, the second is 1 and so on. This code pushes the string object on the stack, the `expr` code pushes the object holding the length of `fred` on the stack, then the `n%` object and the number of arguments (3) and the object for `Proc Message()`. The `Call` statement collects its arguments from the stack and has no result to push.

The following block of code is a for loop:

```
for n% := 0 to 9 do
    ...
next
```

We will start by ignoring the loop contents. The loop itself is implemented by:

```
n% const=0 const=9 const=1 MakeIntFor 16
   ...
IntNext
```

A `for` loop is set up by pushing the for loop variable object, followed by objects holding the loop start, end and step (in this case 1 as no step was specified). The instructions that manage the loop are `MakeIntFor` (which pops the loops arguments off the stack and pushes a loop managing object onto the stack) and `IntNext`, which deals with jumping back to the start of the loop and cleaning up the loop managing object. The value 16 is not an object, it is the distance, in instructions, from the `MakeIntFor` instruction to the end of the loop. It is required to cope with the case of a loop that runs 0 times so execution must jump over the body of the loop.

Within the loop, the generated code is:

```
    var x := 3;
    x const=3 Flt:=
```

Although the variable x is a global and is initialized to a constant and so will be created with the value 3, it is in a loop, so needs re-initializing each time around. The generated code pushes the `x` object on the stack and the value 3 and the `Flt:=` (*real number assignment*) instruction pops off the value and the `x` object and assigns the value to the **x** object.

```
x += 1;
x const=1 Flt+=
```

This increments the value of x by 1. The generated code pushes x on the stack followed by 1. The Flt+= (real number add) instruction pops the two objects off the stack, adds the values of the objects and pushes the result back on the stack.

```
if x <> 4 then Message(x) endif;
x const=4 FltNotEq JumpRelImmF 4
x const=1 Message Call
```

The final part of the loop evaluates the expression x <> 4. The objects for x and 4 are pushed on the stack, then the FltNotEqu (*pop top two stack items and push true if not equal, false if equal*) instruction runs. The JumpRellImmF (*jump relative immediate if false*) pops the result of the comparison from the stack and if it is false, jumps forward 4 instructions. If the jump is not taken, there is a call to the Message() command, this time with a single argument.

There is always an extra Halt instruction at the end of the script (not shown by Annotate as there is no source line for it). This is why the script window states that there are 33 instructions, but only instructions 0 to 31 are listed.

Annotation items

The annotation text in rectangular boxes is composed of the following elements, each one corresponding to one number in the compiled script:

n:	At the start of a line the n is the index into the compiled script of the first instruction following the colon.
const=n	n is one of the integers 0-20 that are built in objects.
$n"string"	A string literal in the script.
#n(1.23)	A real number literal in the script.
%n(99)	An integer literal in the script (not 0-20 as these are built in).
6, -12	An integer number without qualification is not an object but is stored as an immediate value in the script. Used for jump offsets.
Name	The name of a built-in function or a user-defined item.
Instruction	A mnemonic for one of the script instructions. These are not documented here.

Grid view commands

These are the View menu and mouse click commands that apply to Grid views.

Edit grid cell contents

There is always a current cell, even when a range of cells is selected. You can type into the current cell or double click on a cell to edit the contents. When you are editing a cell, a text caret appears in the cell and cut, copy and paste commands apply to the cell. When you are not editing a cell, cut copy and paste commands apply to the selected range of cells.

Optimise Column Widths

This command scans all the data in the grid and sets each column to be wide enough to display all the data in each cell. The script language equivalent is GrdColWidth().

Align Column

Each column has an alignment of Left, Centre or Right. The script language equivalent is GrdAlign(). This applies the selected alignment to the selected columns.

Show Column/Row Header

You can turn off and on the column and row headers. The script language equivalent is GrdShow().

Grid Size...
This opens the Grid Size dialog in which you can change the grid size. Changing the grid size cannot be undone.

Context menu commands ([10.05] onwards)
If you right click in the grid area you can open a context menu to operate on the selected cells to set text and background colours and reset the selected colours. From [10.06] you can cut, copy and paste text, resize the grid and fit the view frame to the cell area.

Double-click on header row ([10.05] onwards)
This opens a dialog in which you can edit the column title.

Double-click in non-cell area ([10.05] onwards)
If the view is larger than the cell area, if you double-click in the grey non-cell area the view will resize to fit the cell area. The script language equivalent is `WindowSize()`.

See Also:
Toolbar and Status bar

Enlarge View Reduce View

Y Axis Range

X Axis Range

X Axis Extra Time

Standard Display

Show/Hide Channel

Vertical Markers

TextMark List

Pen Width

File Information (Time View)

File Information (Result View)

Channel Information...

Channel Image

Info windows

XY Key Options

XY Autoscale

Trigger/Overdraw

Channel Draw Mode

Multimedia files

Spike Monitor

Font

Use Colour and Use Black And White

Change Colours

Colour scale dialog (Sonogram, Density map colours)

Folding

Show Gutter

Show Line Numbers

ReRun

Annotate

Edit column header

Double-click the header of a column to open the Grid column header dialog to edit the column title.

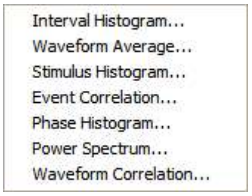
Normally column headers are alphabetic letters, but they can be modified to text of your choice. You can revert to the original by setting the text blank or by clicking the Reset button. Changes made by this dialog can be undone and also recorded. The script language equivalent of the dialog is: `GrdSet("header", col%, -1);`

9: Analysis menu

Analysis menu

This menu creates result and XY windows that hold analysed data from time view channels, manages the memory buffers, deletes, duplicates and calibrates channels, gives access to the marker filter control and manages spike shapes. It also holds the digital filtering command.

New Result View



This command is available when the current window is a time view. It is a pop-up menu from which you can select an analysis type. This leads into a dialog where you define the parameters to construct a result window.

A result window holds arrays of data that can be drawn as histograms or waveforms. There is one data array for each channel you analyse. Some types of analysis can store additional data, for example the peristimulus histogram can store event times for a raster display and the waveform average can store extra data so that you can display error bars in the result.

Once you have set the required values in the dialog box, Spike2 creates a result window with all data values set to zero. A new dialog appears to prompt you to select a region of the original time window to analyse. The results of analysing different areas can be summed.

The maximum size of each channel data array (number of bins) is limited to 100,000,000. This size was chosen to match the maximum script language array size and to limit the data processing time. You should be aware that you are very likely to run out of memory if you create a result view of this size. Even if you succeed in creating a view with such a size, many operations work with copies of the result view data, and each time a copy is made, there is the risk of running out of memory.

The following analyses are implemented: interval histogram, frequency histogram, waveform average, peristimulus time histogram, event time cross correlation, event phase histogram, power spectrum and waveform correlation.

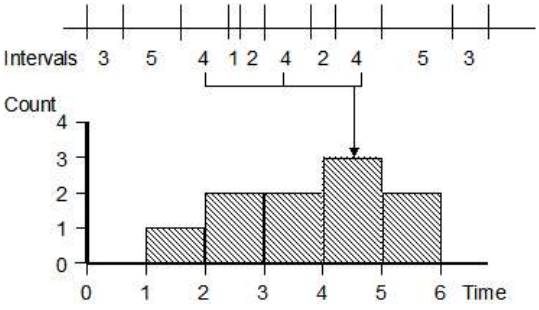
Each time you create a new result view, a *process* is associated with the time view to manage the data processing. You can associate multiple processes with a time view.

Measurements

You can also create processes that take measurements from a time view and send then to an XY view or event to a new channel in the time view.

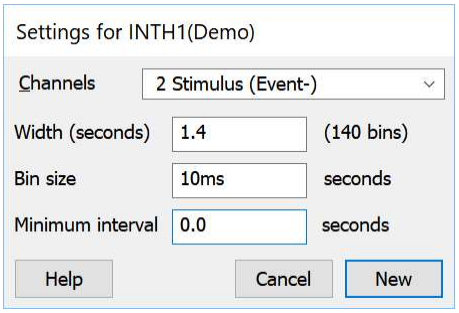
Interval histogram

An interval histogram (INTH) displays a histogram of the intervals between events on a selected channel. All times in a Spike2 data document are expressed as a multiple of the basic time unit set when the data document was created. Therefore you cannot get better time resolution than the basic time unit (usually set to around 1 µs).



The picture illustrates how an interval histogram is formed. To make the diagram easy to understand, we have shown several events with all the intervals between them as a whole number of seconds. The interval histogram formed from these events has bins set with a width of one second.

The interval between each pair of events is divided by the bin width to give the bin number to increment (fractional bin numbers are rounded down, the first bin is bin 0). In this case, the bin width is 1, so there is no rounding. In a more realistic case with an interval of 2.3 milliseconds and a bin width of 1 millisecond, bin number 2 (spanning the time period 2 to 3 milliseconds) would be incremented.



Event times are rounded down to the nearest base time unit, so an event time of `t` base units means that the actual time was greater than or equal to `t` and less than `t+1` units. An interval of `n` units means that the real time interval between two events was greater than `n-1` units and less than `n+1` units. If you need to form interval histograms of very short periods, make sure that you have set the duration of the base time unit short enough to resolve the information you require.

The Channel field selects one or more event channels; type a list of channels or select from the drop down list. Each channel generates a histogram in the result view. If you type a channel list or use the selected channels, the order of the channels in the result view is the order of channels as you enter them.

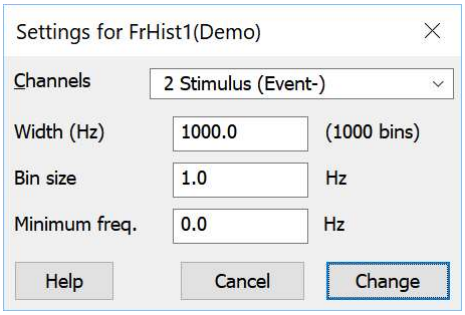
Width sets the time width of the histogram from left to right. The Minimum interval field is the smallest interval to put in the histogram and sets the left hand end of the x axis. It must be positive and is usually zero. Bin size is rounded to the nearest multiple of the underlying time units used in the time window. The number of bins (Width/Bin size) must be at least 1. The maximum size is limited only by available system memory. You can enter the Width, Bin size and Minimum interval fields in milliseconds by following the numeric value with `ms`. You can force microseconds with `us`. For example, `0.001`, `1ms` and `1000us` are all the same value.

The New button accepts the values in the dialog and creates a new result window. This dialog can also be activated by the Process Settings menu command, in which case the New button is replaced by a Change button. Change clears any previous results.

Click the New or Change button to open the Process dialog in which you set the time range of data to analyse. Only events that fall within the time range are considered, intervals that start or end outside the time range are ignored.

Frequency histogram

This analysis generates a histogram of the reciprocal of the intervals between consecutive events on a channel. It is very similar to the INTH analysis, but instead of binning the intervals between events, it bins the reciprocal of the event intervals. Events are sampled to a time resolution set by the file resolution, typically around 1 microsecond.



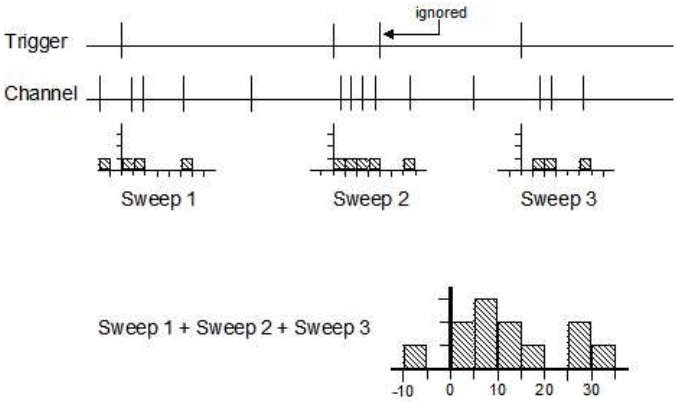
The Channel field selects one or more event channels; type a list of channels or select from the drop down list. Each channel generates a histogram in the result view. If you type a channel list or use the selected channels, the order of the channels in the result view is the order of channels in this field.

Width sets the frequency width of the histogram from left to right. The Minimum freq. field is the lowest frequency to include in the histogram and sets the left hand end of the result view x axis. It must be positive and is usually zero. Bin size sets the width of each bin in the histogram, in Hz. The number of bins (Width/Bin size) must be at least 1.

The New button accepts the values in the dialog and creates a new result window. This dialog can also be activated by the Process Settings menu command, in which case the New button is replaced by a Change button. Change clears any previous results.

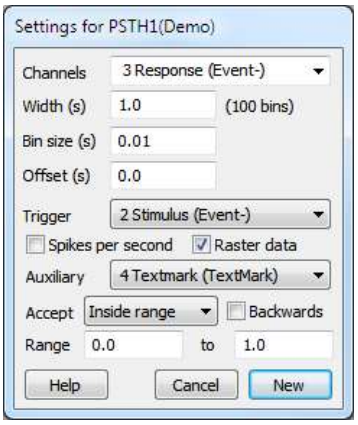
Click the New or Change button to open the Process dialog in which you set the time range of data to analyse. Only events that fall within the time range are considered, intervals that start or end outside the time range are ignored.

Peri-stimulus time histogram



A peristimulus time histogram (PSTH) forms a histogram of events on one or more response channels around a stimulus event on a trigger channel. Each source channel generates a separate histogram.

Each trigger event that does not lie within the previous sweep generates a new sweep. Trigger events that fall in the previous sweep are ignored. If you require overlapped sweeps, use the event correlation analysis. Sweeps are accumulated to produce a histogram. You can scale the result into spikes per second rather than event counts.



In the Settings dialog, the Channel field selects the channels to analyse. The Width field sets the width of the histogram in seconds and the Bin size field sets the width of each bin. Their ratio sets the number of bins in the histogram. The Offset field sets the pre-trigger time to display. If the time offset is 0.5, the histogram time axis starts at 0.5 seconds. All times are rounded to the nearest multiple of the basic time unit. You can supply values for the Width, Bin size and Offset fields in milliseconds by appending `ms` to the value or in microseconds by appending `us`, for example `0.01` seconds can be entered as `10ms` or as `10000us`.

The Trigger field sets a channel to use as the trigger, or you can select no trigger. If you do not select a trigger channel, the process command uses the start time as the trigger time for a single sweep. If you select a trigger channel, the process command defines a time range to search for triggers.

Spikes per second

If you check the Spikes per Second box, the histogram shows equivalent spike rate rather than the event count in each bin. To change from event count to rate, Spike2 divides the event count by the number of sweeps to give the spikes per sweep, and then divides the result by the bin width to give spikes per second.

You can change between count and spikes per second after you have created the result view and analysed data. Click on the result view, open the Process settings dialog and change the state of the Spikes per second check box, then click Change.

Raster data

If you check the Raster data box, the new result view stores the times of all the events that are added into histograms and you can to draw the result as a raster display. When you create the result view, Spike2 automatically duplicates each result channel and draws the duplicated channel in raster drawing mode above the histogram. As each event uses memory and takes time to draw, the maximum number of events is limited by the memory in your system and your patience. This should not be an issue unless you work with hundreds of thousands of spikes with the result view raster enabled.

Once you have created the result window and analysed data you can delete the raster data by opening the Process settings dialog and clearing the Raster data check box. You cannot add raster data to an existing result view without clearing all the histograms.

Auxiliary measurements

You can take one measurement per sweep from an auxiliary channel. If you choose a waveform or RealWave channel, the measurement is the channel value at the trigger point or 0 if there is no waveform data at the trigger. For all other channels types, the measurement is the time between the sweep trigger point and the first event on the auxiliary channel after or before the trigger. Check the Backwards box to search before the trigger. Pre-trigger times are negative, times after it are positive. If no event is found, 0.0 seconds and the maximum time possible in the file are used as virtual event times.

Accept sweeps

You can accept all sweeps, or only those for which the measured value lies inside or outside the values in the Range fields. Sweeps with no data found are treated as outside the range. When the auxiliary channel holds waveform or RealWave data the units of the range fields are the channel units, for all other types the units are seconds. When the units are seconds, you can enter values in milliseconds by appending `ms` to the value.

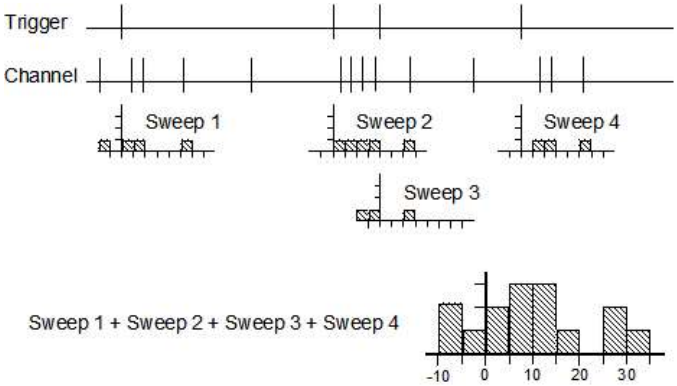
Sorting rasters

If you enable raster data, you can sort raster sweeps based on event latencies or waveform values. You set the sort mode from the Draw Mode dialog. Each raster sweep can store four sort values. The auxiliary measurements system uses the first sort value. The `RasterSort()` script command can access all the sort values.

Raster symbols

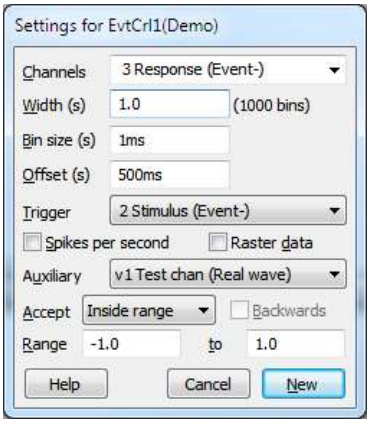
Each raster sweep can store eight event times that you can choose to display as symbols with the Draw Mode dialog. If the auxiliary measurement is an event time, and an event was found, the time is saved as the first symbol time. Script users can access all eight symbol times with the `RasterSymbol()` command.

Event correlation



This command performs event cross-correlations and event auto-correlations between a trigger channel and one or more other event channels. A cross-correlation produces a measure of the likelihood of an event on one channel occurring at a time before or after an event on another channel. The result can be displayed as a spike rate or as a count.

Each trigger generates one sweep of analysis (unlike a PSTH where a trigger event that falls within a sweep is ignored). The zero times of each sweep are aligned, and then the sweeps are accumulated to form a histogram.



The Channel field sets a channel or a channel list to analyse. The Width field sets the width of the histogram in seconds. The Bin size field sets the width of each bin in seconds. The ratio of these fields sets the number of bins in the histogram. The Offset field sets the pre-trigger time to display in seconds, so if the time offset is offset, the histogram time axis starts at a time of -offset. All times are rounded to the nearest multiple of the basic time unit. You can set the Width, Bin size and Offset fields in milliseconds or microseconds by appending ms or us to the entered value. For example, `0.001`, `1ms` and `1000us` all represent the same time value.

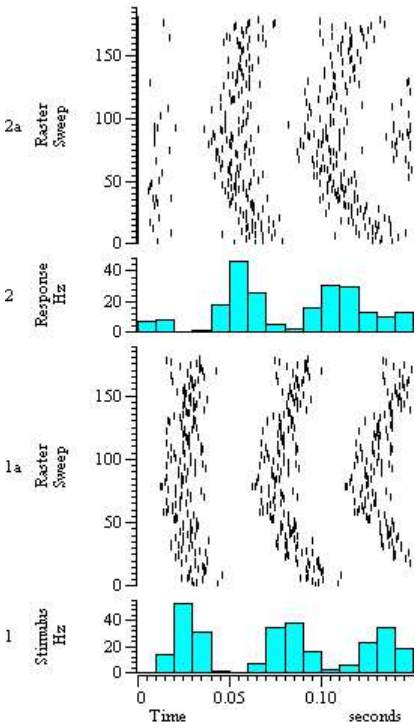
The Trigger field is a drop down list in which you can select a trigger channel or Manual mode. In Manual mode, the process command start time is the trigger time for one sweep. With a trigger channel, the process command sets a time range to search for trigger events.

Spikes per second

If you check the Spikes per Second box, the histogram shows equivalent spike rate rather than the number of events in each bin. To change from event count to rate, Spike2 divides the number of spikes in each bin by the number of sweeps to give the spikes per sweep, and then divides the result by the bin width to give spikes per second.

Raster data

If you check the Raster data box, the new result view stores the times of all the events that contributed to the histogram and you can use raster drawing mode for the result view channel. When you create the result view, Spike2 duplicates each channel and draws the duplicate in raster mode above the histogram.



As each event uses memory and takes time to draw, the maximum number of events is limited by the memory in your system and your patience. This should not be an issue unless you work with hundreds of thousands of spikes with raster enabled.

Auxiliary measurements

You can take one measurement per sweep from an auxiliary channel. If you choose a waveform or RealWave channel, the measurement is the channel value at the trigger point or 0 if there is no waveform data at the trigger. For all other channels types, the measurement is the time between the sweep trigger point and the first event on the auxiliary channel after or before the trigger. Check the Backwards box to search before the trigger. Pre-trigger times are negative, times after it are positive. If no event is found, 0.0 seconds and the maximum time possible in the file are used as virtual event times.

Accept sweeps

You can accept all sweeps, or only those for which the measured value lies inside or outside the values in the Range fields. Sweeps with no data found are treated as outside the range. When the auxiliary channel holds waveform or RealWave data the units of the range fields are the channel units, for all other types the units are seconds. When the units are seconds, you can enter values in milliseconds or microseconds by appending `ms` or `us` to the value.

Sorting rasters

If you enable raster data, you can sort raster sweeps based on event latencies or waveform values. You set the sort mode from the Draw Mode dialog. Each raster sweep can store four sort values. The auxiliary measurements system uses the first sort value. The `RasterSort()` script command can access all the sort values.

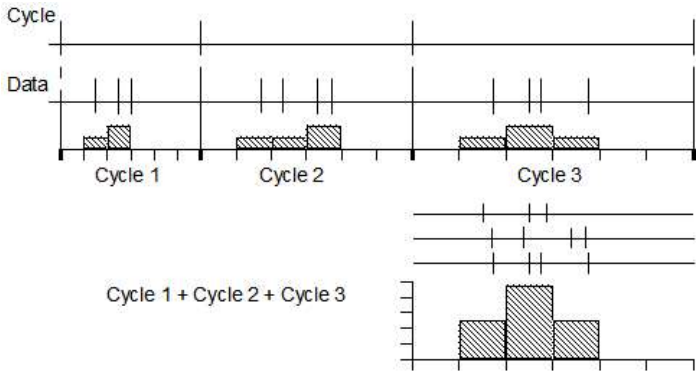
Raster symbols

Each raster sweep can store eight event times that you can choose to display as symbols with the Draw Mode dialog. If the auxiliary measurement is an event time, and an event was found, the time is saved as the first symbol time. Script users can access all eight symbol times with the `RasterSymbol()` command.

Auto-correlation

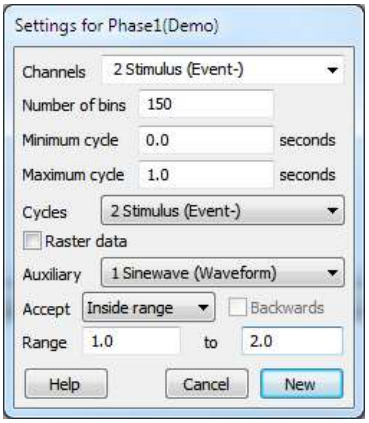
For an auto-correlation, set the Channel and Trigger fields to the same channel. The analysis for an auto-correlation has one difference from that for a cross-correlation. In an auto-correlation, the correlation of each event with itself at time 0 in the histogram is ignored.

Phase histogram



A phase histogram is used to show how events are distributed with respect to a cyclical process that may vary in cycle time. One event channel marks the start and end of cycles. The end of one cycle is the start of the next. Events on another channel are placed in bins depending upon their position within each cycle.

You can limit the range of cycle times to consider for inclusion in the histogram, either to eliminate impossible data or to show how responses vary with the cycle times.



The Channels field sets the channels to analyse. The Cycle channel sets the channel with cycle markers. The Number of bins field sets the bins per cycle. The width of each bin depends on the duration of each cycle. The Minimum cycle time and Maximum cycle time fields exclude cycles that are too long or too short to belong to the data and are set in seconds. You can set these values in milliseconds or microseconds by appending `ms` or `us` to the value. Check the Raster data box to save the times of all events and duplicate each result channel in raster mode.

In the Process command, the start and end times determine the time range of the data in the Cycle channel to search for cycle markers. Cycles that are longer than the maximum time or shorter than the minimum time are ignored. If no Cycle channel is supplied, the start and end times are used as the start and end of a single cycle.

Auxiliary measurements

You can take one measurement per sweep from an auxiliary channel. If you choose a waveform or RealWave channel, the measurement is the channel value at the trigger point or 0 if there is no waveform data at the trigger. For all other channels types, the measurement is the cycle position of the first event on the auxiliary channel after or before the cycle start. Check the Backwards box to search before the trigger. Pre-trigger positions are negative. The cycle start position is 0 degrees, the cycle end position is 360 degrees. If an event is at time *Te* and the cycle start and end times are *Cs* and *Ce*, the event position is 360*(*Te*-*Cs*)/(*Ce*-*Cs*). If no event is found, 0.0 seconds and the maximum time possible in the file are used as virtual event times.

Accept sweeps

You can accept all sweeps, or only those for which the measured value lies inside or outside the values in the Range fields. Sweeps with no data found are treated as outside the range. When the auxiliary channel holds waveform or RealWave data the units of the range fields are the channel units, for all other types the units are degrees.

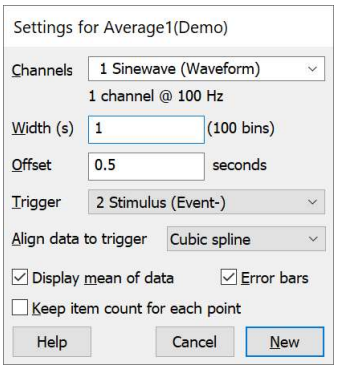
Sorting rasters

If you enable raster data, you can sort raster sweeps based on cycle positions or waveform values. You set the sort mode from the Draw Mode dialog. Each raster sweep can store four sort values. The auxiliary measurements system uses the first sort value. The `RasterSort()` script command can access all the sort values.

Raster symbols

Each raster sweep can store eight event times that you can choose to display as symbols with the Draw Mode dialog. If the auxiliary measurement is an event time, and an event was found, the time is saved as the first symbol time. Script users can access all eight symbol times with the `RasterSymbol()` command.

Waveform average



This command averages waveform channels with respect to a trigger signal, or if no trigger is set, averages waveform channels with respect to user-defined trigger times. The Channels field selects the channel, or channels to process. You can select channels from the drop-down list, type a channel list, for example `1..100`, or you can choose Selected from the drop down list and then select channels in the Time view. The field below the Channels selector displays the number of channels in the average and the sample rate of the first channel in the list. Note that all channels must have the same sample rate to average multiple channels. You can re-sample channels to match another channel with the Channel Process command or with a virtual channel.

The Width field sets the time range spanned by the average, in seconds. The Offset field sets the start time of each sweep before the trigger, in seconds. You can provide the Width and `Offset` values in milliseconds or microseconds by appending `ms` or `us` to the value. The Trigger field sets the source channel for trigger times or Manual trigger mode.

Display mean of data

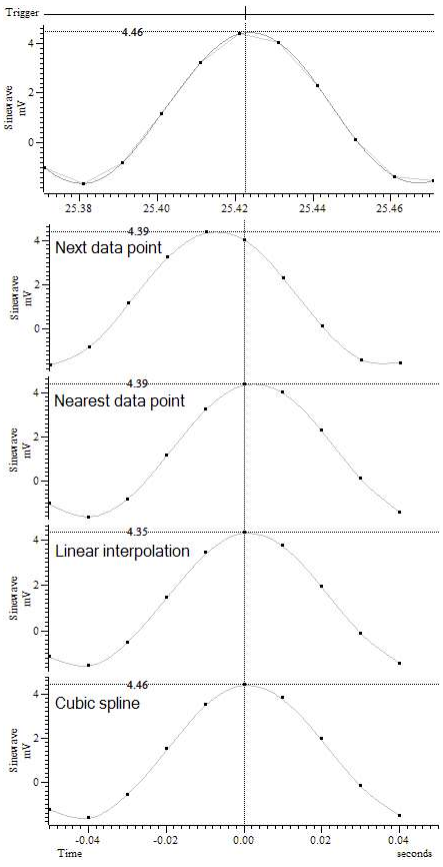
The Display mean of data chooses between a display of the mean of all the sweeps or a display of the sum of all the sweeps. If you return to this dialog after forming an average you can change this setting to redraw the result without needing to process the data again.

Error bars

If you check the Error bars box, extra information is saved with the result so that you can display the standard deviation and standard error of the mean of the resulting data. The Channel Draw Mode command controls the display of the error information.

Align data to trigger [6.01]

In general, trigger times will fall between samples of the waveform channel. If you are looking at features that are only a few sample points in duration, for example transient responses, the data may change significantly between the sample points. You have a choice of four methods to set the alignment of the data to the trigger.



Next data point

The data is aligned at the next sample after the trigger time. This is the method used by all Spike2 versions before [6.01].

Nearest data point

The data is aligned at the nearest sample to the trigger point. This is the best method to use if you don't want to interpolate.

Linear interpolation

The data values are aligned to the trigger by linearly interpolating between the data points.

Cubic spline

The data values are aligned to the trigger by cubic spline interpolation. This is the best method to use if your data contains no components at frequencies above half the sample rate. It is also the slowest method.

The best method to use depends on your data. If your data is known to contain no frequencies above half the sample rate (which is what you would hope to be the case), using a cubic spline will produce the best results. If this is too slow, you could try linear interpolation, which is faster to compute, but which will tend to reduce the amplitude of narrow features.

If your data contains transients with frequency content above half the sampling rate, cubic interpolation will become inaccurate and the best you can do is choose the Nearest data point method. The Next data point method is provided for backwards compatibility with older versions of Spike2.

Keep item count for each point

If you check this box, each point in the result view keeps track of the number of items that were averaged to create it. If you do not check the box, all points are presumed to have averaged the same number of items (the sweep count). The box makes a difference when sweeps of data are truncated or interrupted by gaps or the start or end of the channel data. If you do not check the box, missing data is presumed to hold zeros, which may cause discontinuities in the average. If you check the box, missing data makes no contribution to the average. If you use this option, saved result view files will not be compatible with versions of Spike2 before 5.16 and 6.00.

New (Change) button

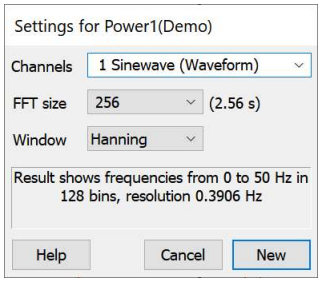
The New button (or Change when used from the Process Settings command) closes the dialog, creates a new result window and opens the Process dialog. With a trigger channel, the events between the start and end times set in the process dialog are triggers. In Manual mode, the start time is the trigger for a single sweep.

Sweep count

The result window for an average keeps track of the number of data sweeps that have been added into the average. A sweep is counted if a least one channel adds at least one data point into the average. It displays the mean waveform by accumulating the data and dividing by the number of sweeps or, if the Keep item count... box was checked, the item count for each point.

The start of the section of data to add to the average is found by subtracting the time in the Offset field from the trigger time. The data channel is then aligned to the trigger by the method set in the Align data to trigger field. If there is a gap in the waveform data, such that there is no data point that falls within time from the start of the section to the start plus the width of the average, no data is added.

Power spectrum



This command creates a result window that holds the power spectrum of a section, or sections of data. The result of the analysis is scaled to RMS power, so it can be converted to energy by multiplying by the time over which the transform was done. You can transform multiple channels, but they must have the same sample rate. Spike2 uses a Fast Fourier Transform (FFT) to convert the waveform data into a power spectrum.

The FFT is a mathematical device that transforms a block of data between a waveform and an equivalent representation as a set of cosine waves. The FFT that we use limits the size of the blocks to be transformed to a power of 2 points in the range 16 to 262144. You set the FFT block size from a drop down list. The result window ends up with half as many bins as the FFT block size. When you use the Process command, the selected area must hold at least as many points as the block size, otherwise no analysis is done.

The result window spans a frequency range from 0 to half the sampling rate of the waveform channel. The width of each bin is given by the waveform channel sampling rate divided by the FFT block size. Thus the resolution in frequency improves as you increase the block size. However, the resolution in time decreases as you increase the block size as the larger the block, the longer it lasts.

Windowing of data

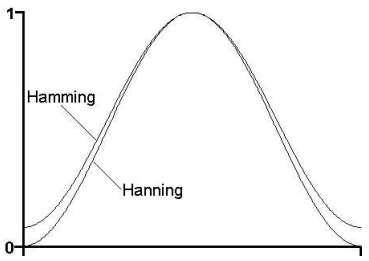
The mathematics behind the FFT assumes that the input waveform repeats cyclically. In most waveforms this is far from the case; if the block were spliced end to end there would be sharp discontinuities between the end of one block and the start of the next. Unless something is done to prevent it, these sharp discontinuities cause additional frequency components in the result.

The standard solution to this problem is to taper the start and end of each data block to zero so they join smoothly. This is known as *windowing* and the mathematical function used to taper the data is the *window function*. The use of a window function causes smearing of the data, and also loss of power in the result.

You can find all sorts of windows discussed in the literature, each with its own advantages and disadvantages; windows shaped to have the smallest side-lobes spread the peak out the most. By reducing the side-lobes you decrease the certainty of where any frequency peak actually is (or the ability to separate two peaks that are close together). Spike2 implements the following windows:

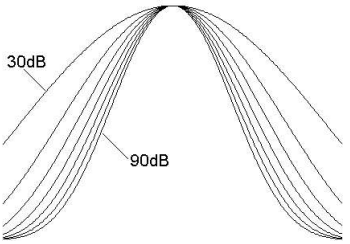
No Window — You might use this if there is one sine wave, or if more than one, they all have similar amplitude. This has the sharpest spectral peaks, but the worst side-lobes.

Han or Hanning — This is a good, general purpose, reasonable compromise window. However, it does throw away a lot of the signal. It is sometimes called a "raised cosine" window and is zero at the ends. If you are unsure about which window would be best for your application, try this one first.



Hamming — This preserves more of the original signal than a Hanning window, but at the price of larger side-lobes.

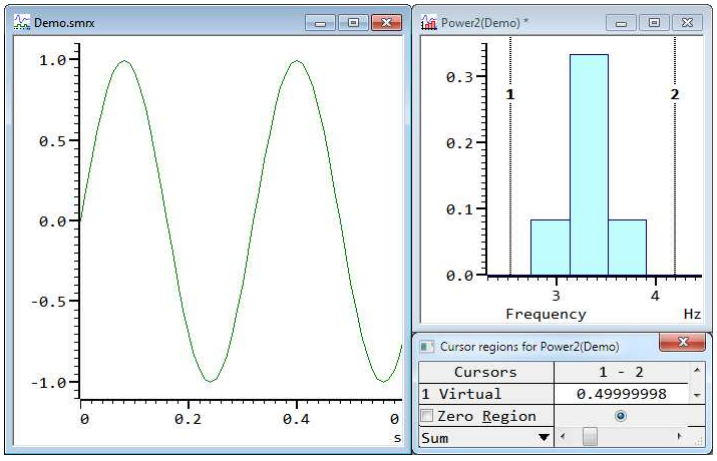
Kaiser — These are a family of windows calculated to have known maximum side-lobe amplitude relative to the peak. Of course, the smaller the side-lobe, the more signal is lost and the wider the peak. We provide a range of windows with side-lobes that are from 30 to 90 dB less than the peak.



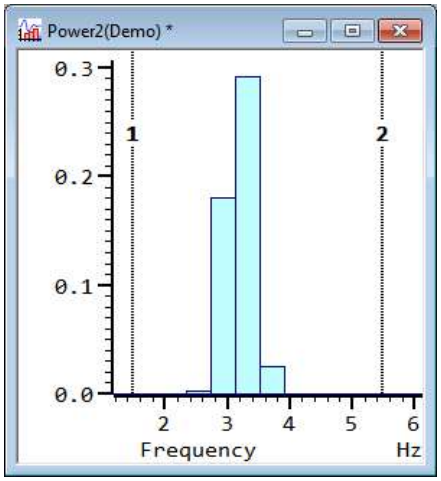
Overlapping data blocks [7.10]
From Spike2 version 7.10, each source data block overlaps the previous, contiguous block by 50%. This is sometimes referred to as Welch's method. Previously, blocks were not overlapped, which is sometimes called Bartlett's method. Overlapping the blocks mitigates the effect of under-weighting data at the start and end of each block, especially with the Hanning and Hamming windows, where it weights all data the same. However, it also slows down the processing by up to a factor of two. You can disable the overlap in the Edit menu Preferences in the Compatibility tab. When overlapping blocks, we will increase the overlap, when needed, to make sure that all data in the range contributes to the power spectrum.

Power spectrum of a sine wave

If you sample a pure sine wave of amplitude 1 Volt and take the power spectrum, you will not get all the power in a single bin. You will find data spread over a few bins, and the sum of the these bins will be 0.5 Volt^2. The factor of 2 in the power is because we give the result as RMS (root mean square) power. This is illustrated by the example below where we have created a 1 Volt sine wave using a virtual channel sampled at 100 Hz with the expression `WSin(3.125,0)`. We have formed the power spectrum of the signal using a 256 point transform using a Hanning window and zoomed in around the bins where the result lies.



We have used the cursor regions dialog to sum the power, getting 0.49999998, which is as close to 0.5 as one could hope for. If you repeat this and change the cursor regions measure to Mean in X, you will get a result of 3.125, which is the frequency we started with. We chose 3.125 Hz for this example so that the side lobes of the result were symmetrical. If you change the window from Hanning to No Window you will get an even better result with all the power in a single bin, this is because at 3.125 Hz with data sampled at 100 Hz, 256 points spans exactly 8 cycles of the data.

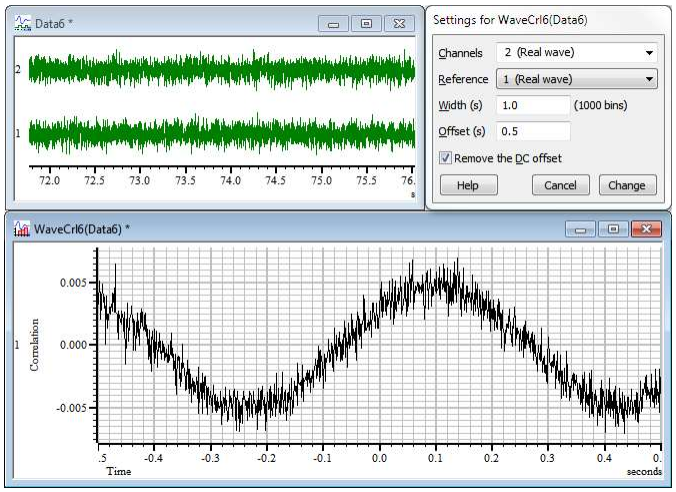


The image to the right shows the same measurements made with the frequency set to 3 Hz. In this case, the sum of the power is reported as 0.49999802 and the Mean in X value is 3.0000024 Hz, still very much in agreement with the original data. In this case, if you set No Window, 256 data points contains 7.68 cycles, and the FFT will see this as a discontinuity in the data, resulting in spurious frequency components.

If you need access to the real and imaginary components of the FFT, see the `ArrFFT()` script function.

Waveform correlation

This command creates a result view that holds the correlation between two waveform channels sampled at the same rate or the auto-correlation of a channel with itself. If you select multiple source channels (to form multiple correlations), they must all have the same sample rate; any channel with a rate that differs from the first channel in the list will be ignored.



The correlation measures the similarity of a source waveform and a Reference waveform. In the example below, the two waveforms hold random noise to which has been added a low amplitude 1.5 Hz sine wave. The sine wave in channel 2 is shifted forward by 0.1 seconds compared to channel 1, hence the peak in the correlation is at 0.1 seconds.

The correlation is calculated by multiplying the two waveforms together, point by point, and summing the products. The sum is normalised for waveform amplitudes and the number of points. This produces one result. The reference waveform moves one point to the right and the process is repeated to produce the next result. This is repeated for all the result bins. The results range between 1.0, meaning the waves are identical (except for amplitude) through 0 (uncorrelated) to -1.0, meaning identical but inverted.

The bins in the output are the same width as the sampling interval of the two input waveforms. You can choose to remove the DC component from the signals before calculating the correlation. This can be important as DC offsets can dominate the result. The correlation is calculated in such a way that you can change the setting of the DC offset removal without recalculating the correlation. To do it, open the Process Settings... dialog again, click in the box and then click the Change button.

The Process dialog sets the time range of the reference waveform. The analysis is only done for regions of data in which both waveforms exist. If you calculate correlations over long sections of data, the calculations will take some considerable time! For this reason we do not recommend this as an on-line analysis (although there is nothing to stop you using it). The number of calculations (and hence the time taken) is proportional to the number of points in the result times the number of points in the reference waveform.

To allow us to process data incrementally and to cope with gaps in the waveforms, the data processing does not use the Fast Fourier Transform method of forming waveform correlations. There is a description of how to use that method (from a script) in the Technical Notes that are installed with Spike2 if you select Extra Documentation in the Spike2 installer.

Measurements

This command is available as a button on the system toolbar and as the Analysis menu Measurements command when the current view is a time view. It leads to a menu with the following entries:

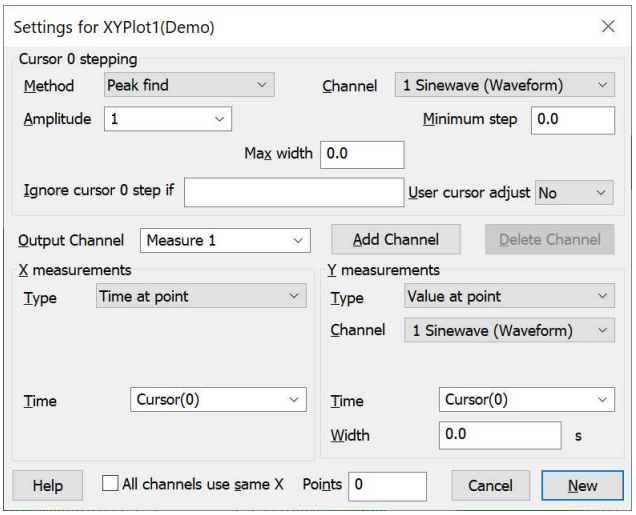
XY View...
This opens the Measurement to XY view dialog that creates a new XY view to hold measurements taken from the current view.

Measure Now
This is enabled if there is a Measure to XY process associated with the current view. It takes a measurement with the current cursor positions. This is not implemented for Measure to channel.

Data channel...
This opens the Measure to channel dialog that creates a new channel in the current time view that holds the results of measurements taken from the time view.

Measurements to XY views

You can generate trend plots in an XY view both on-line in real time and off-line. Use the Measurements... menu item and select XY view. This command generates a new XY window with one or more channels of data derived from the current time view. The script language equivalent is `MeasureToXY()`.



The basic idea is that cursor 0 steps through the data following a user-defined rule. This can be as simple as *move to the last cursor 0 position plus 1 second* or it can be a complex data-searching algorithm. Each cursor 0 move also triggers the other active cursors to search. For each cursor 0 position, we take an `x` and a `y` measurement and pass the (`x`,`y`) point to a channel in the XY view. Using this mechanism, a wide variety of measurements can be made and displayed.

A Process command sets the data region to search for cursor 0 step points, exactly as for the result view analysis commands. You can process data both on-line and off-line. Processing can take a while if there are many cursor 0 steps to do. A progress dialog appears in a long process to allow you to stop early.

From version [10.17], you can take an XY measurement manually with the current cursor positions (no cursor searches) when the Time view is the current view with the `Ctrl+M` text command (`M` for Measure) or with the Measure Now command in the Measurements menu. Note that this combination is the same as the `Enter` key. This is an experimental feature, and may change.

Cursor 0 stepping
This dialog area controls how cursor 0 moves through the data. When a new measurement process is created, if cursor 0 is already set to a suitable active cursor mode, this mode is copied to the dialog. However, the cursor 0 stepping mode you set here belongs to the measurement process; setting it here does not change the cursor 0 active mode. If you are setting a complex measurement involving multiple active cursors, *set the active cursors first* before opening the Measurement dialog. You can have multiple measurements processes, each with their

own cursor 0 stepping mode, but they all share the other active vertical and horizontal cursors settings. Most users keep things simple and have one measurement active at a time.

You can set any active cursor mode that can iterate through data. These are: Peak find, Trough find, Rising threshold, Falling threshold, Outside thresholds, Within thresholds, Slope peak, Slope trough, Turning point, Data points, Expression, Gap start and Gap end. The other fields in this section depend on the stepping mode.

You can set the Minimum Step field to 0. This will usually work, but some analyses work over a range of data, and can get "stuck", returning the same position over and over. Setting a non-zero minimum step avoids this.

You can use the Ignore cursor 0 step if field to reject a cursor 0 step and step again. If this field is not blank and evaluates without error as true (not zero), the current cursor step is skipped. Expressions usually involve cursor values, for example `Cursor(0)>Cursor(1)`. You will normally leave this field empty.

User cursor adjust
Normally, you will want the measurement process to run to completion as quickly as it can. However, if your measurement process involves active cursors other than cursor 0 you have the option of adjusting their positions. You can set this field to:

No — The measurement process rejects all measurements where active cursor searches failed or for which the Ignore cursor 0 step if field evaluated non-zero.

Yes — You are given the opportunity to adjust the cursor positions after each cursor 0 step.

On fail — You are given the opportunity to adjust the cursor positions after each cursor 0 step where one of the other active cursors failed to search (so is displayed with a ! in the label) or if the Ignore cursor step if field evaluates to a non-zero value.

This field is ignored during sampling and Measurements are taken as if this field was set to No. You also cannot use this to recover from a failure of cursor 0 steps. When the adjustment is triggered, the Cursor adjustment dialog opens, the measurement process pauses and you can drag cursors and accept or reject the current measurement or Cancel the measurement process.

Cursor 0 can be driven by other sections of Spike2: Spike shape analysis, Offline Waveform output and optionally by multimedia replay. From version [10.01] we have taken steps to reduce conflicts between these drivers. When any of the Spike shape analysis, Measurement processing, Offline waveform output or multimedia replay items starts to driver cursor 0, all the other items are notified and stop driving it. Any items in Spike2 that are linked to cursor 0 will update with the new cursor 0 position at the end of each measurement process.

Plot Channel

The trend plot can generate up to 32 channels of data in the XY view. The dialog starts with one channel. You can set the channel title in the Plot Channel box and create additional channels with the Add Channel button. The Delete Channel button deletes the current channel; you cannot delete the last channel.

X and Y measurements
These two areas have identical functionality. They generate the x and y values that are passed to the XY view for each point. The fields displayed in these areas depend on the measurement Type field. Most measurement types depend on times. For a useful result, you will set these to times relative to cursor 0, or to an active cursor that was positioned relative to cursor 0.

All channels use same X
With multiple plot channels check this box to use the same x measurement for all channels. This is most commonly used when the x measurement is a time.

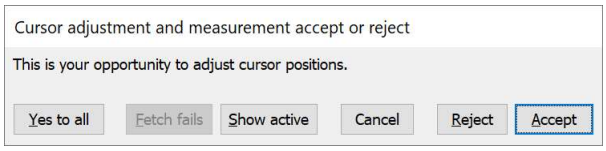
Points
Set this field non-zero to limit the displayed XY points. Points are added up to this limit, then each new point causes the oldest to be deleted. This is most useful on-line, for example to show pressure-volume curves or to display eye movements when you have horizontal and vertical input data.

New and Cancel
The New button creates the XY view and opens the Process dialog, ready to set a time range to step cursor 0 through to process the data. If you return to this dialog after creating the XY view, New becomes Change. Cancel closes the dialog.

Tabulated output

You can tabulate the results in an XY view with the Edit menu Copy as Text command.

Cursor adjustment dialog



When the dialog opens, any Progress dialog is hidden and your options are:

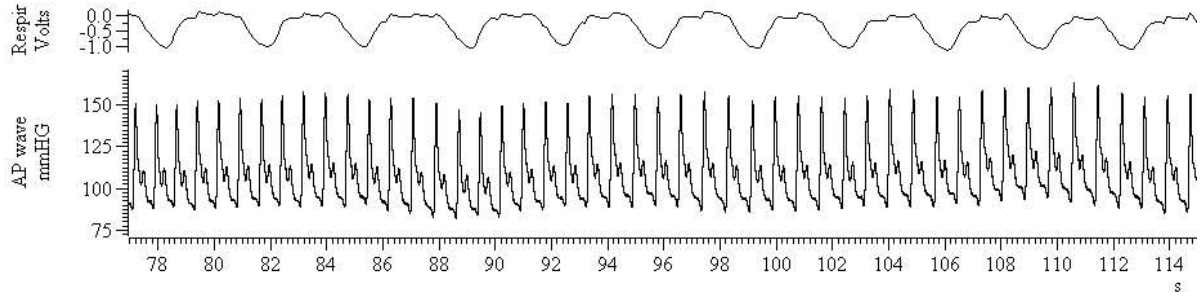
Yes to All	Accept this set of cursor positions and all future positions without displaying this dialog.
Fetch fails	This is enabled when a search for active vertical or horizontal cursors (not cursor 0) has failed. It moves all the vertical active cursors that failed to search into the visible x axis range of the time window.
Show Active	Changes the visible x axis range to make all vertical active cursors visible.
Accept	Accept this set of cursor positions. Cursor 0 will step again.
Reject	Reject this set of measurements. Cursor 0 will step again.
Cancel	Reject this measurement and stop the measurement process.

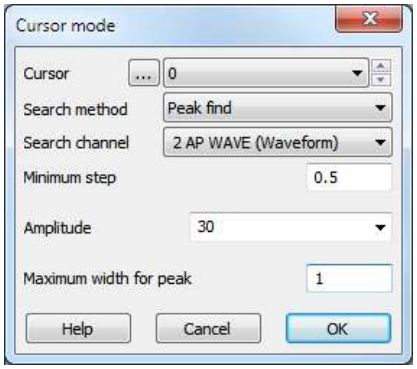
There are three reasons that this dialog appears based on the setting of the User cursor adjust field in the Measurements to XY views dialog or in the Measurements to a data channel dialog and on the results of the cursor 0 step and any active cursor searches based on the cursor 0 movement:

1. You set the field to Yes, so this dialog appears after every cursor 0 step. The dialog prompt is: This is your opportunity to adjust cursor positions. The Yes to all button is enabled. The dialog default button is set to Accept.
2. You set the field to On fail and an active cursor search to an active vertical cursor (not cursor 0) or and active horizontal cursor has failed. The dialog prompt lists the vertical and horizontal cursors that failed the active cursor search, for example: Active cursor search failed: cursor 1. Move invalid cursors to validate them. The character ! is added to failed cursor labels. The Fetch fails button is enabled. The dialog default button is set to Reject. If your measurement depends on an invalid cursor position you must validate the cursor (by dragging it), otherwise clicking on Accept will not take a measurement.
3. You set the field to On fail, the active cursor searches all succeeded and the condition set in the Ignore cursor 0 step if field has evaluated to a non-zero value. The dialog text contains the expression that evaluated non-zero to help you decide if a cursor adjustment can fix the problem. The dialog default button is set to Reject.

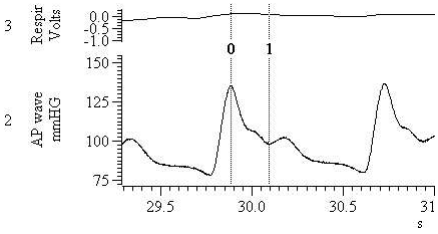
Example plot

As an example of a trend plot, consider this data file containing an arterial blood pressure on channel 2 and a respiration signal on channel 3. Let us suppose that you are interested in the position of the dicrotic notch (the small downward blip after the peak of the blood pressure) relative to the peak of the blood pressure.

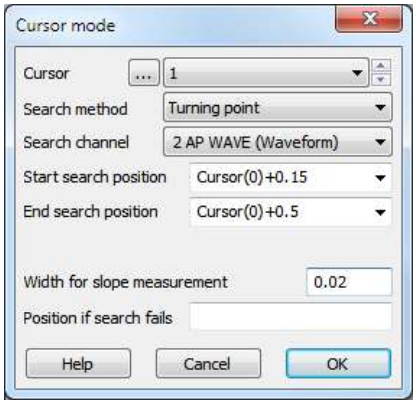




The first step is to decide how to step through the data. The obvious method is to locate the peaks in the blood pressure signal. I used the Cursor menu Active modes… command to select cursor 0, set the search method to Peak find, the search channel to 2 and Amplitude to 30. Any Amplitude from 5 to 40 would work as each cycle is at least 40 mm Hg from peak to trough and the biggest wobble in between is around 5 mm Hg. I set the Minimum step to 0.5 and the Maximum width for peak to 1 to reject artefacts (although this data does not have any). I checked that this was working with `Ctrl+Shift+right` and cursor 0 stepped from peak to peak.



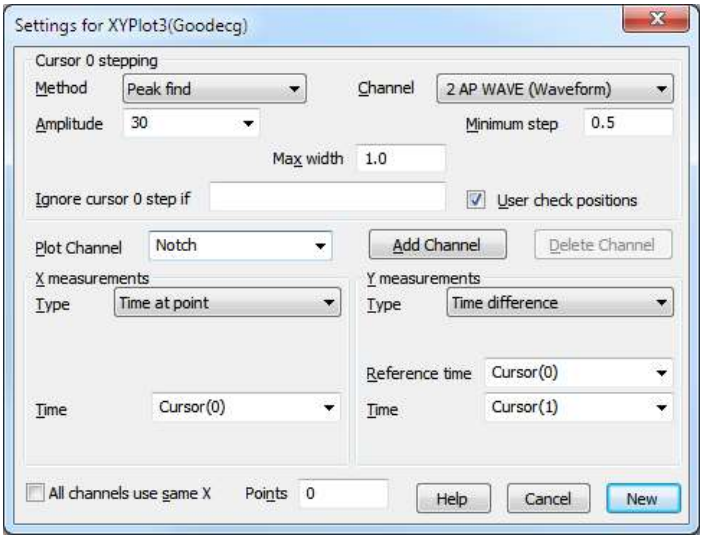
The next step is to locate the notch. I added cursor 1 to the window with `Ctrl+1`, and then opened the cursor Active modes dialog for cursor 1. This time I set the method to Turning point, set channel 2, set the search range to start at `Cursor(0)+0.15` and end at `Cursor(0)+0.5` and set the Width for slope measurement to `0.02` seconds.



I used Turning point mode and not Trough because the amplitude of the peak after the notch may be very small. I started the search just after the peak because the peak is also a turning point (where the slope changes sign) and I wanted to exclude it. From a visual inspection of the data I could see that the notch was always more that 150 ms past the peak, so by starting the search at `Cursor(0)+0.15` I avoided the possibility of detecting the change of slope between the peak and the notch.

The end of the search is set so that any reasonable notch will be found. The hardest item to set is the Width for slope measurement. This needs to be big enough so that noise in the signal doesn't cause false turning points, but small enough so we don't miss a small one. I checked it was working with `Ctrl+Shift+right` and now both cursors stepped along the data. Position if search fails is left blank so that if the turning point cannot be found, no measurement is taken.

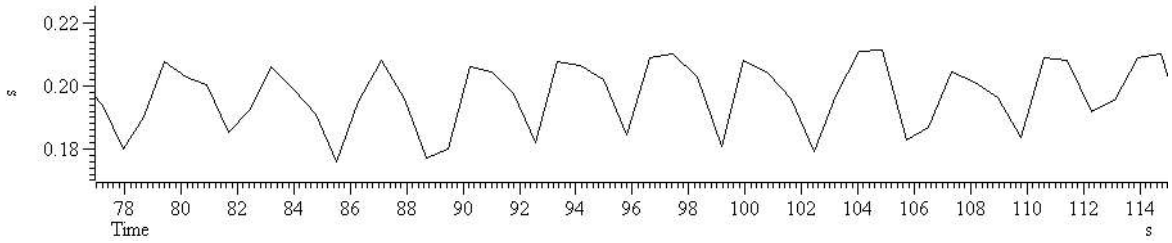
The final step is to generate the graph. I used the Analysis menu Measurements command and selected XY View. When the dialog opens it automatically picks up any active cursor setting from cursor 0.



I wanted to plot the notch position relative to the peak against the position of the peak. To do this I set the X measurement to be Time at point and selected Cursor(0) to be the point. I set the Y measurement to Time difference and set the Reference time to Cursor(0) and Time to Cursor(1) to form Cursor(1)-Cursor(0) as my measurement.

All that remained was to click New to create the XY window and open a Process dialog. I adjusted the time window so that it displayed the data I wanted to process, then set the start and end times to `XLow()` and

`XHigh()`, selected Process all data and clicked the Process button. The picture below shows the result. For each heart beat in the time range, we have a plot of the time delay from the peak to the dicrotic notch.



Of course, we could have made a wide variety of measurements. The x axis need not be time, it could be a value derived from a different channel. For example, looking at the results, we can see that there seems to be a link between the notch position and the respiration signal, so we might have set the x axis to the value of the respiration channel at the cursor 0 time.

Measurement fields

The X measurements and Y measurements areas determine the x and y values passed as a pair into each XY channel. The Type field sets the measurement method. The contents of the measurement area depend on the measurement type. The following fields are used:

Channel
The data channel used for a measurement. Select the data channel from the list.

Time
This is a time for use either as a value, or the time at which a measurement of the data channel is to be made. This field will usually be set to an expression that contains a cursor position, for example `Cursor(0)-1.3` or `(Cursor(0)+Cursor(1)/2)`. You can select expressions from the drop down list.

Expression
This is used for Expression measurement mode only and is the expression to be used for a measurement. This will usually be a view expression that evaluates to a time.

Prompt
This is used in User entered value measurement mode only, and is the prompt to use to request data values from the user.

Reference time
This field is used in the same way as the Time field. It specifies a time to subtract from the time in the Time field for the Time difference measurement, or the time at which to measure the data value to subtract for the Value difference measurement.

Width
This field is used to set the width of a point measurement around a time. If you set this to 0, the nearest data point is used, otherwise a mean value over the time range from Time-Width/2 to Time+Width/2 is used.

Start time, End time
These two fields always occur as a pair and identify a time range in which to take a measurement. Both fields will accept an expression for a time and you can select likely expressions from the drop down lists.

Minimum step
Cursor 0 does not use the Start time and End time fields. Instead, you set the minimum step from the last cursor 0 position. The search range for cursor 0 ends at the end of the file (or the start if you are searching backwards) and starts at the minimum step away from the last cursor 0 position. If you set a minimum step, you will not be able to find items within the minimum step of the start time set to process. For example, if you set a

minimum step of 1 second, a Peak search would not find a peak in the first second of a file. From [10.11, 9.14], if you set a negative start time to process, the very first search runs as if you had set a minimum step of 0 seconds.

Measurement types

In the description of each measurement type we refer to waveforms and events. By waveform, we mean either a waveform channel, or a WaveMark channel drawn as a waveform. The measurements are:

Value at point
This is the value of the nominated Channel at the specified Time. If Width is non-zero, the measurement is the mean value over the time range from Time-Width/2 to Time+Width/2. For a waveform or a channel drawn as a rate, mean frequency or instantaneous frequency, the result is in y axis units. For all other event drawing modes, the result is the time of the first event at or after Time.

Note that for a waveform drawn as a cubic spline with Width set to 0, the value is the splined value at the point.

Value difference
This is the value of the nominated Channel at Time minus the value at Reference time. If Width is non-zero, the measurement is the mean value from -Width/2 to Width/2 around each time point. For a waveform or a channel drawn as a rate, mean frequency or instantaneous frequency, the result is in y axis units. For all other event drawing modes, the result is the time difference between the first event at or after Time and the Reference time.

Value above baseline
This is the same as Value difference, except that the Width measurement is applied only to the reference time value. The expectation is that the reference time is set in the middle of a baseline region, and value is a spot value at Time.

Value ratio
This is identical to the Value difference measurement except that instead of subtracting the values, we divide the value of the nominated Channel at Time by the value at the Reference time. Attempted division by zero does not produce a measurement.

Value product
This is identical to the Value difference measurement except that instead of subtracting the values, we multiply the value of the nominated Channel at Time by the value at the Reference time.

Time at point
This is the value of the expression in the Time field. This is often a cursor position, for example: `Cursor(0)`.

Time difference
This is the value of the expression in the Time field minus the value of the expression in the Reference time field. In many cases, both these values will be cursor positions.

Fit coefficients
A fit is requested on the channel set by the Channel field after each cursor iteration and the value of the coefficient selected by the Coefficient field is the measurement. For this to be useful, you must define a fit for the channel using iterating cursors for the start and end times. If a valid fit exists for the channel you can choose the coefficient by name, for example: a0: Amplitude, a1: Time constant, a2: Offset, otherwise you select the coefficient as a0, a1, a2 and so on.

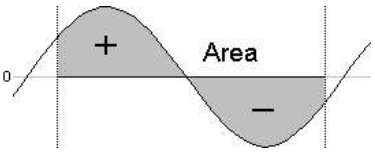
Expression
This in a dialog expression (usually involving cursor positions) that is evaluated at each step to produce the value.

Use entered value

You will be prompted to enter a value at each step of the cursor 0 position. If you enter a non-blank prompt, this is used when you are asked for a data value. The prompt field re-uses the space normally used for the Time measurement. If you use this online (during sampling), all other measurements are disabled while the prompt dialog is displayed. In particular, if the cursor 0 position is likely to change very frequently, you will find it difficult to do anything else except service the prompt to enter a value.

Area

If the Channel is drawn as a waveform (has a y axis), the result of this measurement is the area between the waveform channel and the y axis zero over the time range set by Start time and End time. The area is positive for curve sections above zero and negative for sections below zero. For all other cases this is the count of events in the time range.



Mean, Mean of Abs

For channels drawn as waveforms, the mean is the average value of the data points between Start time and End time. For all other channel types the value is the number of events in the time range divided by the time range. This could be thought of as the mean event rate in the time range. Mean Abs differs from the mean when the data values could be negative and is the mean ignoring the sign of the data. For a waveform this is equivalent to the mean of the rectified data.

Slope

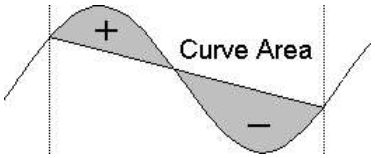
This has meaning only for a channel with a y axis. The result is the slope of the best-fit straight line by the least squares method to the data between Start time and End time.

Sum

For a channel drawn as a waveform (with a y axis) this is the sum of the values of the data points. For all other cases it is the same as the Area measurement.

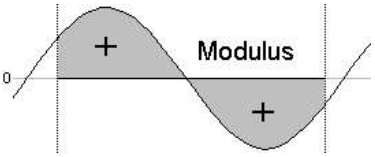
Curve area

This measurement only has meaning for channels that are displayed with a y axis, such as waveforms or mean frequencies. It is the area between the straight line joining the ends of the waveform data in the time range and the data points. Curve areas above the line add to the result; areas below the line subtract from the result.



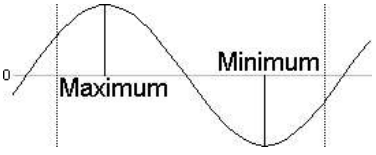
Modulus

This has meaning only for channels drawn as waveforms (with a y axis). It is the same as the Area measurement except that all areas (above and below the zero line) count as positive. For channels drawn as events there is no measurement.



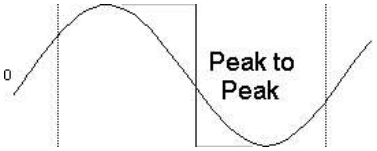
Maximum, Minimum, Abs Max

These have meaning only for a channel drawn as a waveform. They measure the maximum and minimum values in the time range and Abs Max measures the larger of the maximum and minimum value ignoring the sign. For example if the maximum value was 6 and the minimum value was –7, the extreme value would be 7.



Peak to Peak

This measurement has a value for channels drawn as a waveform (with a y axis). It is the difference between the maximum and minimum value of the channel in the time range Start time to End time.



RMS amp

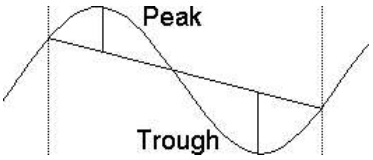
This measurement has a value for a channel drawn as a waveform. It is calculated by summing the square of each data point, dividing the sum by the number of data points and then taking the square root of the result.

SD and RMS Error

The standard deviation (SD) has a value only for a channel drawn as a waveform. It is calculated by finding the mean of the data, then summing the squares of the differences between each data point and the mean, dividing the sum by the number of data points minus 1, and finally taking the square root of the result. The RMS Error is the same, but the division is by the number of data points.

Peak and Trough

These measurements only have meaning for waveforms. These values are the maximum positive and negative distances between the waveform and a straight line joining the end points of the waveform in the time range Start time to End time. The Peak value is always greater than or equal to 0. The Trough value is always less than or equal to zero.

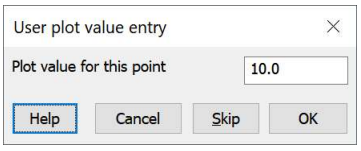


Mean in X, SD in X

These value are only meaningful when all the data is positive. If the data values represented mass, the Mean in X would be the centre of mass and SD in X represents the spread of the mass around the mean. See the Cursor Regions measurements for more.

User plot value dialog

This dialog opens when you have selected a User entered value item in one of the Measurement dialogs. For each iteration of cursor 0 you are prompted to type in the associated data value. You might use this when there is a condition that is difficult to assess automatically, but which is quantifiable by eye. The options available are:



OK	Accept the current typed in value.
Skip	No value is available, but keep going.
Cancel	End the measurement process.
Help	Display this information.

The value typed in is a floating point number, or can be a time or day as `hh:mm:ss.frac` (which will be converted to seconds past midnight).

Online use

We suggest that you do not use this feature online (when sampling), though it will work. In order to prevent recursive calls into the data process system using this feature stops any other process running online.

Measure Now (Ctrl+M or Enter)

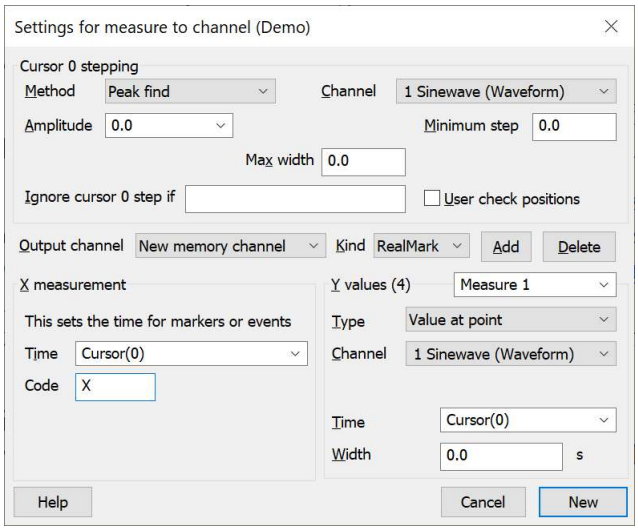
Form Spike2 version [10.17] you can generate a measurement interactively using the current cursor positions with the Analysis menu Measurements->Measure Now command or the short-cut `Ctrl+M` or the `Enter` key To use this command there must be a Measure to XY process set up and the current view must be the Time view from which the XY view was created.

Each time the menu command is used, or the `Ctrl+M` or `Enter` key is pressed, the system will attempt to take the measurement(s) set for the target XY view.

You can use this feature when the measurement conditions cannot be created with the active cursor system, or the measurement positions can only be set by hand.

This feature is experimental and may be modified. The Script language equivalent of this is `Process()` with no arguments.

Measurements to a data channel



Use the Measurements menu item and select Data channel to open the dialog. You can also activate this dialog by right clicking on a channel that was created by a measurement and selecting Process Settings from the channel pop-up menu. The dialog generates a new event, marker or RealMark data channel in the current time view.

This command is similar to the Measure to XY view command with All channels use same x, but with fewer choices for the x (time) measurement. The new channel holds the results of measurements made with the active cursors.

The basic idea is that cursor 0 steps through the data following a user-defined rule. This can be as simple as *move to the last cursor 0 position plus 1 second* or it can be a complex data-searching algorithm. Each cursor 0 move also triggers the other active cursors to search. For each cursor 0 position, we generate one Event, Marker or RealMark data point. The time of this data point is usually related to a cursor position. If the output channel is a RealMark, one or more measurements can be attached to the measurement. Using this mechanism, a wide variety of measurements can be made and displayed.

You can use this command on-line as well as off-line. For example, if you were recording a blood pressure or ECG signal, you could use this to pick signal peaks to provide a real-time heart rate channel, directly from the blood pressure signal.

The script language equivalent of this dialog is `MeasureToChan()`.

Cursor 0 stepping

The fields in this dialog area are identical to those in the Measurements to XY views. They determine the cursor 0 iteration to use when processing data. Changes made to the cursor 0 stepping in this dialog do not change the cursor 0 active mode.

Output channel

When you are creating a new channel this field holds a drop-down list of possible channels. You can choose an unused disk channels or New memory channel. If you open the dialog after creating a channel, the field holds the channel number. You cannot replace an existing channel from this dialog; you must delete the existing channel first.

Keyboard channel output (online only)

From Spike2 [10.10] you can also select the Keyboard channel as a destination when the current time view is being prepared for sampling. If any new values are detected when the channel process runs during data acquisition, a keyboard marker is written at the current time with the marker code set by the *Code* field. This allows you to trigger writing to disk, or the output sequencer or arbitrary waveform output without needing to write a script. If you use this feature you will likely want to set the Update window every field to 0.0 and the Leeway for processing field to 0 (or as close to it as your iteration method permits) in Automatic mode in the online Process dialog. Also, you will like want to open the Edit menu Preferences Scheduler to set the *Minimum gap between update routine end and next start* to a small value (for example 5 ms) to run the idle routine as often as possible.

Note that there are no timing guarantees for Keyboard markers written by this mechanism. If Spike2 is busy doing other things there may be no idle time available and there can be long delays between the cursor 0 step location and sending the key to the keyboard channel. On a machine that is not highly loaded with sampling tasks, delays of 5-20 milliseconds are common, but longer delays can occur. If there are other actions or scripts running, delays can be very long.

Kind
This sets the type of the output channel to create, one of: RealMark, Marker or Event. If you change the channel type after processing this will delete all the channel data when you click Change. For RealMark data, the target channel has one or more (up to 32) real values attached to each marker. Values are set by the contents of the Y measurements area of the dialog. If you select the Keyboard channel online the channel kind is set to Marker and cannot be changed.

Code
From Spike2 [10.10] this field appears whenever the output channel is a Marker or RealMark channel. It allows you to set the single character or a 2 character hexadecimal marker code to apply to each measurement. This is mainly for use with the new ability to send data to the keyboard channel during sampling. The script equivalent of this field is the `MeasureX()` command `expr2` argument. If you want to have a choice of Marker codes (for example, depending on some measurement), you cannot use this mechanism. You must either control sampling with a script or use the new (from [10.09]) feature to trigger a script on an event.

Add
This button is enabled for RealMark output channels when fewer than 32 measurements are defined. It adds a new measurement to the list of existing measurements. The new measurement starts as a copy of currently selected measurement. Use the Y value (n) field to give the new measurement a title. The n is the number of measurements set for the RealMark channel.

When there are multiple measurements per output item, all must succeed to write an output value. This differs from measurements to an XY view with a common x value where failed measures are omitted from their target channel, but other channels at the same x-coordinate are saved. Measurements can fail if they are inappropriate for the channel, or if they depend on a channel that no longer exists.

Delete
This button is enabled for RealMark output channels with more than 1 measurement. It deletes the currently selected measurement.

X measurement (time)
The Time field in this dialog section sets the time of each item that is added into the data channel. You will normally type in an expression that is related to active cursor positions. The most common choice will be `"Cursor(0)"`. When writing to a disk-based channel, each iteration must increase the Time measurement. This field is disabled and ignored when you select the keyboard marker as the target channel (as all keyboard markers are written at the current time).

Y values / Channel title
In the case of RealMark measurements, the Y value field is a drop-down list of editable measurement names to select the measurement to edit. For RealMark data, the channel units are copied from the Channel used for the Y measurement.

For Event and Marker output, the field holds the output channel title.

The fields in this dialog area are identical to those in the Measurements to XY views. You can use this section of the dialog when the type of the target channel is RealMark otherwise the fields in this section (apart from the channel title) are disabled. The value(s) extracted here set the RealMark data.

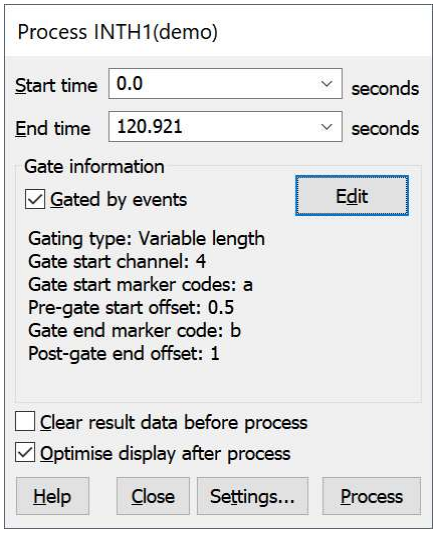
New/Change
This button is labelled New when you are creating a channel and Change if you return to this dialog after creating the channel. If you change the channel type and click Change, any previous data stored in the channel is deleted. You are asked to confirm that this is what you intended to do.

When you click the New or Change button, the Process dialog opens for you to choose the time range to process to generate data for the new channel. If you are sampling data into the time view, the on-line version of the Process dialog opens.

Process settings

This command opens the analysis set up dialog of the current result or XY window. The window must have been created by the Analysis menu New Result view or Measurements command. It is the same as the dialog that created the window except the New button is now a Change button. The Change button is enabled if you change any field in the dialog and accepts changed settings and clears the result.

Process



This command opens the Process dialog when the active window is a result or XY view attached to a time view. The dialog is also used to process measurements to a channel in a time view. It prompts you to select a time range of the data document to process. You can choose to add to the results of previous analyses, or you can clear the output. You can also choose to optimise the y axis of the result after the analysis is complete. The Settings button takes you back to the Settings dialog.

Start time and End time

These fields set the time range to process. You can type in time ranges or select suitable times as dialog expressions from the drop-down lists (click on the down arrows at the right-hand end of the fields).

For triggered analyses, for example Waveform averages, peri-stimulus histograms and event correlations, the time range sets the trigger points to use. For non-triggered analysis, for example power spectra and measurements, the time range sets the data to analyse.

If a triggered analysis has Manual as the trigger channel, then the Start time is used as the trigger for a single sweep. In the special case of a Phase histogram with a Manual trigger, both start and end times are used to mark the beginning and end of a single cycle.

If there is more than one time window associated with the data document (when it has been duplicated), the drop down lists include the duplicate window number. For example, View(2).Cursor(1), meaning the position of cursor 1 in the second duplicate time window.

For Measurements to data channel and to XY view processes, you can set a negative start time. This allows the first search to find data within the Minimum step time of the start of the file.

Gated analysis

You will normally want to Process all data in the time range. However, you can also chose to process only those sections of the time window that lie within a user-defined time of particular events in the time range set by the Start time and End time fields. For example, you might use events to mark a specific treatment type. You could then analyse data that fell within a time range of the treatment. To do this, check Gated by events to enable gated analysis mode. The Gating information area contains a synopsis of the current gating mode. You can modify the gating by clicking the Edit button to open the Gate Settings dialog.

Processing

When you click Process, Spike2 evaluates the start and end fields and processes the data selected by the dialog. The dialog remains on screen until removed with Close. This allows you to accumulate the results of processing different areas. When processing to a time view channel that is not a memory channel, you can append data at the end of the channel only; you cannot add new data before the end of the channel.

Check the Clear xxxx before process box to clear all result data before the results of processing are added. The xxxx depends on the type of processing.

Check the Optimise Y axis after process box to rescale the results to display all values. If you are processing to an XY View you may wish to disable this option and use the XY Autoscale option if you want more control over the display scaling. If you enable both, the Process dialog option wins, but the system will waste time calculating the Autoscale values.

Breaking out of Process

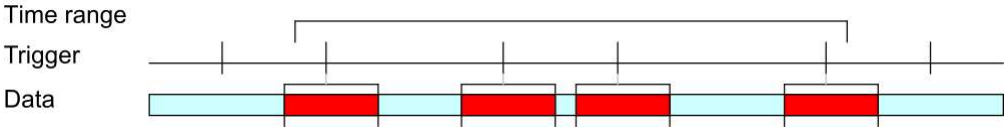
Processing operations can take quite a time, especially in large data documents. If Spike2 detects a lengthy process operation, it displays a progress dialog in which you can cancel the operation. You can also stop processing early with the Esc key.

Time range processing

When a process gate is not defined, processing of data takes a source time range and uses this to determine the range of data to be analysed to generate the result. How this is done depends of the type of the data analysis: Triggered or non-triggered.

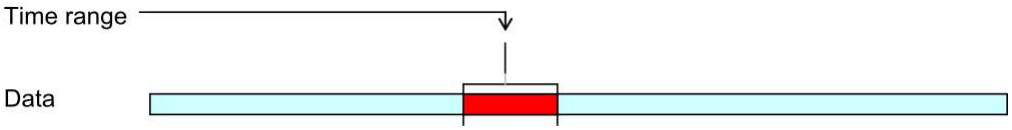
Triggered analysis

The triggered analyses are: PSTH, Waveform Average, Waveform and Event correlations and the Phase histogram. In these cases, there is a channel designated as the Trigger that is used to align data for accumulation.



The time range defines the trigger events to be included in the analysis. In the example above, there are 4 trigger events that fall in the time range, so in this case, 4 sweeps of data are accumulated.

There is a special case of triggered analysis where the trigger channel is set to Manual:



In this case, the start time of the process is taken as the trigger time, so only a single sweep is accumulated. In the special case of a Phase Histogram in Manual mode, the start and end times define a single sweep.

Non-triggered analysis

The non-triggered analysis types are: INTH, Frequency histogram, Power spectrum and Waveform correlation. In these cases, the time range sets the range of data to be processed.

Gated processing

When a process gate is defined, processing of data takes a source time range and uses this to determine the range of data in a gate channel(s) to be searched for suitable gate events. In fixed-length mode, each gate event defines a time range to process. In variable-length mode, one gate event sets the start of a time range and a second gate event (which can be on the same channel) determines the end of the time range to process.

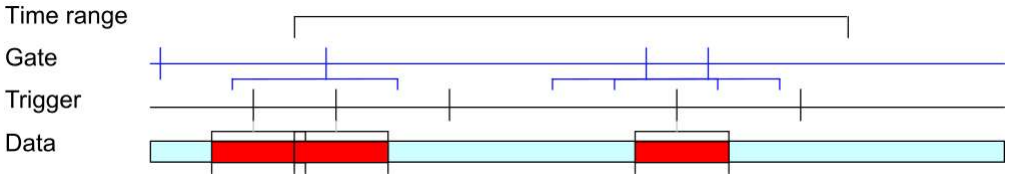
The diagrams below show the situation for fixed-length gate mode where each gate event selects a fixed time range. The time range starts a preset time before the gate event and runs for a fixed time. If two gate time ranges overlap, the time range is amalgamated into a single time range.

In the case of variable length gate mode there are two gate specifications. The first sets the channel and any marker filter to detect the gate event that defines the start of the time range to process. The second sets the channel and any marker filter to detect the gate event that defines the end of the time range to process. The search for the event that defines the end of the range starts after the event that defines the start of the range (so if you use gate events on the same channel you need two events to define a time range). The search for the next start gate for a range starts after the event that ended the previous range.

How this is done depends of the type of the data analysis: Triggered or non-triggered.

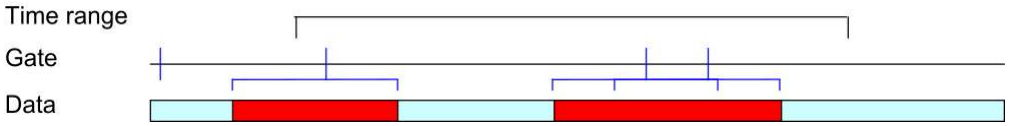
Triggered gated analysis

The triggered analyses are: PSTH, Waveform Average, Waveform and Event correlations and the Phase histogram. In these cases, there is a channel designated as the Trigger that is used to align data for accumulation. The initial time range determines where we search for gates. In this example, in fixed length gate mode, the first gate selects two triggers. The next two gate conditions generate an overlapped time range, which is amalgamated into a single range. If they were not amalgamated, any triggers that happened to fall into both the gate ranges would be counted twice.



Non-triggered gated analysis

The non-triggered analysis types are: INTH, Frequency histogram, Power spectrum and Waveform correlation. In these cases, the time range determines where we search for data, exactly as for the triggered case. However, we now process the data range that the gates generate:



Gated processing during sampling

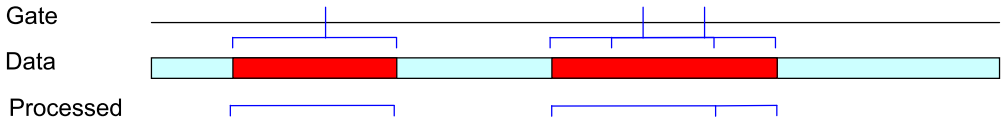
When you are sampling data in gated mode Spike2 holds off processing until the current time has reached the end of the gate period. For example, in fixed length gating mode, if you set a pre-trigger time of 0.5 seconds and a gate duration of 2 seconds, processing will be held off until 1.5 seconds after the trigger event. In variable length mode, processing is held off until the stop event time plus the post-trigger offset.

Overlapped gates

To match the off-line behaviour, if gate periods overlap, the processing period is extended. However, processing occurs at the end times of each gate (otherwise you would never see any result if new, overlapping gates were continuously added). The first process in an overlapped sequence is as normal, but the subsequent ones start at the end time of the previous process and run to the end time of each gate period.

This works perfectly in triggered process modes where the gate period selects the stimuli to be used and it does not make any difference how the data to be processed is broken up.

In continuous process modes this can make a difference. For example, in Power spectrum mode, the gate period determines the length of data to be analysed. There must be at least enough data to match the Fourier Transform data block size set for the analysis. If the overlapping gates are too close together, each subsequent period may be too small to process, resulting in no data from them.



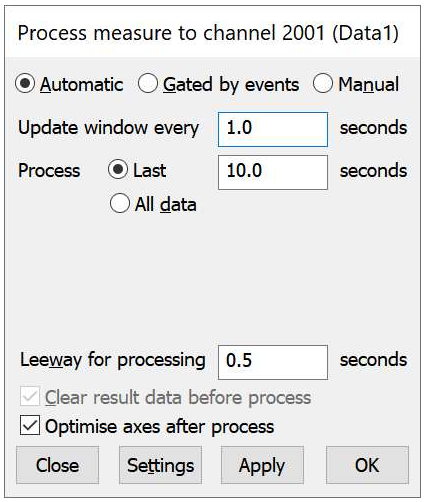
In this example, using a fixed period gate, there are three gate events. This causes three process actions, each of which happens at the end time of the gate caused by each trigger. The first process is of the gate length, the second is also of the gate length and the third is for the extra time period. If you has set the Clear Result data before process check box, the result would be cleared before each non-overlapped set of gates.

Process command with a new file

When the current window is a result or XY window attached to a sampling data file, the Process command activates a modified version of the process dialog. This dialog also appears when you create a result or XY window while sampling. It is also used to control processing measurements to a channel during sampling.

This dialog gives you control over when and how the result is updated. You can select Automatic, Gated by events or Manual updates. The dialog contents depend on the update mode. The two check boxes operate in the same way as in the Process dialog, described above. The Settings button takes you to the Process Settings dialog, Close removes the dialog. Apply and OK both apply the settings; OK closes the dialog.

Automatic mode



In Automatic mode, the result updates as close to a user-defined interval as possible. Set the interval to 0 for the most frequent updates. Updates occur when the system has idle time.

The Clear result data before process check box is gray as each of the two Process modes has only one useful setting. The modes are:

Last — Re-calculates the result for the time period set using the most recent data. Use this mode to follow changes. Clear before process is always checked.

All — The result of processing new data is added to the analysis. Clear before process is always unchecked. If it were checked, each update would show only the new data since the previous update.

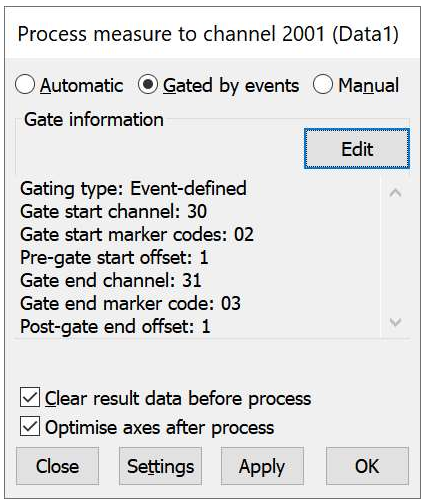
Leeway for processing is visible when processing with active cursors to an XY or time view channel. It sets the time to allow after cursor 0 to wait before processing to allow for other cursors and measurements. If you set this to 2 seconds, the cursor 0 iteration is limited to 2 seconds back from the current time, allowing 2 seconds for other cursors and to take measurements. If your analysis does not generate any data, check that this value is large enough.

Measure to Keyboard channel

If you use the Measure to Keyboard channel feature (version [10.10] onwards), each time the process runs it generates either one or no keyboard markers. Markers get the current sampling time, not the measurement feature detection time. Because of this, you will almost certainly want to set *Automatic* mode, *All data* and *Update window every* 0 seconds and 0 seconds *Leeway for processing* to minimize the delay between detection of the process event and output of the Marker code to the Keyboard channel.

It is also possible to use *Gated by events* in *Automatic* mode. *Manual* mode can be set, but will not be useful.

Gated by events mode



Gated by events mode analyses data around a specific gating event or from a time specified by one event up to a time specified by a second event.

Prior to Spike2 version 10, we supported a simple gated mode where a gate event caused a fixed period of data to be processed. From version 10 onwards we also support a more complex mode where one event starts data processing and a second event turns it off. In either case, if the event is from a marker channel, you can set a marker filter to specify which event marker codes start and stop data processing.

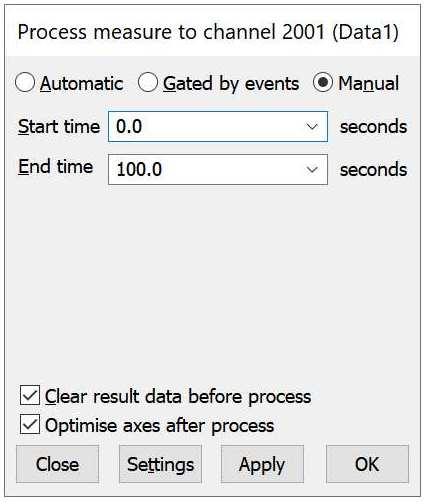
The Gate information area of the dialog displays a synopsis of the gate. Click on the Edit button to open the Gate Settings dialog to change the gate. See the documentation for the Gate Settings dialog for details of Gated analysis.

If you do not want the results to accumulate for all analysis periods, check the Clear before process box in the Process dialog. Check the

Optimise axes after process box to rescale the y axis of the result after each process so that the full range of the data is visible.

Consider the case where you are generating a waveform average in response to a stimulus and you want to see how the average differs in the presence of inhibitors and a control. You could set up separate stimulus channels for each inhibitor and the control, but it is often easier to have a single marker channel and use differing marker codes to mark the start (and end) of the regions for a particular treatment. This is generally the case if you use the Output Sequencer to control the presentation of stimuli.

Manual mode



Manual mode is a very simple-minded analysis mode which processes data between two times. It is similar to using the Process dialog in offline mode but without the possibility of gated analysis. Most users find that they are quite busy when capturing data and it is usually easier to choose one of the other, more automated, forms of data analysis.

If you select Manual updates you must provide a start and end time for analysis. If the End time you set has already been reached, the data will be processed as soon as you click Apply (process and leave the dialog open) or OK (process and close the dialog).

However, if the End time is yet to occur, there is no processing on Apply or OK until the End time is available in the data document. At that point, the data will be processed and the target window will update. If you wish to process a different area, set a new time range and click the mouse on the Apply or OK button again.

Gate Settings

The Gate Settings dialog is accessed from the offline and online Process dialogs when gated processing is enabled. When you choose gated analysis, event or marker channels determine the start and end time of data processing. The script language equivalent of the dialog is `ProcessGate()`.

When you set Gated mode and run the Process command, the time range set in the Process command is searched for enabled gate regions, then processing is run for each enabled region. If there are N gate areas in the process time range the result is as if you ran the process command N times, once for each gate time range and accumulated the results.

Note that the gate runs from the start time up to, but not including the end time. Beware that if you use gating with Measure to XY or Measure to a data Channel, and if you iterate over data points using the same events as define the gate such that an event you wish to find occurs at exactly the start of the gate, the iteration will not find the event. This is because iteration is forwards from (but not including) the start time. You can use the Pre-gate offset field to start each gate just before the event to work around this problem.

Processing is done at the end of each gate period.

Overlapped gate periods

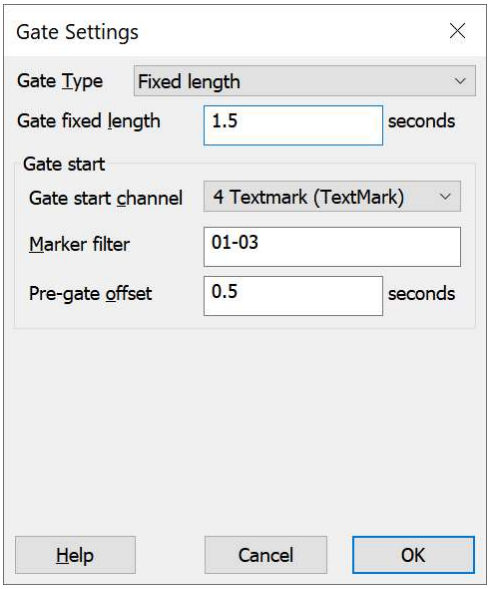
Offline, if resultant gate periods overlap, they are amalgamated to make a single longer period. If you selected Clear Result data before process, the amalgamated period counts as one process. Online, processing happens at the end times of each gate period (so you see a result, even if all the gates overlap). If gate periods overlap, the start of each subsequent overlapped period is moved to be the end of the previous period, the intention being to make the result as similar as possible to the offline case.

Gate Type

The dialog has two modes: one for compatibility with the fixed length gating supported by Spike2 before version 10, and the second to support variable length gates for version 10 onwards.

Fixed length gates

The simplest form of gating, which is also backwards compatible with versions of Spike2 before version 10 is Fixed Length. In this mode, each gate event selects a time region of fixed length that starts at a fixed time offset before the gate event.



Gate fixed length

This field is visible in Fixed length mode and sets the duration of the gate. The time period is in seconds, but you can specify a time in milliseconds or microseconds by typing: `1500ms` or as `1500000us`. The gate length must be greater than 0.

Channel and Marker Filter

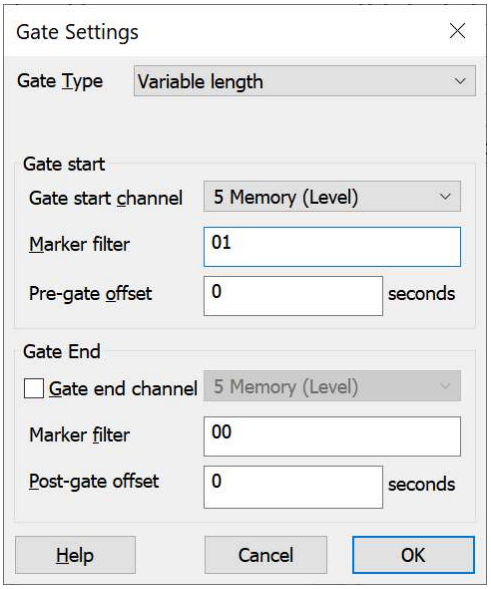
You must choose an event, marker or marker-derived channel to define the gate times. If the channel holds markers, you can specify a marker filter as text and only events in the time range that match the marker filter will be used as gates. Leave the field empty to use all markers in the time range in the current channel marker filter mask. In the example, the marker filter is set to `01-03`, meaning accepts codes `01`, `02` and `03` in the first marker code and anything for the other three codes.

Pre-gate offset

This fields defines the start of the gate before each detected gate event. You are allowed to set a negative period to start the gate after the gate event. If gate events are closer than the gate length, the gates are merged; the data is processed once only, not once per overlapped gate event. This field is set in seconds, but you can provide the values in milliseconds or microseconds by appending `ms` or `us` to the value. For example, `0.5`, `500ms` and `500000us` are all the same value.

Variable length gates [10.00]

Variable length gates are a new feature in Spike2 version 10. The gate start is set in the same way as for Fixed Length gates, but the end of the gate is determined by a second gate event, either on the same channel or optionally on a different channel.



The search for a Gate end event starts immediately after the event that started the gate. The search for the following gate start event starts immediately after the event that ended the previous gate. The gate can be expanded beyond the start and end times using the Pre-gate offset and Post-gate offset fields. You are allowed to narrow the gate by setting negative pre-gate and post-gate offset values.

Gate end channel

If you check this box, you have a free choice of event and marker-based channels for the Gate end event, otherwise the same channel is used as for the Gate start channel.

Get end Marker Filter

If the channel displayed for the Gate end channel is marker based, or is a Level channel in a 64-bit smrx data file, you can specify a marker filter as text for the Gate end event. Note that Level channels in 64-bit smrx files use code `00` for low and code `01` for high. If you set any other code (apart from leaving the field blank) the OK button is disabled and the dialog displays an explanatory error.

Post-gate offset

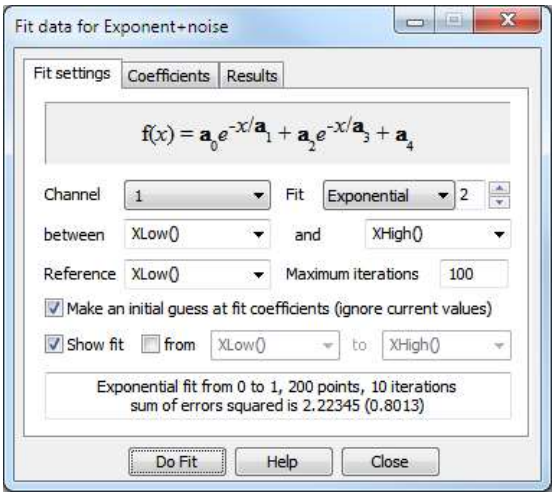
The value you set here is added to the end gate event time. Positive values expand the range of the gate, negative values make the gate period shorter.

Level channels

The example displayed above shows gating set using an event channel set to process data when the level channel is high and to not process when it is low.

In a 64-bit smrx file, Level event channels are stored as Marker channels using code 00 in the first marker code for a low event and code 01 for a high event. However, in a 32-bit smr file, the Level event channel is stored an event data and is assumed to alternate between low and high states; there is no Marker data, so you cannot use a Marker filter. If you want to process data stored in a 32-bit smr file using a level channel as the gate, use the File menu Export command to export the data into a 64-bit smrx file first.

Fit Data



This command opens a tabbed dialog from which you can fit mathematical functions to channels in a time, result or XY view. In a time view you can fit to an event-based channel as long as the data uses a display mode that has a y axis. If you fit data to a channel in a result view and error bars are displayed, the fit minimises the chi-squared value, otherwise the fit minimises the sum of squares of the errors between the data and the fitted curve. In addition to best-fit coefficients and an estimate of how much confidence to place in them, you also get an estimate of how likely it is that the model you have fitted to your data can explain the size of the chi-squared value or sum of errors squared. If your data does not have error bars, these estimates are based on the assumption that all data points have the same, normally distributed error statistics. The dialog has three tabs:

Fit settings Set the fit type and range of data to fit and range to display

Coefficients Set the starting point for your fit and optionally fix coefficients

Results Display the fitting results and residual errors

The three buttons at the bottom of the dialog are common to all pages. The Help and Close buttons do what they say. Do Fit attempts to fit with the current fit settings.

Link to Measurements

You can use curve fitting as part of the Measurements to an XY view or to a data channel. If you do this, the result is one of the fit coefficients.

Fit settings

This page of the Fit Data dialog controls the type of fit, the data to fit and what to display. The area at the bottom of the window gives a synopsis of the current fit state. Fields are:

Channel

You can select a single channel from the current view. If this is a time view, the channel must have a y axis. If you change the display mode of an event-based channel, any fit associated with the channel will most likely become invalid. You could not fit to a WaveMark channel waveform before version 8.03. In a result view, the fit is done to the basic channel data, even if the channel shows raster data.

Fit

The fit to use is defined by its name and the order of the fit (a number). For example, an exponential fit allows single exponents or double exponents. The window at the top of the dialog displays the mathematical formula for the fitting function. The following fits are currently supported (N is the maximum order allowed):

Name	N	Comments

Exponential	2	This fit includes an offset. You can force a zero offset in the coefficients page.
Polynomial	5	These fits do not require starting values for the coefficients; there is no iteration.
Gaussian	2	If you attempt to fit two overlapping peaks you may need to manually adjust the guesses for the peak centres to get convergence.
Sine	1	You can fit a single sinusoid with an offset. If the frequency guess (in radians) is not reasonably close, the fit may not converge.
Sigmoid	1	This fits a single Boltzmann sigmoid by an iterative method.

Range
You fit data over a defined x axis range, set by the between and and fields. You can choose values from the drop down list or type in simple expressions, for example `Cursor(1)+1`. There must be at least as many data points to fit as there are coefficients. For example, to fit a double exponential, which has 5 coefficients, you need at least 5 points. Most fits will use many more points than coefficients.

Reference
This is the x axis position to use as the zero value of `x` in the fitting function. The most common value for this would be the start point of the fit. However, in some cases you may want this to be elsewhere. For example, in exponential fitting, you may want to calculate the amplitude of a trace at some position. Making this position the reference point makes it easy to calculate the amplitude (it is the sum of the even-numbered coefficients). Note that the position of the reference will affect some of the coefficient values.

You will usually want to use a reference position that is within or close to the range of the x data values. This is particularly true in a Time view with a long time axis when fitting a short piece of data. In this case, setting a Reference position that is close to the fitted data will reduce loss of accuracy due to taking the difference of large numbers.

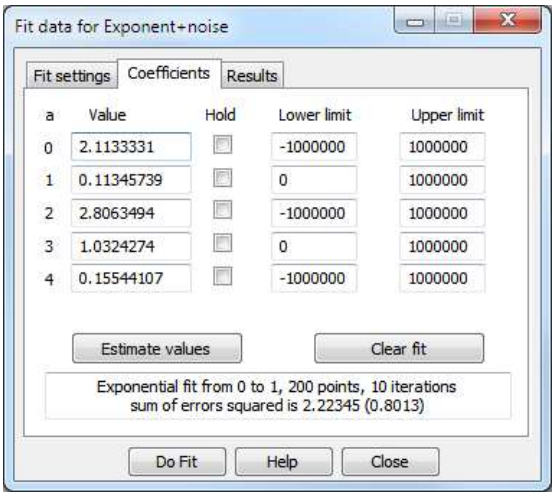
Beware
In a Result or XY view that includes 0.0 on the x axis you will likely want this to be set to 0.0 so that x axis values display as axis values. This is particularly the case for Gaussian fits where you expect the centre of the Gaussian to display as the x axis value, not as a value relative to the reference. From [10.16], this field will start at 0.0 when not fitting to a Time view unless a different value was set previously.

Maximum iterations
All fits except the polynomial are done by an iterative process. Each iteration attempts to improve the coefficient values. The iterating stops when improvements in the fit become insignificant, the iteration count is exceeded, the mathematics of the fitting process suggests that the fit is not going to improve or there is a mathematical problem. This field sets the maximum number of iterations to try before giving up.

Make an initial guess
The iterative fits need a starting point. There are built-in guessing functions that usually generate a starting point near enough to the solution that the fitting process can converge. If you check this box, these guessing functions are used each time you click the Do Fit button. Otherwise, each fit starts with the current values.

Show fit
Check this box to display the current fit. If the from box is checked, the from and to fields set the desired time range for the fit display. If this box is not checked, the fit is displayed over the fitted data range. The further away you get from the fitted data, the less reliable the fit becomes. To prevent, or at least to reduce, misleading displays, Spike2 limits the displayed fit range to a maximum of 4 times the fitted time range before and after the fitted region.

Coefficients



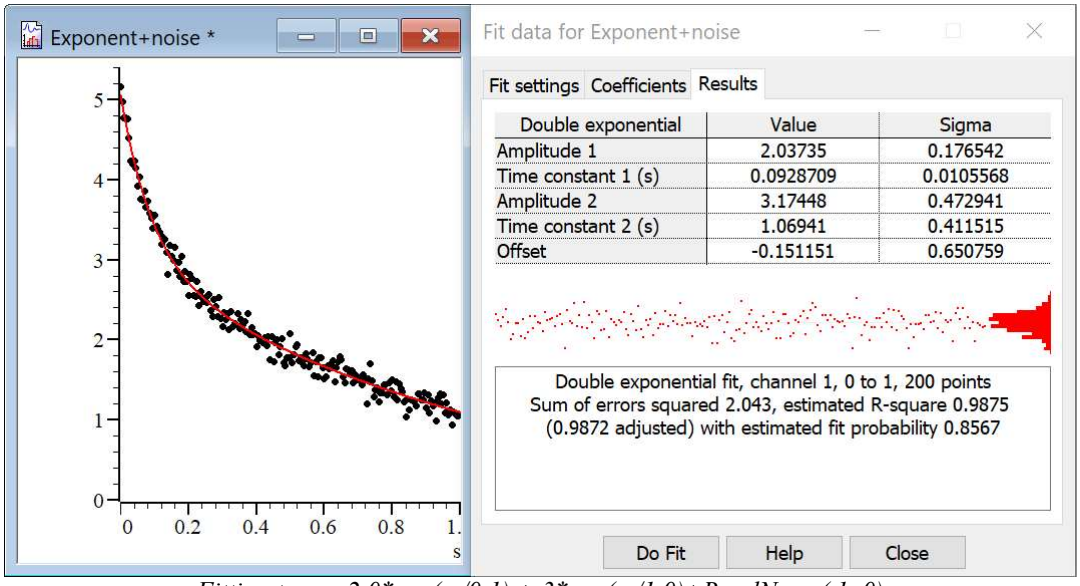
This page of the Fit Data dialog lets you set the starting values for iterative fits. You can also use this page to hold some of the coefficients to fixed values and you can set the allowed range of values for fitting.

If you know the value of one or more of the coefficients, type the value in and check the Hold box next to it. For example, in an exponential fit you may know that the final coefficient (the offset) is zero. If a coefficient refers to an x axis value, it is relative to the Reference position set on the Fit settings page.

The limit values are applied after each iteration. The fit may have to follow a convoluted path before it converges on a solution, so do not set the fit limits too close to an expected solution as this may prevent convergence.

The Estimate values button can be used to guess initial values for fitting based on the raw data. The Clear fit button removes the fit from the channel.

Results



*Fitting to y = 2.0*exp(-x/0.1) + 3*exp(-x/1.0)+RandNorm(.1, 0)*

The results page of the Fit Data dialog holds information about the last successful fit. The page has three regions: coefficient values at the top, a message area at the bottom, and a plot of the residuals (differences between the fit and the data) in the middle. The residuals are displayed immediately after a fit but will not be displayed if you close the dialog and reopen it.

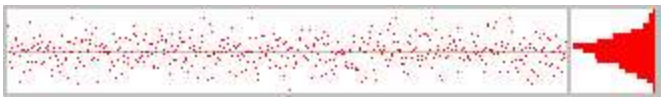
Coefficient values

The Value column holds the fitted value that minimized the chi-squared or sum of squares error for the fit. The Sigma column is an estimate of how the errors between the fitted curve and the original data translates into uncertainty in the fit coefficients given that the model fits the data and that the errors in the original data are normally distributed. If a coefficient is held, the Sigma value will be 0. The Testing the fit section gives more information on the derivation of these values and how to interpret them. You can select rows, columns or individual cells in this area, the use `Ctrl+C` to copy them to the clipboard. This also copies a bitmap image of the page to the clipboard.

If the coefficient is an x axis position, the value is relative to the Reference position set in the Fit settings page.

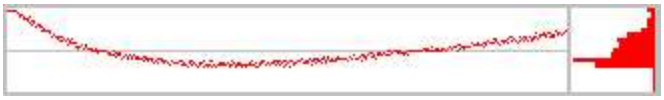
Residuals

This section of the page displays the differences between the data points and the fitted curve in the large rectangle and a histogram of the error distribution on the right. The error plot is self-scaling based on the distribution of errors; the plot extends from +3 at the top to –3 at the bottom times the RMS (root mean square) error. The line across the middle of the plot indicates an error of zero.



In the ideal case where the data can be modelled by the chosen fitting function plus normally-distributed noise, you would expect to see the residuals distributed randomly around the 0 error line and the histogram on the right should resemble a normal curve. The *fit probability* value will likely be in the range 0.1 to 0.9.

If the data cannot be modelled in this way, you would expect to see evidence of this in the residuals. In this example (generated by fitting a cubic polynomial to data that was actually a double exponential), you can see that there are clear trends in the errors. It is possible to ask how likely is this pattern of data to occur by chance (if the model was correctly chosen) and this is the *fit probability* - in this case it is most unlikely (close to 0.0000).



In extreme cases, the error due to the wrong model being used becomes much larger than the errors due to uncertainty in the data values, and you get a residual plot like this one. The *fit probability* will be 0.

Message area

This area displays a summary of the fit information that you can select with the mouse and copy to the clipboard. The first line holds the type of the fit, the channel number, the ordinate range and the number of points in this range. For example: "Double exponential fit, channel 1, 0 to 50, 50 points".

If you are fitting a result view channel that has error information displayed, the second line displays the chi-squared error value for the fit and the probability that you would get a chi-squared value of at least that size if the function fits the data and the errors are normally distributed. For example: "Chi-square value 2.7148, R-square 0.9886 (0.9883 adjusted) with fit probability 0.5867".

Otherwise, the second line holds the sum of the squares of the errors between the data and the fit and an estimate of the probability that you would get a sum of squares of errors of at least this size based on the assumptions that the errors in the original data had a normal distribution that was the same for all points. For example: "Sum of errors squared 1.871, estimated R-square 0.9886 (0.9883 adjusted) with estimated probability 0.7120.

The R-square value is the standard "goodness of fit" value, which is the proportion of the variance of the data that can be explained by the fit, so a larger value is better. If you want to compare fitting different orders of polynomials or exponents you should use the adjusted values as these discount the improvement you would expect just because you increase the number of coefficient.

If the probability value is very low or very high, there are extra lines of information warning that the fitted function plus normally-distributed noise is unlikely to model the data, or that the errors in the original data have probably been over-estimated. See Testing the fit for more information.

Context menu

If you right click on this page you are offered a context menu that contains Copy, Log and Log Titles commands. The Copy command copies selected sections of the results, or all the results if there is no selection to the clipboard as text. It also copies the page as a bitmap. The Log command prints a one-line synopsis of the current fit to the log window. The Log Titles command copies a suitable set of titles for the logged data.

Testing the fit

When you fit a model to measured data to obtain the best-fit coefficients, there are two questions you would like answered:

1. How well does this model fit the data? Put another way, how likely is it that this model plus some degree of random variation can explain my data set?
2. Given that the model does fit the data, how much confidence can I place in each of the fitted coefficient values?

When we talk about fitting curves to data, we are making the implicit assumption that you took measurements from some process that follows a model, and that this model can be expressed as a mathematical function with adjustable parameters, which are our fitting coefficients. Further, we assume that the measurements you make are not perfect; they have random variations with a known probability distribution about the correct value. To allow us to calculate likelihoods, we assume that this probability distribution is a normal (Gaussian) distribution. In the real world, or course, only some of this may apply. You may have no a priori knowledge of the distribution of errors in your original data, and this distribution may be anything but normal.

(Estimated) R-square (R^2 or R2 or r2) value

This is a simple measure of the "goodness of fit" and describes what proportion of the variance of the original data is explained by the model. One would expect values in the range 0 (meaning that the model accounts for none of the variation) to 1 (the model accounts for everything). With non-linear fits, it is possible to get a negative result, meaning that a horizontal line through the mean of the data is a better fit than the model.

We also give an adjusted R-square value. This is useful when you have a model that allows you to choose the order of the fit and you are not sure what order is appropriate. For example, when fitting polynomials you have a choice of order 1 through 5. When you increase the order, you make it easier for the model to fit random changes in the data, so the R-square value will improve as the order increases. The adjusted R-square value is less than the R-square value by an amount that quantifies how much we would expect the R-square value to increase by chance due to having more fitting coefficients.

In most cases, if a fit looks good by eye, it will have an R-square value close to 1.000, and most users will be satisfied. If you care about how well your chosen model accounts for the variability of the data you need to look at the *fit probability*.

Values are described as *estimated* if this is a least-squares fit. In this case we have no error information with the data and have to estimate it.

You can read more about R-square here.

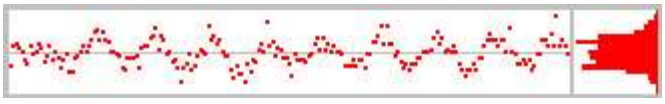
(Estimated) fit probability

This value is an attempt to say how likely the model you have chosen plus normally distributed noise is to explain the measured data. If the residuals of your data show other than a random distribution around the zero line, the probability will be 0 (or close to it). In many (if not most) cases, you will not care about this value and can ignore it. It is entirely possible to get an R-square value of 0.9998 and a fit probability of 0.0000.

Chi-square fits

In the ideal case, where you know the standard deviations of each data point, the fitting minimizes the chi-squared value, which is the sum of the squares of the differences between the model and the data points divided by the standard deviation (expected error) of the original data point values. Given a chi-squared value and the number of points it was measured from, we can calculate that probability of getting a chi-squared value at least this large, due to random variations in the data. This is the fit probability value given in the Results tab. Ideally, you would like to see a value around 0.5, meaning that you were equally likely to get a larger value as a smaller one. Values very close to 1 mean that, given the errors in each point, the data is too close to the model. Either the error estimates are too large, or the data has been "improved". Although you can hope for probabilities in the range 0.1 to 0.9, values down to 0.01 may occur for acceptable fits, and even smaller values can occur if your error distribution is not as normal as you thought.

Very low fit probabilities will occur if your data contains variations that are significant compared to the errors in the input values and that are not included in the model. For



example, if you are fitting exponents to a sampled waveform that includes perceptible mains interference, you can get a good fit (by eye) to the exponential data, but with a probability of 0.0000 as far as the mathematics is concerned because the model does not include the mains hum and cannot explain why the chi-squared value is so high.

If we assume that the model fits the data, we can estimate the standard deviation of the fitted coefficients. This is the likely variation in the fitted coefficients if we re-ran the experiment many times and fitted the data to each set of results. This is presented as the Sigma value in the results tab.

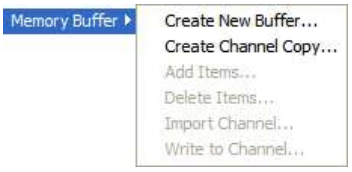
Least-square fits (and estimation of error)

If there is no error information for each point, we assume that all the points have the same, normal error distribution and the fit minimizes the sum of squares of errors between the model and the data. Because there is no independent estimate of the likely spread of the errors in the original data, strictly speaking, there is no way to give a probability of getting an error of at least this size.

However, we can say (though statisticians may shudder), *"Given that the model does fit the data, and that the errors all have the same, normal distribution, then the differences between consecutive errors should also be normally distributed with twice the variance of the errors"*. We use this to estimate the standard deviation of the data and then we apply the probability test. We label this as estimated fit probability. The same comments about likely values apply as for the Chi-square fits, except that very small values may just mean that our estimation process fails for your data.

The coefficient Sigma values are calculated on the assumption that the model fits the data, that all the original points have the same standard deviation, and that the standard deviation of the original data can be deduced from the residual sum of squares errors.

Memory buffer



Each Time view data document can have up to 2000 memory channels in a contiguous range of channel numbers (assuming sufficient memory and system resources). They hold data copied from existing channels, derived from waveform channels, entered by hand or generated by a script. You can display the memory buffers and use them like any other channels and they can be of any channel type. The memory buffers can also be written to the data document as permanent channels. If you do not write them, the memory channels are lost when you close the file; by default, Spike2 warns you about losing memory channels on close.

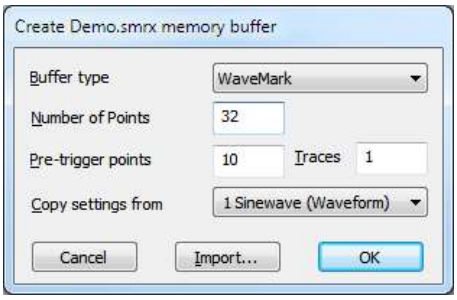
The memory buffer channels are owned by the data document. If you duplicate a Time view with memory channels, the memory channels are visible in all duplicates. If you delete a memory channel in a duplicate, the memory channel is deleted in all the duplicates.

Each memory buffer expands as items are added. The size is limited by available memory and by which build of Spike2 you have installed (32-bit or 64-bit). 32-bit applications are limited to 4 GB of memory, so you should expect that your use of memory channels to be limited to substantially less than this (even if your computer has more than 4 GB of memory). The 64-bit build of Spike2 has access to more memory than the 32-bit version.

Memory buffers are useful because the data is easily modified. This is unlike the permanent disk data channels which are expected to represent what was recorded and is not designed for easy modification (though waveform channels can be edited from the script language, usually to remove artefacts).

Script writers should avoid hard-coding memory channel numbers (for example 2001) into scripts as the channel range reserved for memory channels was different in previous versions of Spike2. You can find the physical channel number of the first memory channel with the `Chan("m1")` script command; this is available from version 8.11 onwards.

Create New Buffer



This command creates a new memory buffer channel of any type. The channel numbers are in the range 2001 to 3999 however we strongly recommend that you never use fixed values to refer to memory channels. Previous versions of Spike2 used different channels and the numbers may change again in future versions. The memory channel numbers are displayed as m1 to m2000. Almost anywhere you refer to a channel you can use m1 rather than 2001. A new channel gets the lowest available number. You can use the Analysis menu Delete Channel command to remove memory buffers. OK creates the channel, Import creates it and opens the Import channel dialog.

Channels created with this command are not permanent. The data is kept in memory and is lost when the file is closed. If you need to make the data permanent, you must write it to a permanent channel with the Write to Channel option.

The fields in the dialog change depending on the type of buffer you choose to create. The example shown is for a WaveMark channel. The fields are:

Buffer type

You can create a channel of any type: Event-, Event+, Level, Waveform, RealWave, Marker, TextMark, RealMark or WaveMark.

Number of …

This field is present for TextMark data where it sets the maximum number of characters to store with each mark, RealMark data where its sets the number of real values to store with each mark and for WaveMark data where it sets the number of waveform points to store with each mark.

Initial level

This field is present for Level event data and it sets the initial level of the channel before any events are added at Low or High. To invert an existing level event channel, create a memory channel with the desired initial state, then import events into it.

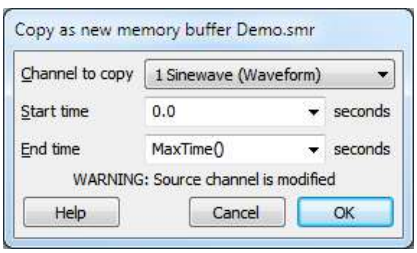
Pre-trigger points and traces

These fields are present for WaveMark data only and set the number of waveform points before the trigger and the number of traces in each WaveMark item in the range 1 to 4.

Copy settings from

When you create a waveform, RealWave or WaveMark buffer, Spike2 needs to know the waveform sample interval and calibration. This field indicates a channel that holds waveform, RealWave or WaveMark data and the buffer is given the same sampling rate and calibration. If you create a buffer from the script language you can choose a sample rate and calibration without reference to an existing channel.

Create Channel Copy



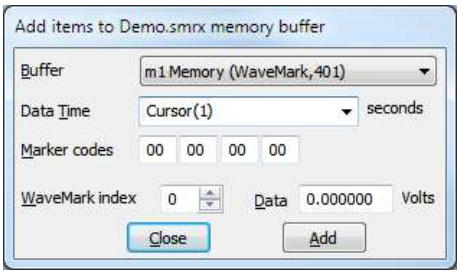
This command creates a new memory channel that is a copy of an existing channel. The source channel can be of any type. If the source channel has a channel process attached, or if it is a marker channel with an active marker filter, the new channel will contain modified data. If this is the case, a message appears in the dialog warning you. You can also activate this command by right-clicking on a channel.

Channel to copy

The source channel to copy. Copied items include the channel title, units and comment.

Start time, End time
These fields let you choose a time range of the original data to copy to the new channel.

Add items to memory buffer



This command adds a data item to a memory buffer of any type. Click Add to place or replace data in the buffer at the specified time. Waveform and RealWave data is aligned in time with existing data.

The data in the dialog is checked as you edit it and illegal input disables the Add button and displays an explanatory error message at the bottom of the dialog after a few seconds delay.

The fields depend on the channel type:

Buffer
The memory buffer channel to use.

Data time
The time at which to add data. If you have cursors enabled in the time view you can add data at the time of the cursor (as in the example). This field is present for all channels.

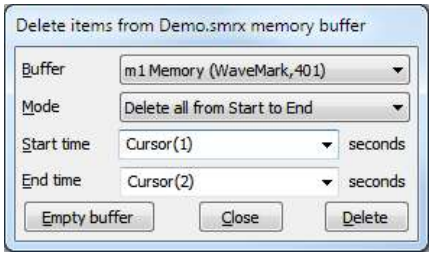
Marker
These four fields are present for all Marker derived data types. You can set the marker codes appropriate for your data here.

Data
This field is present for WaveMark and RealMark channels. Enter the index of the data point that you wish to set in the Data field. There is also a Trace field for WaveMark buffers with multiple traces.

Wave value
This field is present for waveform and RealWave channels. You can type in a number or an expression that can include horizontal cursor values (for example `HCursor(2)` or `H2`).

TextMark string
This field is present for TextMark data only. Enter the text to appear in the new data item.

Delete items from memory buffer



This command opens a dialog in which you can delete one or more data items or a time range from a memory buffer. The Delete button removes one or more items as set in the dialog. The Empty buffer button deletes all data for this channel (it does not delete the channel). Close removes the dialog. The `MemDeleteTime()` script command has the same functions.

Buffer
The memory buffer channel to use.

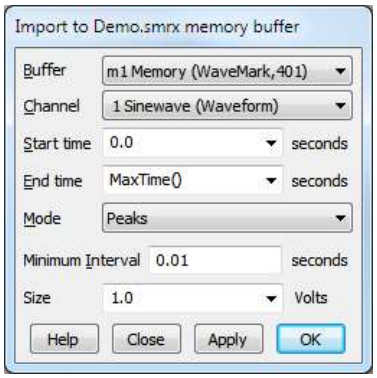
Mode
There are four modes: Delete nearest to Start within Range, Delete all round Start within range, Delete first in range Start to End, Delete all from Start to End. The first two modes delete one or more data items around a time, the other two modes delete the first or all data items in a time range.

Start time
The time range is Start time – Time range to Start time + Time range in the first two modes, and Start time to End time for the last two modes.

End time
The end of the time range. This field appears in the second two modes.

Time range
The time range around the Start time to search for data. This field appears in the first two modes. This field is usually set to a small value so that you can delete events close to the position of a cursor.

Import channel



You can import data into a memory buffer selected by the Buffer field from a channel set by the Channel field. You may not import data from a memory buffer to the same memory buffer. The Start time and End time fields set the time region to import data from.

The Mode field is present if the source is a waveform channel and the destination is not, and sets the method to extract event times from the waveform. The Minimum interval field and one of the Size/Level fields are present if Mode is present.

To import a waveform or WaveMark channel to a waveform or WaveMark channel the sampling rates must match. Apart from this restriction, you can move data from any channel to any buffer. During data import, values that cannot be extracted from the source are set to 0. For example, when importing an event channel into a WaveMark channel, there are no marker codes or waveforms, so these are set to 0. If your target channel is a Waveform or RealWave, you should also look at the capabilities of virtual channels.

The Minimum interval field sets the minimum interval between peaks, troughs or level crossings in the same direction for the data to be acceptable. There is no restriction on how close a peak can be to a trough, or a rising level crossing can be to a falling level crossing. From version [10.17] onwards, it also sets the minimum width of a gap in the input waveform to cause the search to restart (previously any gap did this).

The Size field sets the minimum peak height or trough depth. The Level field replaces the Size field when events are detected by rising and falling levels and sets the level to cross to detect an event.

When importing to a Marker or Marker-derived channel from a Level event channel, falling level transitions are given marker code 00 and rising transitions are given code 01.

The Apply button imports data and leaves the dialog open. OK imports data and closes the dialog. The Close button closes the dialog without importing. The Help button, or using the `F1` key displays this help. Illegal dialog fields disable the Apply and OK buttons and an error message indicating the reason appears at the bottom of the dialog.

Extract events from Waveform channels
You can extract events from waveform, RealWave and WaveMark channels. If the WaveMark channel has multiple traces, only the first trace is used. If the memory buffer is a Marker or is derived from a Marker, the first marker code is set to indicate the source of the data (the remaining codes are set to 0). There are six modes to extract events from waveforms:

Mode	Code	Method
Peaks	2	A peak is defined when the data rises and falls by at least the Size field value. Between each peak there is also a Trough. The peak time is estimated from the waveform around the peak using cubic spline interpolation. If the target memory buffer is a RealMark channel the first attached value is set to the peak amplitude.
Troughs	3	A trough is defined when the data falls and then rises by at least the Size field value. The trough time is estimated from the waveform around the lowest point using cubic

spline interpolation. If the target memory buffer is a RealMark channel, the first attached value is set to the trough amplitude.

Peaks and Troughs	2 & 3	Peaks and troughs are defined as above. Both the peaks and troughs are saved, but with their own codes. From [10.17] the direction of the first Peak/Trough is found by searching forward from the start time to find the first place where the data rises or falls by the set level from the value at the start time.
Data rising through level	4	The event time is set by the time where the data value rises through the nominated level. If the target memory buffer is a RealMark channel, the first attached value is set to the level.
Data falling through level	5	The event time is set by the time where the data value rises through the nominated level. If the target memory buffer is a RealMark channel, the first attached value is set to the level.
Rise and Fall through level	4 & 5	Events are generated by both rising and falling through the nominated level.

We use cubic spline interpolation to give better estimates of the positions of peaks and troughs and level crossings. If your data is noisy, this will make little difference, but can be a large improvement if the data is filtered.

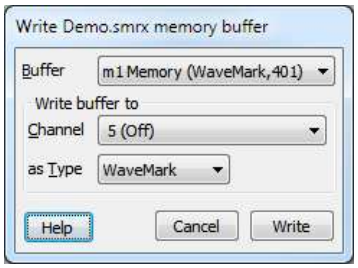
Setting the attached RealMark values and Peaks and Troughs and Rise and Fall through Level modes was added at Spike2 version 8.00.

If the target memory channel is of type RealMark, the first attached value is set to the Peak, Trough or Level associated with the event. Although saving the level is less useful than saving the peak or trough value, it allows you to use multiple level crossing detections in a single channel (for example, allowing you to re-sample a channel based on levels rather than on time intervals).

These modes use the Minimum interval field as the minimum separation of detected events, to filter out events caused by noise in the input waveform. Set this to 0.0 if you do not want to set a minimum period between detected events. The peak search mode looks for a peak followed by a fall of at least the Size field. The trough search mode looks for a minimum, followed by a rise of at least the Size field. The rising and falling level modes detect events when the signal crosses the Level in the selected direction.

To convert a waveform channel to a level event channel such that all data above a level is high and data below a level is low, create level event memory buffer with the desired initial state (low/high), then import rising edges for a level crossing, then import falling edges using the same level.

Write to channel

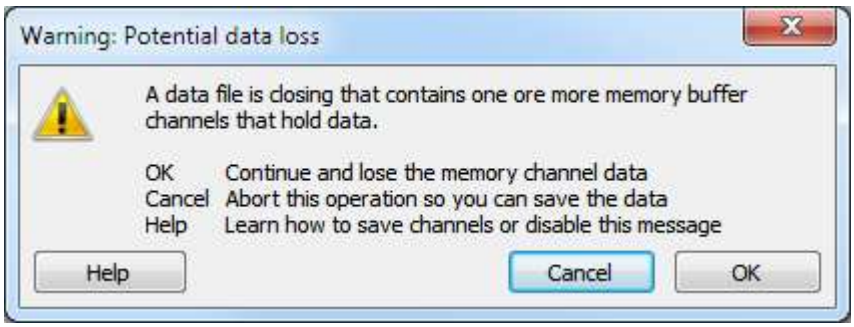


You can write a memory buffer to the data document. The Type field sets the format for the saved data. You can also Append the data to an existing channel. In Append mode, the data in the memory buffer must all occur later than data in the target channel. For modes other than Append, if you select a channel number that is already in use, you will be warned.

This command saves the entire contents of the memory buffer, regardless of any marker filter or channel process that may be attached to the channel. This is different from the Save channel command, which writes the data as displayed, including the effects of marker filters and channel processes.

Memory buffer channel data loss warning

This dialog is displayed when a data file is about to close and this would cause memory buffer data to be lost.



You can disable this warning, or limit when it occurs from the Edit menu Preferences in the General tab with the Warn if file close would lose memory channel data option.

If you get this message and want to preserve the data, click on Cancel to stop the file close process. You can then save the channels to unused channels in the data file. This can be done from the Analysis menu. There are two commands you can use:

Memory Buffer Write to Channel	This allows you to write a nominated memory buffer to an unused channel or to overwrite or append data to an exiting channel. You can also choose the type of the channel you create.
Save Channel	This is a simpler command that allows you to copy any existing channel (including memory buffer channels) to an unused channel in the file.

If you find yourself in the situation of having no available spare channels you can follow the following procedure:

1. Use the File menu Export As command to create a copy of the current file as a data file.
2. Set the file type to Data File (you can choose between 64-bit `smrx` and 32-bit `smr`). Set a suitable file name and location and click the Save button.
3. In the Export dialog, click Add (to select All Channels and the full time range in the file). Also click the Channels button and set sufficient extra channels to store your data. Clear the check in the Time shift data to that the first exported range starts at 0 seconds box unless this is the effect you want.
4. Click Export to create a copy of the original file with the memory channels saved as permanent channel.

If you want more control over the channel numbers assigned to the memory channels you can use the `ChanSave()` script command which allows you to specify exactly what you want and lets you move data between files (even files with differing time resolutions). However, you will require some expertise with the script language to use it.

Virtual Channels

Virtual channels hold RealWave data derived by a user-supplied expression from waveform, event and RealWave channels and built-in function generators. No data is stored; the channels are calculated each time you use them. You can match the sample interval and data alignment to an existing channel, or type in your own settings. Channel sample intervals and alignments are matched by cubic splining the source waveforms, linear or cubic interpolation of RealMark data and by smoothing event rates. The script language equivalent of this command is `VirtualChan()`. You can save a virtual channel to disk (to remove the calculation time penalty) with the Analysis menu Save Channel command.

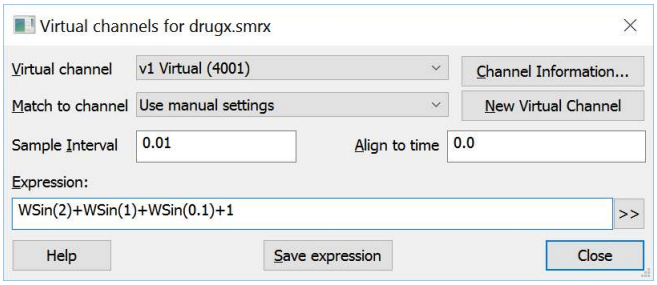
You can use virtual channels to do channel arithmetic (for example sums and differences of channels), to convert event channels into waveforms proportional to the event rate, to linearise non-linear transducers, to perform frequency analysis and to generate waveforms. This is a very powerful feature with a huge range of applications.

Because the data is calculated every time it is required, using virtual channels can be slow. If this is a problem, you can consider saving the virtual channel to a permanent channel using the Analysis menu Save channel command. You can then delete the virtual channel and use the saved channel, which will be as quick to use as any other RealWave data channel.

Virtual channels occupy a contiguous range of channel numbers. From Spike2 version 9 onwards, you can have up to 2000 virtual channels and the first virtual channel is 4001. However, previous Spike2 versions allowed fewer channels and the base channel number was different. In a dialog you can use v1, v2 and so on to refer to virtual channels. In a script you can use `Chan("v1")` to find the first virtual channel number; this is implemented from version 8.11 onwards. The actual maximum number of virtual channels is likely to be limited by memory and system resources.

Create New Channel, Edit Channel Expression

These commands are selected from the Analysis menu Virtual Channels command; they create a new virtual channel and edit the settings of existing virtual channels. Both commands open the Virtual channel dialog. The Create New Channel command makes a new, empty virtual channel and selects it before opening the dialog.



Virtual channel

Use this field to select a channel when you have more than one virtual channel.

Channel Information

This button is a short cut to the View menu Channel Information dialog; you can use that dialog to set the channel title and units.

New Virtual Channel

Click this button to create a new virtual channel.

Match to channel

You can select an existing waveform-based channel (but not a virtual channel) from which to copy the sample interval and data alignment. Alternatively, you can select Use manual settings and type in the interval and alignment yourself.

Sample Interval

This field holds the sample interval between data points in the virtual channel in seconds. You can edit the sample interval if you select Manual Settings in the Match to channel field. This field accepts expressions; for example, to set 27 Hz you can type `1/27`.

Align to time

This field sets the time of a data point in the virtual channel. The time of any point and the sample interval completely defines all the sample times for the channel. You can edit the alignment if you select Manual Settings in the Match to channel field.

Expression

This field holds an expression that defines the virtual channel. Whatever you type here is preserved and evaluated each time data is required from the channel. Expressions are composed of scalars, vectors, operators, channel functions, spectral functions, event kernel functions, waveform generation, mathematical functions and cursor expressions.

A scalar is a number, such as 4.6 or `Sqrt(2)`. A vector is a list of data values that are derived from a channel or generated by a waveform function. A function can be applied to a vector or a scalar to yield a result that is a vector or a scalar. An operator combines vector and scalars, for example 1 + 2 combines the scalars 1 and 2 to generate the scale 3. `Ch(1)+1` combines the vector `Ch(1)` with the scalar 1 to give a vector that is the waveform from channel 1 increased by 1.

Example expressions

If channels 1 to 3 hold waveforms, then `Ch(2) - 2*Ch(1)` displays the difference between channel 2 and twice channel 1.

`Sqrt(Sqr(Ch(1))+Sqr(Ch(2)+Sqr(ch(3)))` displays the square root of the sum of squares of three channels. You could use this to display the magnitude of the resultant of three perpendicular forces.

`Sqr(Ch(1))` is the same as `Ch(1)*Ch(1)`, but `Sqr()` is faster. To generate a polynomial function of the input it is much quicker to use `Poly()` than to use `Ch()`, `Squ()`, `Cub()` and so on to generate a power series.

Error messages

The expression is checked as you type it. If the expression is incorrect, an explanatory message appears to the right of Expression: above the expression entry field.

>>

Click this button to open the Build Expression drop-down list, which helps you to construct expressions interactively without needing to know the virtual channel expression syntax.

Save expression

There is a list of saved expressions in the Build Expression drop-down list. Valid expressions are added to the list when you click the Save expression button, when you change to a different virtual channel and when you close the dialog with the Close button. The expressions are stored in the system registry.

Operators

Arithmetic operators

You can use the four standard arithmetic operators plus (`+`), minus (`-`), multiply (`*`) and divide (`/`) and comparison operators together with numbers, round brackets and some mathematics and channel functions. The result of combining a vector and scalar with an operator is a vector, for example, the expression `Ch(1)+1` is a vector, being the data points of channel 1 with 1.0 added to each of them.

Dividing by a scalar value of zero is an error. Dividing by a vector holding zeros is not an error and generates special floating-point numbers for positive and negative infinities. These can cause problems in subsequent calculations (and they are difficult to display).

Comparison operators

You can also use comparison operators less than (<), less than or equal (<=), equal (=), not equal (<>), greater than or equal (>=) and greater than (>); the result of these is 1.0 if the comparison is true and 0.0 if it is false. In the expression `a op b`, where `a` and `b` are vectors and/or scalars and `op` is a comparison operator, if either `a` or `b` is a vector, the result is a vector with value 1.0 at points where the comparison is true and 0.0 where it is false. For example `Ch(1)>1` has the value `1.0` where channel 1 is greater than 1.0 and 0.0 elsewhere. `Ch(1)<>Ch(2)` is 1 wherever channels 1 and 2 are not the same. Be cautious using <> (not equals) and = (equals) as we are dealing with floating point numbers and exact equality may be compromised by arithmetic rounding.

Operator precedence

Multiply and divide have a higher precedence than all the other operators which all have the same precedence. You can use round brackets to force other evaluation orders. Apart from that, evaluation is from left to right, so `a>b*c+d` is interpreted as `(a>(b*c))+d`.

Channel functions

The following functions create a vector at the sample interval and alignment set for the virtual channel from an existing data channel (but not a virtual channel unless it has a lower channel number than the channel in use).

`Ch(n)` — `n` is a waveform, RealWave, WaveMark or a level event channel. Copy channel `n` data. Level event channels copy as 1 when the level is high and 0 when low. WaveMark channels copy the waveform from the trace set by the Marker Filter dialog.

`Ch(n, t)` — `n` is a WaveMark channel and `t` is the trace to use in the range 0-3 for tetrodes and 0-1 for stereotrodes. You can also use -1, meaning use the trace set by the Marker Filter dialog.

`If(n,g)` — `n` is an event channel to convert to a waveform by linear interpolation of the instantaneous frequency. `g` is the maximum gap to interpolate across, in seconds. Omit `g` or set it to 0 for no limit to the gap.

`Ifc(n,g)` — The same as `If()` except that cubic spline interpolation is used.

`TEvt(n, {c})` — `n` is an event or marker channel to use as a timing reference. `c`, (taken as -1 if omitted) is a marker code to match. The result is the time difference between the current time and the last matching marker code (-1 matches all codes). This matches the first marker code. If a code is set, any other marker code sets the value to 0. This generates ramps with a slope of 1 unit per second that resets to 0 at each matching code and is held at zero by non-matching codes. Added at Revision [10.09a].

`Rm(n,g,i)` — `n` is a RealMark channel to convert to a waveform by linear interpolation of the real data values, `g` is the maximum gap to interpolate over in seconds, and `i` is the real data item to interpolate (the first item is 0). Set `g` to 0 for no maximum gap. You can omit `i` or both `i` and g. Omitted values are treated as 0.

`Rmc(n,g,i)` — The same as `Rm()` except that cubic spline interpolation is used.

Since version [7.09a], you can use channel specifiers like `m1`, `v2`, `1a` in places where only a channel number was previously acceptable.

From version [10.19] you can get the value of a channel at a specific time with the `At(t, n)` command.

Use of virtual channel numbers in an expression

To prevent recursive references, you can only use a virtual channel in an expression if it refers to a lower-numbered virtual channel than the channel being defined. Although this can sometimes make things easier to visualise and allows common sub-expressions to be used by other virtual channels, this does not save you any calculation time (any may even be slower) than writing out the full expression. This is because the virtual channel is calculated whenever it is needed; it is not cached. A virtual channel may not depend on a duplicate of a virtual channel.

Spectral functions

The following functions all calculate the power spectrum of an input channel and generate a waveform illustrating how some feature of the power changes with time. These functions are particularly useful in EEG and EMG analysis. The power spectrum is calculated using the Fast Fourier Transform (FFT), applied many times to span the data. These functions can take some time to calculate. If you find that these operations are slow, consider saving the virtual channel as a real channel so that you need only suffer the slow operation once. Arguments common to all commands are:

`n` — The input waveform or RealWave channel. The maximum frequency for use in bands available to any of the commands is half the sample rate of this channel. Missing data values (gaps) in the input channel are treated as zero values.

`f` — The desired frequency resolution of the output. For a resolution of `f` Hz, the FFT used to calculate the power must span at least `1/f` seconds of the input data. So 0.1 Hz resolution implies an FFT spanning 10 or more seconds of input data. The FFT we implement uses a power of 2 data points,

which constrains the available resolutions; we round `f` down to the nearest available value. There is a trade-off between frequency and timing resolution. If you set `f` to 0, the FFT size is 256 points.

`l` The low edge of a frequency range, in Hz. If this is an optional argument and you omit it, 0 is used.

`h` The high edge of a frequency range, in Hz. if this is an optional argument and you omit it, half the sampling rate of channel `n` is used.

The available functions are:

`Pw(n,f,l,h)` Calculate the power in a frequency band from `l` to `h` Hz. There is a dialog to help you build the expression.

`Pw(n,f,l,h,r{,s})` Calculate the power in the frequency band from `l` to `h` Hz divided by the power in the band from `r` to `s` Hz. If `s` is omitted, the band extends from `r` to half the sample rate of the input channel.

`SpE(n,f,p{,l{,h}})` Calculate the frequency at which `p` percent of the power in the band from `l` to `h` Hz lies below. If `l` is omitted, 0 Hz is used, if `h` is omitted, half the sample rate of the input is used. There is a dialog to help you build the expression.

`MF(n,f{,l{,h}})` Calculate the mean frequency in the band from `l` to `h`. If `l` is omitted, 0 Hz is used, if `h` is omitted, half the sample rate of the input is used. The mean frequency is sum of products of frequency times power divided by the sum of the power. There is a dialog to help you build the expression.

`DF(n,f{,s{,l{,h}}})` Calculate the Dominant Frequency. We search the band from `l` to `h` Hz for the frequency region with the maximum power. s sets the distance in Hz to smooth the data either size of each point (0 for no smoothing). There is a dialog to help you to build the expression.

Technical details

Power is calculated using the Fast Fourier Transform (FFT) with a raised cosine window. There are three regimes used to calculate the power, these depend on the ratio of the width of the FFT (being the FFT size times the sampling interval of the input waveform) relative to the interval between the output data points:

Ratio Method

>8 We calculate the power spectrum with a single FFT at a time interval of the FFT width/8 and then interpolate the results to the output. This prevents excessive calculations when the user sets a high output sample rate. In this case, 3/4 of the data points are shared in common between two consecutive transforms, so the change between them is small.

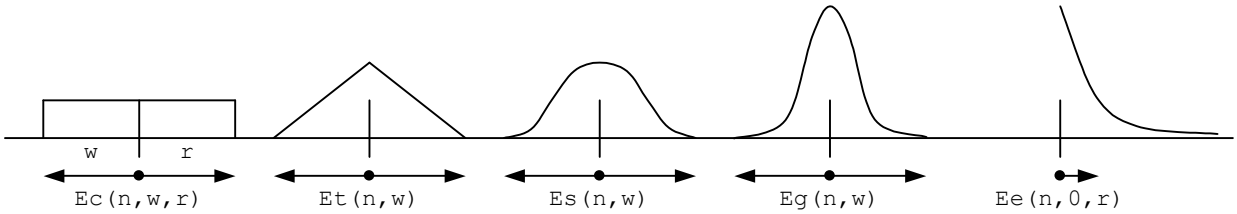
8 to 2 We calculate the power spectrum with a single FFT centred on each output point.

<2 We calculate the power spectrum with multiple FFTs. These span a space from 1/2 an output interval plus 1/4 of the FFT width before each output point to the same distance after each point and the FFTs overlap by at least 50%.

This sequence should achieve a smooth transition between different regimes as the output sampling rate changes.

Event kernel functions

The `Ec(n,w,r)`, `Et(n,w,r)`, `Es(n,w,r)`, `Eg(n,w,r)` and `Ee(n,w,r)` functions convert event channel `n` into a virtual wave and can be used anywhere in an expression that you can use the `Ch()` function. They replace each event by a kernel (shape), centred on the event time. For an event at time `t`, the kernel extends from `t-w` to `t+r` seconds except for `Ee()`, which extends from `t-8w` to `t+8r` seconds. Normally you will omit `r`, in which case the shapes are symmetrical with r set equal to `w`. The resulting waveform is the sum of the kernels for all the events. The area of each kernel is unity, so the area under the waveform between any two times is the number of events in that time interval.



`Ec(n,w{,r})` The replacement shape is a rectangle, think Event Count, running from `w` to the left to `r` to the right.

`Et(n,w{,r})` The replacement shape is a Triangle running from `w` to the left to `r` to the right.

`Es(n,w{,r})` The replacement shape is a Sinusoid running from `w` to `r` to the right.

`Eg(n,w{,r})` The replacement shape is a Gaussian with a sigma (standard deviation) of `w/4` to the left and r/4 to the right.

`Ee(n,w{,r})` The replacement is an exponential function with a time constant of `w` to the left of each event and `r` to the right. The kernel extends to 8 times the time constant in each direction.

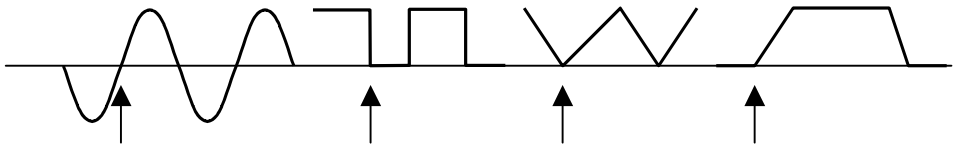
The `Ec()` function simply counts the events in the time range. This is the fastest analysis method, but produces the most jagged output. The `Et()` function weights the events with a triangle function. This is slower than `Ec()`, but much faster than `Es()` and `Eg()`. The `Es()` function weights the events with a raised cosine. The `Eg()` function weights the events with a Gaussian curve extending to 4 sigmas.

The `Ee()` function is rather different from the others as it is not suitable for use as a smoothing function and is more likely to be used in a single-sided form. It weights the events with an exponent `exp(-t/r)` to the right and `exp(-t/w)` to the left (t stands for the time difference between the point and the time). The exponent extends to 8`w` to the left and to 8`r` to the right.

If none of these conversions are suitable, you can define you own weighting functions and create a new (not virtual) channel with the `EventToWaveform()` script command.

Waveform generation

These functions generate waveforms without the need for an input channel. All these functions produce output from time 0 to the end of the file. All the arguments are in units of seconds, except the sine wave frequency, which is in Hz. All these waveforms have unit amplitude. You can use the standard maths operators `*/+` and `-` to scale and shift the output to different values.



Generated waveforms and their alignment points

If you attempt to create a cyclic waveform with a frequency above half the channel sample rate, no output will be generated. The `a` argument sets the time alignment; if omitted the value 0 is used. You can generate the following outputs:

`WSin(f, a)` Sine wave of frequency `f` Hz aligned so that phase 0 (the point where the rising sinusoid crosses 0) is at time `a` seconds. The amplitude of the output runs from -1.0 to +1.0. The sinusoid is most accurate at the start of a file; the accuracy falls off as the number of cycles increases (this is not usually noticeable).

`WSqu(l,h,a)` Square wave with low period `l` seconds and high period `h` seconds aligned so that a low period starts at time `a` seconds. Both `l` and `h` must be greater than 0 seconds. The output level of the low section is 0, the output level of the high section is 1.

`WTri(r,f,a)` Triangle wave with a rise time of `r` seconds and a fall time of `f` seconds aligned so that a rise starts at time `a` seconds. Either of `r` or `f` may be zero, but not both. The triangle output level is from 0 to 1.

`WEnv(r,h,f,a{,t})` Envelope with a rise time of `r` seconds, a hold time of `h` seconds, a fall time of `f` seconds with the rise starting at time `a` seconds. The output waveform is 0 before time `a` and after time `a+r+h+f` and is 1 during the hold time. At least one of `r`, `h` or `f` must be non-zero. The optional `t` (`t`ype) argument sets the shape of the rise and fall phases. The default is 0 for linear rise and fall, 1 for *cos*(θ) with θ set to -π/2 to 0 for the rising phase, 0 to π/2 for the falling phase), 2 for a raised cosine (1 + *cos*(θ))/2 with θ set to -π to 0 for the rising phase, 0 to π for the falling phase.

`WPoly(s,e,r,L)` Polynomial in time from `s` to `e` seconds and 0 outside this range. The value at time *t* (between `s` and `e`) is a function of (*t*-`r`) where `r` is a reference time. `L` is a list of 1 to 6 coefficients.

`WPoly(s,e,r,`c_0`,`c_1`,`c_2`,`c_3`,`c_4`,`c_5`)` is implemented as:

$y(t) = c_0 + c_1*(t\text{-}r) + c_2*(t\text{-}r)^2 + c_3*(t\text{-}r)^3 + c_4*(t\text{-}r)^4 + c_5*(t\text{-}r)^5$

`WT(s,e)` Ramp from time `s` up to time `e` of value at time t of (t-`s`) with value 0 before `s` and from `e`. Omit `e` to run to the end of the file, omit both `e` and `s` to start from the beginning.

Mathematical functions

The mathematical functions all have the form `Func(x)`, where `x` can be either a scalar or a vector and `Func()` is the operation. If x is a vector, the result is a vector with the function applied to each value otherwise the result is a scalar.

`Abs(x)` The absolute value of `x` (negative values are replaced by positive values of the same size). Use this to rectify a vector.

`Hwr(x)` Half wave rectify `x`. Negative values are replaced by zeros. This is faster than `Max(x,0)`.

`Sqr(x)` This calculates the square of `x`. `Sqr(x)` is the same as `x*x`, but is faster, particularly when x is a vector.

`Cub(x)` This calculates the cube of `x`. `Cub(x)` is the same as `x*x*x` but is much faster, particularly when `x` is a vector.

`Sqrt(x)` This calculates the square root of `x`. When `x` is a vector, negative values are set to 0. If `x` is a scalar, negative values cause an error.

`Ln(x)` Natural logarithm of `x`. Negative or zero `x` values generate -100 when `x` is a vector and an error when `x` is a number.

`Exp(x)` Exponential of `x`. If the result overflows, this is an error when `x` is a number and sets the largest allowed result when `x` is an array.

`Max(x,y)` The maximum of x and y. This can be used to set a lower limit, for example `Max(Ch(1),1)` is a vector with minimum value 1.

`Min(x,y)` The minimum of x and y. This can be used to set an upper limit, for example `Min(Ch(1),Ch(2))` can be thought of as the value of channel 1 with the maximum value limited by the value of channel 2 (or vice versa).

`Sin(x)` Calculates the sine of `x` (`x` is in radians). If `x` is in degrees, divide by 57.295779513 to convert to radians.

`Cos(x)` Calculates the cosine of x (`x` is in radians).

`Tan(x)` Calculates the tangent of x (`x` is in radians). The result can be infinite.

`ATan(x)` The arc tangent of `x`. The result is in radians in the range -`pi`/2 to `pi`/2.

`ATan(s,c)` Both `s` and `c` must be vectors or both must be scalars. The result is the arc tangent of `s/c` with the quadrant set by the signs of `s` and `c`. The result is in the range -`pi` to `pi`.

`Poly(x,L)` Replace `x` with a polynomial in `x` of order 1 to 5; `L` is a list of 1 to 6 coefficients. Use this to apply a non-linear calibration to a signal. For example: `Poly(x,a,b,c,d) = a + b*x + c*x*x + d*x*x*x`

You can insert spaces between operators and numbers and between round brackets and the items within them. You may not insert spaces between a function name and the opening bracket that follows it.

The multiply and divide operators have higher precedence than add and subtract, so `1/2+3*4` is `12.5`. You can use brackets to force other evaluation orders, for example `1/(2+3)*4` is `0.8`. Apart from that, evaluation is from left to right.

Dividing by a scalar value of zero is an error. Dividing by a vector holding zeros is not an error and generates special floating-point numbers for positive and negative infinities. These can cause problems in subsequent calculations (and they are difficult to display).

Cursor expressions

At version [10.14] we added the ability to include vertical and horizontal cursor values in the channel expressions. The expressions we provide are designed to be very similar to the ones provided with dialog expressions elsewhere in Spike2. However, they are a sub-set of these values. Only the expressions listed below can be used.

`Cursor(n)`	Where `n` is the vertical cursor number (0..9). This returns the cursor position in seconds, or 0.0 if the cursor does not exist. If there are duplicate views, this always gets the cursor positions from the first view.
`C0, C1 ... C9`	A short-hand version of `Cursor(0)`, `Cursor(1)` to `Cursor(9)`.
`HCursor(n)`	Where n is a horizontal cursor number (1..9). This returns the horizontal cursor position in y axis units, or 0.0 if the cursor does not exist. If there are duplicate views, this always gets the cursor positions from the first view.
`HC1 ... HC9`	A short-hand version of `HCursor(1)` to `HCursor(9)`.

Although having cursor positions as part of the virtual channel expressions can be very convenient, especially when dealing with waveform alignments, each time a cursor that is used in a virtual channel expression changes, the virtual channel is marked as invalid and will redraw at the first opportunity. If you display a lot of data, this may take a long time. You should use this feature carefully.

If you use cursor expressions in a helper dialog that assists you to build expression, the expressions is evaluated in the helper dialog, so does not persist. If you want to use these expressions dynamically in the virtual channel, you must type the cursor expression directly into the Expression field of the virtual channel dialog.

Channel values

At version [10.19] we added the ability to refer to the value of a channel at a particular time in the channel expressions. To avoid problems of infinite recursion you can only refer to virtual channels with a channel number less than the target channel.

`At(t, chan)`	Where `t` is the time and `chan` is the channel. The channel must be waveform based, or can be a Level Event channel,which has values 1 when the signal is high and 0 when low. It is an error to refer to other channel types. The time can include cursor expressions. If there is no waveform data at the time, the result is 0.

If the time is set by a cursor expression, changes to the nominated cursors will invalidate the entire channel, causing a repaint at the first opportunity. However, changes to the referenced channel data will not invalidate the channel unless the time falls in the displayed range, in which case the invalidation will only be of the changed data, so the display may not fully refresh if the result of the change involves a wider range of data. Values used for other than display purposes will always reflect the current channel value.

The helper dialogs that assists you to build expression do not recognise `At(...)`. If you want to use it in a virtual channel, you must type it into the Expression field of the virtual channel dialog.

Build expression

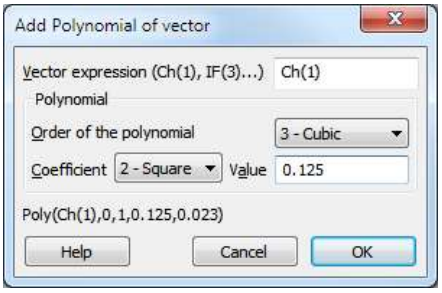
There is no need to remember all the expression functions; just click the >> button and choose from a list of possible items to add. All the commands in this section generate a text string that is inserted at the caret position (replacing any selection) in the Expression field. Simple command insert the text immediately; more complex commands open a dialog to build the expression. Commands that open dialogs display the expression at the lower left corner. You can choose from:

Waveform from channel	These commands create a waveform from an existing channel. You can choose from Copy waveform or Level channel, Event channel using Kernel, Event Instantaneous Frequency, RealMark data item and Time since Event.

Spectral functions	These commands create waveforms based on the spectral content of a channel. You can choose from Power in band or ratio of bands, Spectral edge, Mean frequency and Dominant frequency.
Generate waveform	Create a waveform independently of any channel, for example a sinusoid or a triangle wave or an envelope function.
Rectify and Absolute value	These commands inert the `Abs()`, `Hwr()`, `Max()` and `Min()` commands that rectify and limit values.
Mathematical functions	These commands insert mathematical functions (`Sqrt()`, `Sin()`, `Cos()`... and also the Poly() command which generates a polynomial of a vector or scalar.
Mathematical operators	Select one of these to insert the operator to replace the selection..
Previous virtual channel expressions	This lists expressions that you have used previously; the most recently used is listed first. Valid expressions are added to the list when you click Save expression, when you change to a different virtual channel and when you close the dialog with Close. The expressions are stored in the system registry.

Cursor Expressions
If you use a cursor expression as part of one of the helper dialogs that assist in building a virtual expression, the cursor expression will be converted to its current value. If you want to preserve the cursor expression so that the virtual channel responds to cursor position changes dynamically, you must type the cursor expression into the virtual channel dialog, not into the helper dialog.

Apply polynomial to data



This dialog is opened from the Virtual channel expression field. The Vector expression field is set to whatever was selected when this dialog opened, or to `Ch(1)` if nothing was selected. This field is not tested for validity. Each element `x` of the vector is replaced by a polynomial in `x`. You can set the order of the polynomial (order means the highest power of x used) in the range 1 to 5. The command is:

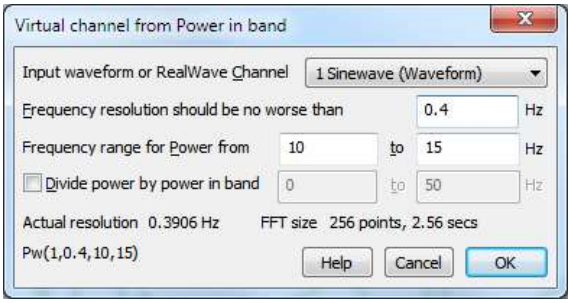
`Poly(x, a0, a1, a2, a3, a4, a5)`

where *x* is the vector expression (you can use a scalar, but this is not very useful) and the `a0` to `a5` are the coefficients of the polynomial (you must supply these). This expression generates the quintic:

$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5$

For lower order polynomials, omit coefficients from the right. For example, for a quartic (fourth order polynomial), omit `a5`, for a cubic omit `a5` and `a4`. To set coefficient values in the dialog, use the Coefficient field to select the required coefficient and then set its value.

Power in Band



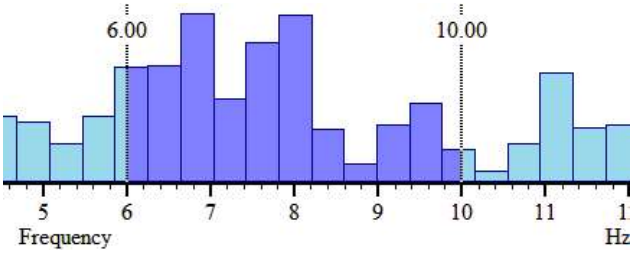
This dialog (accessed from the Virtual Channel build expression dialog) generates a waveform channel representing either the power in a frequency band, or the ratio of power in a band to the power in another band. The fields are:

Input waveform or RealWave Channel
Use this to select a suitable input channel to be analysed. The maximum values you can set for the frequency band is half the sample rate of this channel. If you change the channel, the frequency resolution field and the frequency range are reset to default values.

Frequency resolution should be no worse than ... Hz

When you open the dialog or change the channel, this value is set to the frequency resolution you would get for a FFT size of 256 points (which is usually a sensible value). Follow this link for more information about the frequency resolution. The actual frequency resolution and FFT size that will be used to achieve it are displayed in the dialog.

Frequency range for Power from ... to ... Hz



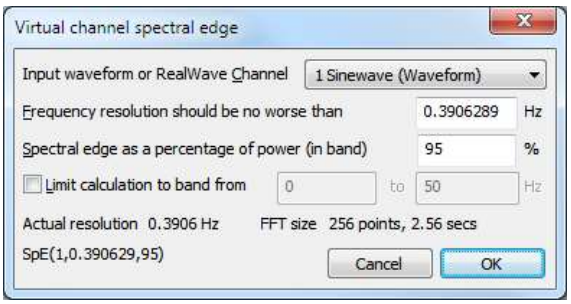
These fields set a frequency range where the power is to be calculated. The lower frequency must be less than half the sample rate of the source channel. The upper frequency must be greater than the lower. The results of the power calculation are in bins of fixed frequency width so it is unlikely that the frequencies you set will match the edges of the power bands. The power is calculated starting at the lower frequency and extending up to the upper one. If a band starts or ends with a fraction of a bin, then that fraction of the power in the bin is included.

Divide power by power in band from ... to ... Hz

If you check the box, the result is not power, but is the ratio of the power in two bands. Usually, the second band is set from 0 to half the sampling rate, giving a measure of the proportion of the total power, but you can set any frequency range you like as long as the lower frequency is less than half the sample rate. If the band used for division does not completely overlap the first band there is the possibility of division by zero. If this occurs, the result is set to 1000000 (rather than setting infinity, which is difficult to draw).

If you do not check the box, the output waveform is calibrated in power such that the output is the mean square of the data in the frequency band. That is, if the input data were a sinusoid of amplitude A units of a frequency in the band, the output would be $A^2/2$ units squared. Put another way, if the frequency band was set to include all frequencies, the output would be more or less that same as the input channel squared and then smoothed over a time of the FFT size.

Spectral Edge



This dialog (accessed from the Virtual Channel build expression dialog) generates a waveform channel holding the frequency at which the sum of the power starting at 0 (or the low edge of a defined band) is a used defined percentage of the total power (or the total power in a defined band). The fields are:

Input waveform or RealWave Channel

Use this to select a suitable input channel to be analysed. The maximum values you can set for the frequency band is half the sample rate of this channel. If you change the channel, the frequency resolution field and the frequency range are reset to default values.

Frequency resolution should be no worse than ... Hz

When you open the dialog or change the channel, this value is set to the frequency resolution you would get for a FFT size of 256 points (which is usually a sensible value). Follow this link for more information about the frequency resolution. The actual frequency resolution and FFT size that will be used to achieve it are displayed in the dialog.

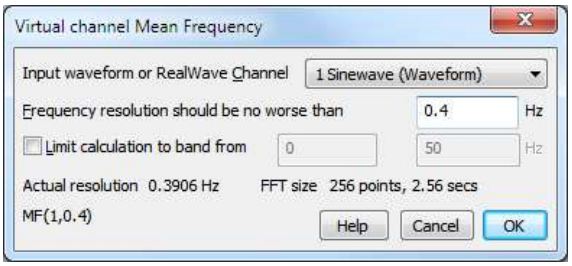
Spectral edge as a percentage of power (in band) ... %

You set the proportion of the signal as a percentage in the range 0 to 100. If you want the Median frequency as used in some EMG work, set the percentage to 50.

Limit calculation to band from ... to ... Hz

If you check the box, you can set limits for the band of frequencies that are used in the calculation. The range of bins used is inclusive from the nearest bin in the power spectrum to the low frequency up to the nearest bin to the high frequency.

Mean frequency



This dialog (accessed from the Virtual Channel build expression dialog) generates a waveform channel holding the mean frequency, defined as the sum of the product of Power and frequency divided by the sum of the power. You can limit the calculation to a range of frequencies. The fields are:

Input waveform or RealWave Channel

Use this to select a suitable input channel to be analysed. The maximum values you can set for the frequency band is half the sample rate of this channel. If you change the channel, the frequency resolution field and the frequency range are reset to default values.

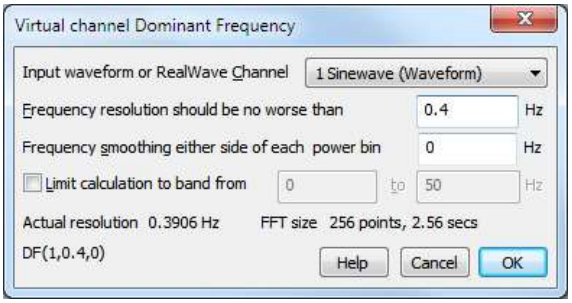
Frequency resolution should be no worse than ... Hz

When you open the dialog or change the channel, this value is set to the frequency resolution you would get for a FFT size of 256 points (which is usually a sensible value). Follow this link for more information about the frequency resolution. The actual frequency resolution and FFT size that will be used to achieve it are displayed in the dialog.

Limit calculation to band from ... to ... Hz

If you check the box, you can set limits for the band of frequencies that are used in the calculation of mean frequency. You could use this to exclude a DC offset from the calculation.

Dominant frequency



This dialog (accessed from the Virtual Channel build expression dialog) generates a waveform channel holding the frequency at which the most power was found. You can limit the search to a frequency range. The fields are:

Input waveform or RealWave Channel

Use this to select a suitable input channel to be analysed. The maximum values you can set for the frequency band is half the sample rate of this channel. If you change the channel, the frequency resolution field and the frequency range are reset to default values.

Frequency resolution should be no worse than ... Hz

When you open the dialog or change the channel, this value is set to the frequency resolution you would get for a FFT size of 256 points (which is usually a sensible value). Follow this link for more information about the frequency resolution. The actual frequency resolution and FFT size that will be used to achieve it are displayed in the dialog.

Frequency smoothing either side of each power bin ... %

If you leave this value as 0, the result is the power bin with the largest power and the result is quantised to the frequency resolution. The value you set here is divided by the frequency resolution to give the number of extra bins either side of each searched bin to include when looking for the maximum. If the extra bins would extend

before 0 Hz or after the bin corresponding to half the sampling rate, these non-existing bins are presumed to hold 0 power. When a maximum is found, the result is the weighted mean frequency in the range of bins.

Limit calculation to band from ... to ... Hz

If you check the box, you can set limits for the band of frequencies that are used as the centre of the search for the dominant frequency. The actual result can be outside this range if the frequency smoothing is set and a large power peak is just outside the range.

Spectral frequency resolution

The three virtual channel functions that generate spectral functions all have a frequency resolution parameter. To best use this you need to understand how the spectral functions are calculated. Spike2 converts from a waveform (the time domain) to power at a given frequency (the frequency domain) using a radix 2 Fast Fourier Transform (FFT). The analysis we do takes a sequence of n input data points and generates `n/2+1` output points that describe the power, spanning the frequency range 0 to half the waveform sampling rate. The output points are equally spaced in frequency.

The frequency spacing of the output points is thus the input waveform sample rate divided by the number of points, `n` in the transform. However, the sample rate is also the reciprocal of the sample interval, and `n` times the sample interval is the time duration of the input `n` points:

Frequency resolution `f` = sample rate / n = 1 / (n* sample interval) = 1/(input duration)

When you specify the frequency resolution, you are also specifying the time duration of the FFT transform. This means that if you want the power at time `t`, with a frequency resolution of `f` Hz, we must transform the data from time `t-1/(2*f)` to `t+1/(2*f)` seconds. That is, the more accurately you want to determine frequency, the less accurately the position in time where the frequency occurred can be known.

The details

The FFT we use does not accept arbitrary lengths of input data; it works with powers of 2 data points. If the number of samples set by a duration of `1/f` seconds is not a power of two (as is generally the case), it is rounded up to the next power of two. The value of `f` used will always be less than or equal to the value that you request. The resolution `f`, the FFT size used and the time period that the FFT spans are displayed in the dialog so you can make sure that the settings makes sense for your application.

Each waveform section that is transformed is first multiplied by a Hanning window. This weights the input data so that the values nearer to the centre get more weight in the result that those at the edges; it also prevents artefacts in the result.

If the sample interval for the virtual channel is more than half the FFT size in seconds, the result for a particular point is calculated by using overlapped transforms (the overlap factor is at least half the FFT width) of the input data from half an output sample interval before the point to half a sample interval after.

If the sample interval for the virtual channel is half the FFT size down to 1/8 the FFT size, each output point is calculated from one transform with the data centred on the output point.

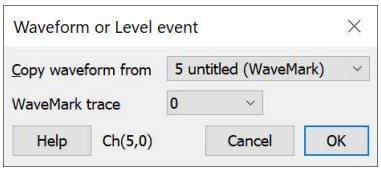
If the sample interval for the virtual channel is less than 1/8 the FFT size, we calculate the power spectrum for an interval of 1/8 of the FFT size, then interpolate the values.

Waveform from Channel

The commands in the Waveform from Channel section of the Virtual channel dialog expression field generate waveforms directly from data channels. The commands all open a dialog:

Copy Waveform or Level event
Event channel using kernel
Event instantaneous frequency
RealMark data item
Time since Event

Copy Waveform or Level event



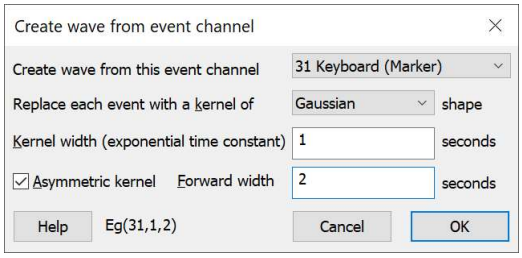
This dialog builds the `Ch()` command, allowing you to copy data from a waveform, RealWave, WaveMark trace and Level event channel.

If you select a level event channel, the output is 0 for a low level and 1 for a high level. You can generate a ramp from a Level Event channel with the Time since last event (`TEvt()`) command.

The WaveMark trace line is hidden unless the selected channel is a WaveMark channel and has more than a single trace. This allows you to choose which trace is the source of the waveform.

When you copy from a waveform or WaveMark channel you will likely want to set the virtual channel sample rate to match the source channel.

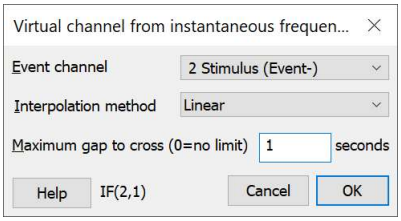
Event channel using kernel



This dialog lets you choose between the various Event kernel functions: Rectangular, Triangular, Raised Sinusoid, Gaussian and Exponential. These replace each event by a shape with unit area (time in seconds, amplitude in Hz). The shapes are summed, so the result is a representation of event frequency, smoothed by the selected kernel shape and width.

Normally you will leave the Asymmetric kernel box unchecked so that the shape that replaces each event is symmetric about the event time. If you select an Asymmetric kernel, a second time field (Forward width) appears to set the right hand side time constant. The original time constant sets the left side time constant.

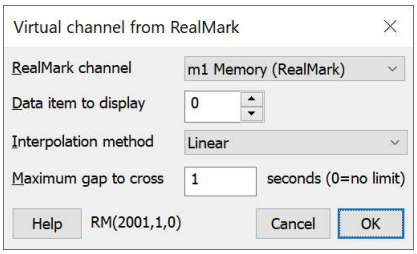
Event instantaneous frequency



This dialog generates the `IF()` and `IFC()` commands, which convert a channel of event times into a waveform of the channel instantaneous frequency. The frequency points at the events can be linked by a straight line (linear interpolation) or by a cubic spline.

If you set the Maximum gap to cross field to a non-zero value, this will generate gaps in the waveform if the gap between two instantaneous frequency values exceeds the gap. Setting a 0 value removes any limit and all points are linked, regardless of the gaps between them.

RealMark data item

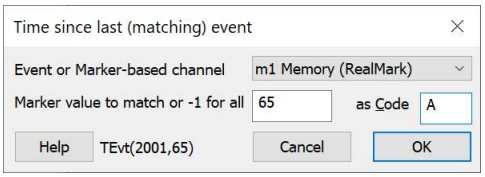


This dialog generates a waveform from the data values attached to a RealMark channel. Each RealMark data item has a list of floating point values for data items associated with it, so you must choose which one to display with the Data item to display field.

The Interpolation method field sets how the points are joined. You can choose a straight line (linear interpolation) or cubic spline interpolation.

If you set the Maximum gap to cross field to a non-zero value, this will generate gaps in the waveform if the gap between two RealMark items values exceeds the gap.

Time since Event



This dialog generates the `TEvt()` command to display the time interval since the last event on the channel. If the channel type is Event Rise or Event Fall, the second line is hidden.

If the channel is Marker-derived (or Event Level), you can choose to only start timing from events that match a particular marker code and the timing is cancelled by any other marker code.

The Marker code can be specified as a number in the range 0 to 255 or -1 to mean no matching in use, or you can use the as Code field to specify the marker code as a single printing character or as two hexadecimal digits.

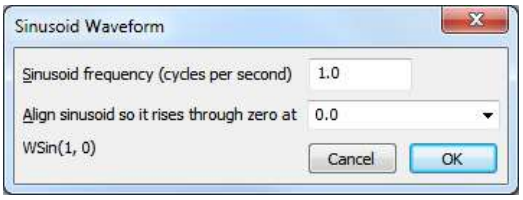
Generate Waveform

The commands in the Generate Waveform section of the Virtual channel dialog expression field generate waveforms without any reference to data channels. Note that these commands generate data that can only exist in the time span from 0 to the length of the data file defined by other channels. If you have a completely empty data file you will need to add a real or a memory channel with some data at the maximum time you want to generate data for.

From version [10.14] you can use cursor positions to set timing alignments, for example `WEnv(C2-C1, C3-C2, C4-C3, C1)`. However if you use `Cn` in this dialog, the values are accepted, but will be converted to their numeric equivalents. To enter cursor positions as `Cn` and have them tracked dynamically, you must enter them in the Virtual Channel dialog expression field.

You can generate:

Sinusoid



This generates a `WSin()` command that extends for the entire length of the data file. You set the frequency in Hz and a point in time at which the sinusoid rises through zero. The generated waveform has mean value of 0 and a peak to peak amplitude of 2. The waveform frequency must be less than twice the sampling rate set for the channel.

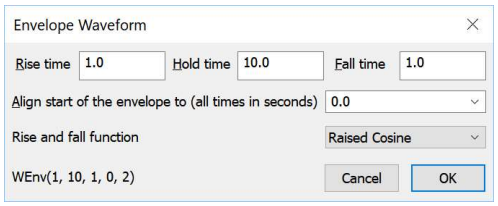
Square wave

This generate a square wave that extends for the entire length of the data file with the `WSqu()` command. You define the wave by setting the low (waveform value 0) and high (waveform value 1) time periods and an alignment point where the waveform rises from zero. The period of the waveform (low plus high time) must be at least two sample periods of the virtual channel.

Triangle wave

This generates a triangle wave that extends for the entire length of the data file with the `WTri()` command. The waveform is defined by a rise time and a fall time in seconds and by a time at which a rise starts. The period of the wave must be at least two sample periods of the virtual channel.

Envelope

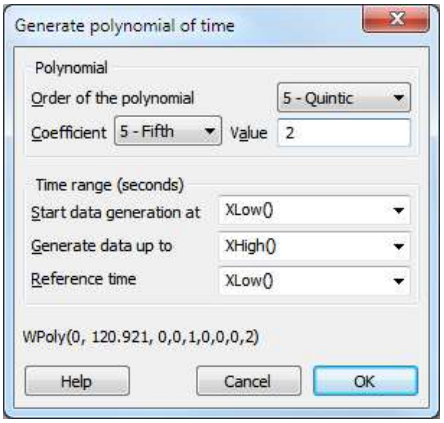


An envelope is zero everywhere except in one region where the value rises to 1, holds at 1, then falls back to 0. It is implemented by the `WEnv()` command. It is defined by the start time, rise time, hold time and fall time, all in seconds. Prior to Spike2 version 9.01, the rise and fall sections were linear. From version 9.01 onwards there is an additional Type argument to specify how the rise and fall sections are implemented. You can choose linear or sinusoidal sections:

Function	Type	Implemented as
Linear	0 (or omit)	Linear interpolation of 0 to 1 and 1 to 0.
Cosine	1	cos(θ) with θ set to -π/2 to 0 for rising, 0 to π/2 for falling.
Raised Cosine	2	0.5 * (1 + cos(θ)) with θ set to -π to 0 for rising, 0 to π for falling.

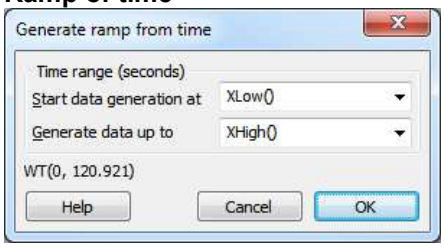
Polynomial of time

The `WPoly()` command generates polynomials in time relative to a reference time, time being measured in seconds. The polynomial has a value in a defined time range and is zero elsewhere. Such curves can be used to create complex envelopes, or to subtract out a curve generated by the interactive curve fitting routines. The order of the polynomial can be set to: Constant, Linear, Quadratic, Cubic, Quartic or Quintic, representing the highest power of the time used. The Coefficient field selects the coefficient to set in the Value field.



We expect polynomials to be used over a restricted time range; open ended time ranges will tend to generate very large data values at large time distances away from the intended time range.

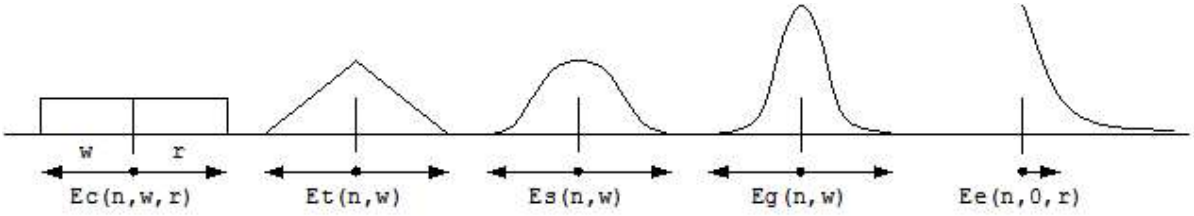
Ramp of time



The `WT()` command generates a ramp from a start time up to, but not including an end time. The data is zero outside this time range. Within the time range, the ramp value is the current time in seconds minus the start time. You will usually want to scale the ramp by multiplying it by a constant value.

The Event to Waveform functions

The `Ec(n,w,r)`, `Et(n,w,r)`, `Es(n,w,r)`, `Eg(n,w,r)` and `Ee(n,w,r)` functions convert event channel `n` into a virtual wave and can be used anywhere in an expression that you can use the `Ch()` function. They replace each event by a kernel (shape), centred on the event time. For an event at time `t`, the kernel extends from `t-w` to `t+r` seconds except for `Ee()`, which extends from `t-8w` to `t+8w` seconds. Normally you will omit `r`, in which case the shapes are symmetrical with r set equal to `w`. The resulting waveform is the sum of the kernels for all the events. The area of each kernel is unity, so the area under the waveform between any two times is the number of events in that time interval.



The `Ec()` function simply counts the events in the time range. This is the fastest analysis method, but produces the most jagged output. The `Et()` function weights the events with a triangle function. This is slower than `Ec()`, but much faster than `Es()` and `Eg()`. The `Es()` function weights the events with a raised cosine. The `Eg()` function weights the events with a Gaussian curve extending to 4 sigmas.

The `Ee()` function is rather different from the others as it is not suitable for use as a smoothing function and is more likely to be used in a single-sided form. It weights the events with an exponent `exp(-t/r)` to the right and `exp(-t/w)` to the left (t stands for the time difference between the point and the time). The exponent extends to 8`w` to the left and to 8`r` to the right.

If none of these conversions are suitable, you can define you own weighting functions and create a new (not virtual) channel with the `EventToWaveform()` script command.

Duplicate Channels

This Analysis menu command duplicates selected channels in the current time or result view. This command is also available from the channel context menu (right-click on a channel, select Channel n, then select Duplicate channel).

Duplicate channels share data, channel scales and comment with the parent and inherit the channel settings. Once you have duplicated a channel you can change title, display mode and y axis range independently of the original. With time view marker data, you can change the marker filter and the marker code (1-4) that is displayed. The script equivalent of this command is `ChanDuplicate()`.

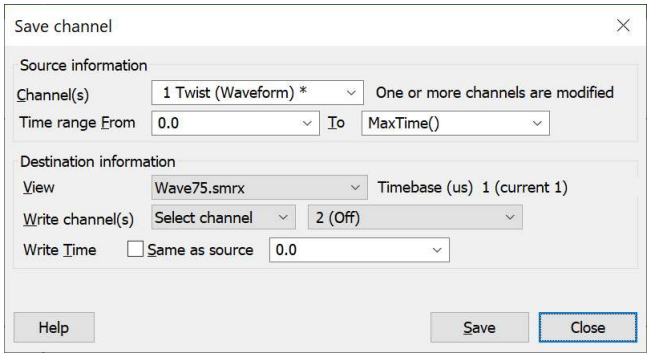
The channel number of a duplicated channel is displayed as the original channel number plus a letter. The first duplicate of a channel gets the letter a, the second b and so on up to z, then A to Z are used. There is a limit of 52 duplicates of a channel. There is a limit of 2000 duplicate channels in a view. Duplicate channels are deleted with the Analysis menu Delete Channel command. New duplicates get the lowest available free letter. Duplicated channels are added into the display as close as possible to the real channel they duplicate, or if there are already duplicates, as close as possible to the last duplicate. If the closest channel is part of a group, the new channel is added after or before the group, depending on the channel sort order set in the Edit menu Preferences.

Duplicate RealMark as waveforms

Spike2 [10.18] adds the ability to duplicate a RealMark channel with N attached items to generate N channels (the original and N-1 duplicates) set to display each attached value as a waveform. To active this command, right-click on a non-duplicate RealMark channel and select the Channel n-> item in the pop-up list. This will pop-up another menu, and the top item will be Duplicate as N waveforms. You are warned if the channel already has duplicates.

From [10.20], the created channels are optimized based on the displayed data (previously they had the display range of the original, if it was in waveform mode).

Save channel



This is a very capable and powerful command that copies a time range of one or more channels to a the same or another time view. The source channels can be disk-based, or virtual channels or memory channels. The destination channels are always disk-based channels and have the same type as the source channels. This is often used to make a virtual or memory channel permanent, but can also be used to move channels with an optional time shift to a different file.

Data is saved as displayed; any changes due to channel processes or marker filters are preserved in the output. The dialog displays a warning if this is the case. To save a memory buffer to a channel in the same window you can use the simpler memory buffer Write to channel command, which also allows you to set the destination channel type.

If a source channel has no channel comment, Spike2 will create one if the source channel is a Virtual channel (when it will save the channel expression) or if the source channel has Channel Processes applied (when it will list them), or if the source channel has a Marker filter applied. The comment will also hold the original source channel if it differs from the destination channel. The equivalent script command is `ChanSave()`.

You can use this command to overwrite existing waveform data or append waveforms to the end of a channel. You cannot use this to fill in gaps in disk-based waveform channels. You can append event-based data to existing disk-based event channels, but you cannot overwrite or merge data with them.

The dialog is in several sections:

Source information	This sets the channels and the time range to copy. It also displays a warning if one or more of the channels is modified.
Destination information	This sets the destination of the data.
Error message area	This area (empty in the above image) holds explanations of any errors in the settings that prevent Saving data.
Commands	This area at the bottom of the dialog holds Buttons and an area used for information messages.

Channel(s)

This field allows you to select one channel or selected channels from the drop-down list, or you can type in a list of channels as a channel specification, such as: `1..3,7..10,23` or `T=Fred` to match channels with titles that contain the text `Fred`. If any channel is modified by a Channel process or has a Marker filter, a message appears to the right of the field. If you drop down the channel list, modified channels have an asterisk (*) at the end of their description.

Time range From, To

These two fields set the range of data to copy. The standard values are `0` and `MaxTime()` to copy the entire channel, but you can copy selected sections. Data is copied from the From time and up to and including the To time.

View

You can choose which View is used as the destination. When you open the dialog this is set to Current to copy data to the same view as it is read from. You can choose any Time view that is open in Spike2. The field to the right of the view displays the timebase of the view, in microseconds. If this view is not the current view, it also displays the timebase of the current view. If you target a view with a different timebase from the current view, Spike2 will write the closest approximation of the data it can in the destination view. Waveform data is resampled to the rate that is as close to the original as possible, and data is interpolated by cubic splines. Event data is written as close as possible in time to the source time (plus any time shift).

Write channel

The first field to the right of this label sets how the output channel is set. There are three possibilities:

Same as source	Each channel to be copied is written to the same channel numbers as in the source. If the View is set to `Current`, you must adjust the time ranges so that the source and destination time ranges do not overlap.
Lowest unused	Channels are added in the order set in the Channel(s) field using the lowest unused channel in the destination view. Once the available channels are exhausted, copying stops.
Select channel	This is only available when a single channel is set in the `Channel(s)` field. A new field appears to the right in which you can select an unused channel.

Write Time

This determines where the range of data selected by the From and To fields is copied to in the destination. If you check the Same as source box, data is copied to as close to the same time (timebase permitting) as possible. Otherwise you can set the destination time. If you set a destination time, the data is shifted by the Write time minus the From time.

Help

This command opens the information you are reading.

Save
Click this button to save the selected source data to the set destination. Errors in the dialog settings disable this button. If there is any problem with a Save operation, a message box appears with the reason. If the operation takes more than a second or so, a progress dialog appears, offering you the opportunity to interrupt the operation and delete any newly created channels.

You can record the equivalent `ChanSave()` script command ff you Turn Recording On before opening this dialog.

Close
Closes the dialog.

Create a new, empty Time view
If you want to manipulate channels to a new file, you need a way to generate one. There is no menu command to do this (unless you set a sampling configuration, sample and immediately quit sampling). A simple way to create a new, empty Time view is to use the script language command:

```
FileNew(7, 1, <upt>, 1, <maxT>, <nChan>);
```

`<upt>` The desired file resolution, in microseconds. This is typically set to 1.0 unless you need to achieve a particular resolution (for instance so you can get a particular waveform rate).

`<maxT>` The length of the new empty file, in seconds. If you write data to the file after this time the file maximum time will increase. If in doubt, set a small value that you will exceed, for example 1 second.

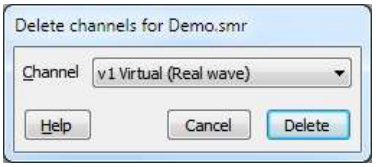
`<nChan>` The maximum number of channels in the file. There is always a minimum of 32 channels.

The easiest way to run this command is to use the Script menu Evaluate... command to open the Evaluate window. Copy the following command and paste it into the Evaluate window, then click the Eval() button:

```
FileNew(7, 1, 1.0, 1, 10.0, 64);
```

This will generate an empty file with 1.0 microsecond resolution of length 10.0 seconds with space for 64 channels. You will be prompted for a file name when you save it. This generate a 64-bit .smrx file. If you need to generate a 32-bit .smr file, add `,1` after the `<nChan>` field.

Delete channel

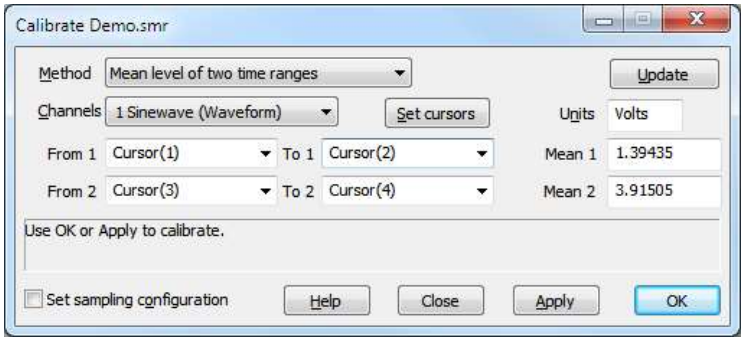


This removes a channel from a Spike2 data file permanently or deletes a channel created with New Buffer from the Memory Buffer command. If you delete a channel with duplicates, the duplicates are also deleted. Deleting a duplicated channel removes the duplicate only. You can also delete channels (except the last one) from XY views and duplicated result view channels. It is also possible to delete items (not the entire channel) from a memory buffer with the Memory Buffer Delete Items command.

Before version [10.01], you were not allowed to delete a channel that was part of a file that had just been sampled until the file was closed and reopened. You are now allowed to delete a sampled channel, but only after sampling has stopped.

Calibrate

If a waveform, WaveMark or RealWave channel has sections with known amplitudes, offsets, slopes or areas, you can calibrate it. Spike2 supports a wide variety of calibration methods for these channels. You can calibrate a single channel, or all selected channels (as long as they have the same calibration values). If you make a mistake, you can Undo the calibration changes. You can also calibrate from a script.



To calibrate your data, click on a time window and open the calibrate dialog. If there are suitable channels selected, the Channels field shows them, otherwise you must choose one. Once the dialog is open you can select channels in the time window; these are added to the selected list in the dialog.

Next, choose a calibration method. The dialog contents change depending on the method. All methods require you to select data areas with known values; this is most easily done with cursors. Click Set cursors to position appropriate cursors in the time window and dialog. You can also type in the times and use expressions like `(C1+C2)/2`. If the method requires two time ranges they must not overlap.

When you change the method or the channel or click the Update button, Spike2 collects the current calibration values from the (first) channel in the Channels field.

Once you have selected your data you must type in the calibration values, and also set the units of the data. A text box displays Use OK or Apply to calibrate or an explanation of any problem that prevents calibration. Click Apply to calibrate and leave the dialog open or OK to calibrate and close the dialog.

If you check the Set sampling configuration box, any calibration changes are passed through to the same channel number in the sampling configuration. If you calibrate with a data file that you are sampling, or that you have just sampled and not yet saved, this box is checked automatically when the dialog opens.

What calibration does

Waveform channels are stored as 16-bit integers (range –32768 to 32767) that are scaled into user units by a `scale` and `offset`:

User units = 16-bit integer * `scale` / 6553.6 + `offset`

The factor 6553.6 is present so that if the input data range spans –5 to +5 Volts (as is usually the case with a CED 1401 interface), the relationship is:

User units = input in Volts * `scale` + `offset`

The `scale` and `offset` are the same values as you set in the sampling configuration dialog or in the channel information dialog in a time window. Calibration changes the `scale` and `offset` so that the displayed data matches your user units.

RealWave channels are stored as 32-bit floating point numbers. Calibration rewrites this data and will take a noticeable time when working on very large data files. You cannot calibrate RealWave data if any channel process is attached.

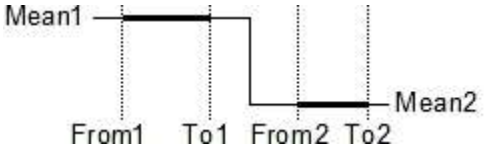
You cannot calibrate any channel with an attached process that has changed the channel scale or offset value. You cannot calibrate a Read-only file as the changes will not be written to the file.

Calibration methods

The Method field of the calibration dialog sets how the calibration is performed:

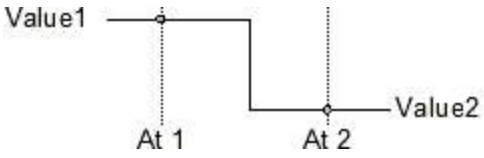
Mean level of two time ranges

You define two time ranges (From1 to To1 and From2 to To2) and supply the mean levels of the ranges (Mean1 and Mean2). The data in the time ranges must have significantly different levels and the mean values you set must not be the same.



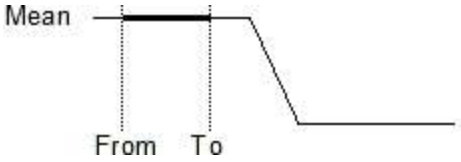
Values at two times

You define two times (At1 and At2) and the two values (Value1 and Value2) that correspond. The two values must be different and the data at the two times must also be different.



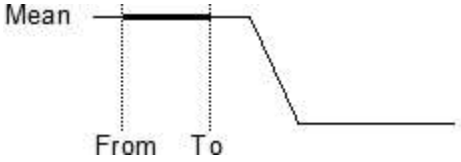
Set offset from mean of time range

If your data is calibrated, but suffers from baseline drift, you can use this method to redefine your base line. Set a time range (From and To) and the mean value of the data in the range (Mean). The data scaling is not changed.



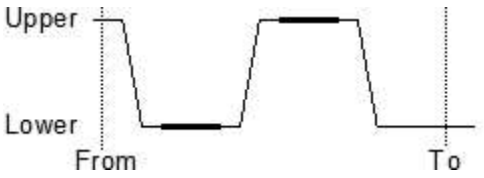
Set scale from mean of time range

This method is for data with a fixed offset (usually 0) and variable gain. Set a time range (From and To) and the mean value of the data in the range (Mean). The data zero level is preserved. The mean value you set cannot be 0 and the mean level of the data before calibration cannot be 0.



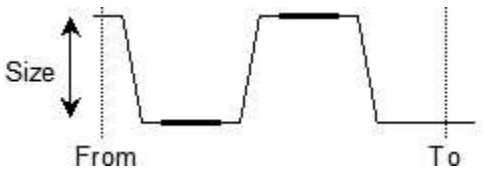
Square wave, upper and lower level

This calibrates a square wave with known Upper and Lower levels; Upper is the larger value before calibration. The time range (From and To) must contain at least three transitions. Spike2 detects the transition points between high and low values. For each high or low section, the first and last 25% of the data is ignored. The upper and lower levels must differ.



Square wave, amplitude (Size) only

This method detects a square wave in the same way as the Square wave upper and lower level method. In this case, you supply the Size (difference between the upper and lower levels) of the waveform. The zero level is preserved.

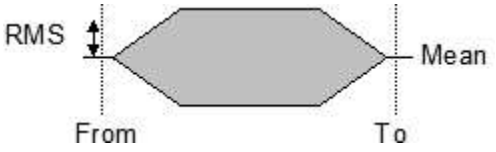


Peak to peak amplitude and mean

This method calibrates based on the peak to peak amplitude (Pk-Pk) and Mean value of the data in the time range (From and To). This could be used for a sinusoidal waveform of known amplitude. The Peak to Peak value must be non-zero.

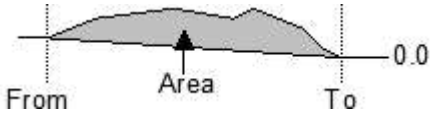
RMS amplitude about mean

This method calibrates based on the RMS (Root Mean Square) amplitude of the data in the time range (From and To) and the Mean level. The RMS value must be non-zero.



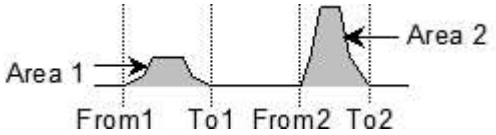
Area under curve, assume zero at end

Use this method to calibrate a rate based on a known area (for example flow rate based on volume). The rate is assumed zero at each end of the time range (From and To) to allow for a drifting base-line; the base line is assumed to run linearly from From to To. Set the area as rate units times seconds.



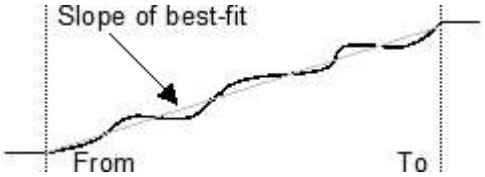
Areas under curve, two time ranges

Use this method when you know the area under the data for two sections (From 1 to To 1 and From 2 to To 2) of your data. The mean level of the two sections must be different. The area you set is in the units of the channel units times seconds.



Set scale from slope (no offset change)

Use this method when you know the slope of a section of the data (for example to calibrate position on a known velocity, or velocity on a known acceleration). The calibration sets the slope of the best-fit line to the data in the range to the slope value you set. This method does not change the offset, so you may need to combine this with the Set offset from mean of time range method for a full calibration. The units of the slope you set are the channel units per second.

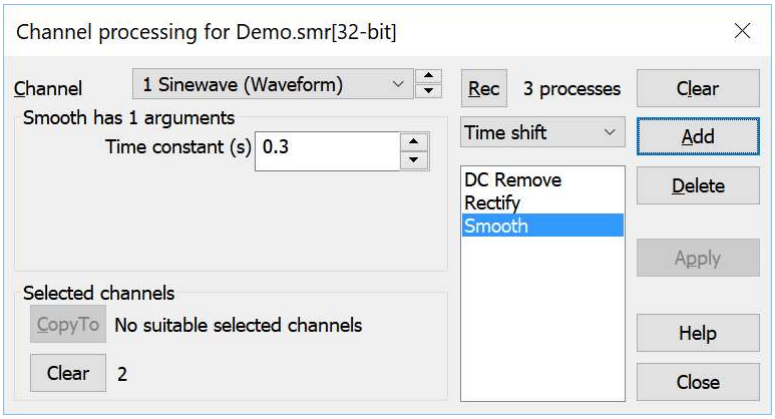


Channel process

A channel process is an operation, for example rectification, applied dynamically to data channels. The original data is not changed, but all users of the channel see data modified by the process. Multiple processes can be applied, for example DC removal then rectification then smoothing of a waveform channel or time shifting and de-bounce of an event channel. Every time you use the data, the processing is applied, so using a processed channel is slower than using raw data. If a channel has an attached process, the channel number is displayed in red.

Before Spike2 9.02, channel processes could be applied to Waveform and RealWave channels only. Now you can apply processes to any channel type, but note that some processes are specific to waveform data and some to event data.

If a channel is already part of an analysis operation, such as a waveform average, adding a process and continuing the analysis may generate incorrect results. This is particularly the case with processes that change the sample rate or interpretation of waveform data. Restart the analysis operation to ensure correct results.



To add a process, open the Channel Process command from the Analysis menu and select a Channel or right-click the target channel and select Channel Process. The list to the left of the Apply button shows processes to apply in order. Editable values for the selected process are displayed on the left of the dialog. Clear removes all processes from the channel. Add appends a new process set by the drop down list to the left of the Add button to the end of the list. Delete removes the selected process. When recording is enabled, Apply copies changed argument values to the process and records the script to make the change, otherwise changes are applied immediately.

Some processes have arguments that you can edit. Unless you have script recording turned on, changes made by editing are applied immediately. The spinner buttons next to the value adjust the value by the smallest useful change, or by 1% of the current value, whichever is the larger.

If you have multiple processes, you can change the order by clicking on a process in the list and dragging it to a new position.

Selected channels

You can copy all the processes set for the current channel to other channels. Select the target channels by clicking their channel numbers (usually on the left edge of the time view) and click the CopyTo button. You can only copy processes to a channel that is capable of applying the current process list.

You can also clear any all channel processes from any selected channels. In the example image, the only channel capable applying the list of processes is channel 1, which is the source channel, so no channels show up in the CopyTo list. However, channel 2 (an event channel), does hold a process, so it can be cleared.

Restrictions caused by processing

Some processes change waveform channel scales and offsets (these are the values that translate between a 16-bit integer representation of a waveform and user units). Such changes do not affect the data on disk and are removed when the process is removed. The Calibrate dialog and the Channel Information dialog will not allow you to change the channel calibration if an attached process has changed the channel scale or offset. You are not allowed to calibrate a RealWave channel that has any attached channel process.

Process types

You can add processes of the following types: Rectify, Smooth, DC Remove, Slope, Time shift, Down sample, Interpolate, Match channel, RMS amplitude, Median filter, Fill gaps, No Nan, Debounce.

Recording actions and the Rec button

If you turn script recording on in the script menu, the dialog behaviour changes and edits you make to arguments are not applied and recorded until you click the Apply button. This is so that we do not fill the recorded script up with every single change you make. Recordings made this way record each action, so adding a smoothing process is recorded in two parts: adding the smoothing process, then setting the desired time constant.

The Rec button is normally hidden and only appears when you enable script recording. If you click Rec, the channel process state is recorded neatly in an easy to understand way with all the argument values on one line. If you want to see the shortest script sequence to create your current channel processes do the following:

1. With script recording off, set up the processes you want in the dialog.
2. Turn script recording on in the Script menu
3. The `Rec` button will appear, click it.
4. Turn script recording off

Rectify (wave)

This replaces all negative input values with positive values of the same magnitude. There are no additional arguments required to define this process. This is a non-linear process, in the sense that the output is not related to the input by a transform of the form `output = scale*input +offset`. This process changes the channel offset to ensure that the full range of the rectified data can be expressed in a Waveform channel. The result of Rectify on a Waveform channel can be limited if the channel has a non-zero offset.

Smooth (wave)

This process has one argument, a time period in seconds, p. The output at time t is the average value of the input data points from time t-p to t+p seconds. This process does not affect the channel scale or offset. More about smoothing.

DC Remove (wave)

This process has one argument, a time period in seconds, p. The output at time t is the input value at time t minus the average value of the input data points from time t-p to t+p, that is, it is equivalent to the original signal minus the result of the smoothing operation, described above. This process does not affect the channel scale, but the channel offset is set to zero.

Slope (wave)

This process has one argument, a time period in seconds, *p*. The slope at time *t* is calculated using an equal weighting of the points from time *t*-*p* to *t*+*p*. This is done by calculating the mean of the points ahead and the mean of the points behind each data point, and the slope is taken from the line through the centre of the points behind to the centre of the points ahead. This calculation is equivalent to applying an FIR filter, it is not a least squares fit through the points. This method is used because it can be applied iteratively very efficiently to a long run of data. If you apply this process to a channel, the channel scale, offset and units change. If the current channels units are no more than 3 characters long, `"/s"` is added to them, so units of `"V"` become units of `"V/s"`. If there is not sufficient space, the final character of the units becomes `"!"` to indicate that the units are no longer correct. The offset becomes 0, and the scale changes to generate the correct units.

Time shift (any channel)

This process has one argument, the time to shift the data. A positive time shifts the data into the future (to the right); a negative time shifts the data into the past. To preserve waveform timing but change the positions where the data was sampled, use the Interpolate or Match channel processes. Beware using negative time shifts online as this will attempt to shift data that does not yet exist, which can cause drawing and data processing problems.

Down sample (wave)

This process changes the sample rate of the wave by taking one point in `n`. There is one argument, prompted by Use one point in, which is the down sample ratio. You might want to use this command after filtering or smoothing a waveform. This is a faster operation than Interpolate.

Interpolate (wave)

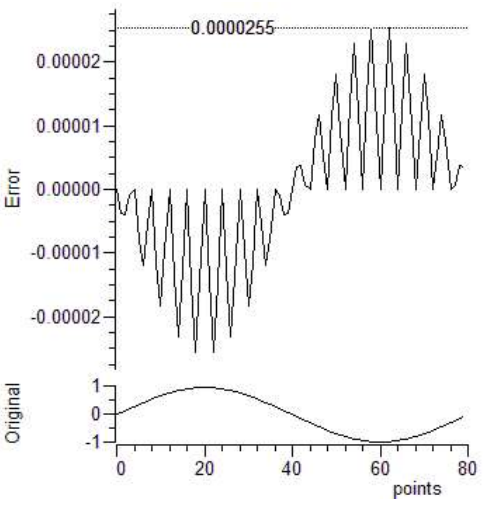
You can change the sample rate of a channel and set the time of the first data point with this process. Interpolation is by cubic spline of the original data. No data is generated outside the time range of the original data points. Interpolation is not too slow, but if you increase the sampling rate it will take longer to draw and process data.

The first argument, Sample interval, is the time in seconds between output data points. You can type in expressions here so, to set 123 Hz, type `1/123`. The actual interval is set as close to the requested one as possible (given the time resolution of the file). When you create the channel process, this is set to the current sample interval of the channel.

The second argument, Align to, aligns the output data to a time. It must be positive. The process places data points at this time plus and minus multiples of the sample interval. You can use this to convert multi-channel data sampled by a 1401 into simultaneously sampled data by giving all the channels the same alignment and sample rate.

Cubic spline interpolation assumes that the input waveform and its first and second differentials are continuous. If the input data was suitably filtered this will be not too far from the truth. Not all data is suitable for cubic splining; splining across step changes generates ringing effects that were not present in the original signal.

Cubic spline interpolation is better than linear interpolation for continuous data, but it is not perfect. The graph shows the error between a sine wave sampled with 20 points per cycle and splined to 80 points per cycle and a calculated 80 points per cycle sine wave. The maximum error is small, in this case 0.0000255 of the original signal (compared to 0.003 for linear interpolation). However, the maximum error increases rapidly the fewer points there are per cycle of the original data. With 5 points per cycle, the maximum interpolation error for a sinusoid is almost 1 per cent of the original signal (compared to 19% for linear interpolation).



Match channel (wave)
This is the same as Interpolate, except that the sample interval and alignment are copied from a nominated channel. The initial channel is set to the current channel, so adding this process should have no visible effect (apart from causing a redraw).

RMS amplitude (wave)
This process has one argument, a time period in seconds, *p*. The output at time *t* is the RMS value of the input data points from time *t-p* to *t+p* seconds. For waveform data, the output may be limited by the 16-bit nature of the data if the channel offset is very large compared to the scale factor.

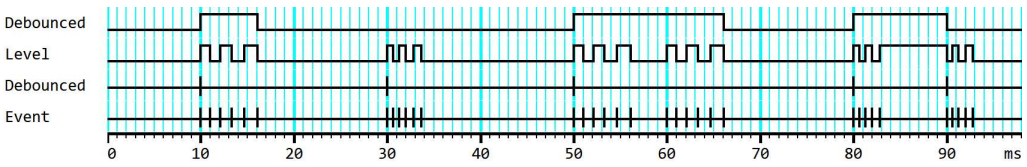
Median filter (wave)
This process has one argument, a time period in seconds, *p*. The output at time *t* is the median value of the input data points from time *t-p* to *t+p* seconds. The median is the middle point after the data has been sorted into order. This can be useful if your data has occasional points with large errors. This filter is slow if *p* spans a large number of data points; set the time period to the smallest value that removes the outlier points. You may find it better to use this method to identify outliers (for example by looking for large differences between raw and median-filtered data) and remove them by other methods (for example Linear Prediction) rather than using the results of the filter directly.

Fill gaps (wave)
Waveform and RealWave channels can have gaps, where no data exists. This process guarantees that the channel has no gaps by filling in gaps greater than a specified time with a fixed level and by linearly interpolating across smaller gaps. There are two arguments: the maximum gap in a waveform to interpolate across and the level to use to fill gaps wider that the maximum gap. At the start and end, gaps less than the specified maximum are filled by duplicating the first or last data point. This is a non-linear process (see Rectify, above). To fill short gaps (especially those generated by the Skip NaN process) you could consider using Linear Prediction.

Skip NaN (wave)
A NaN (Not a Number) is a floating point value that is either undefined in some way, or is infinite (the result of dividing by zero or an arithmetic overflow). These values can occur when RealWave data is imported into Spike2 (for example, some systems that transmit data over a link uses NaN to flag missing data values), or in a virtual channel if you divide by zero. This process removes such numbers from the channel, leaving gaps. Applying this process to a Waveform channel is a waste of processing time as Waveform channels (based on 16-bit integer data) can never hold invalid values. The process has no arguments. You may want to follow this process with the Fill gaps process to keep the data contiguous. Alternatively, you could consider using Linear Prediction to repair the damage by guessing the NaN values (in this case do not use Skip NaN and the NaN values will appear as if they were 0's).

Debounce (event)
This process removes events that are too close to the previous event, usually caused by recording events from a mechanical switch. For all event-based channels except Level events, this process simple removes an event if the preceding one is closer than the Minimum interval (set in seconds). For example, if there are events at 10, 11, 12 and 13 milliseconds with none before and none after, if the minimum interval is set to anything greater than 1 millisecond, only the event at 10 milliseconds will remain. This is equivalent to the on-line event input debounce.

For Level event channels, intervals at or more than the Minimum interval from the previous edge are preserved, but if this results in a pulse (high or low) with a duration of less than the minimum interval, the pulse is removed.

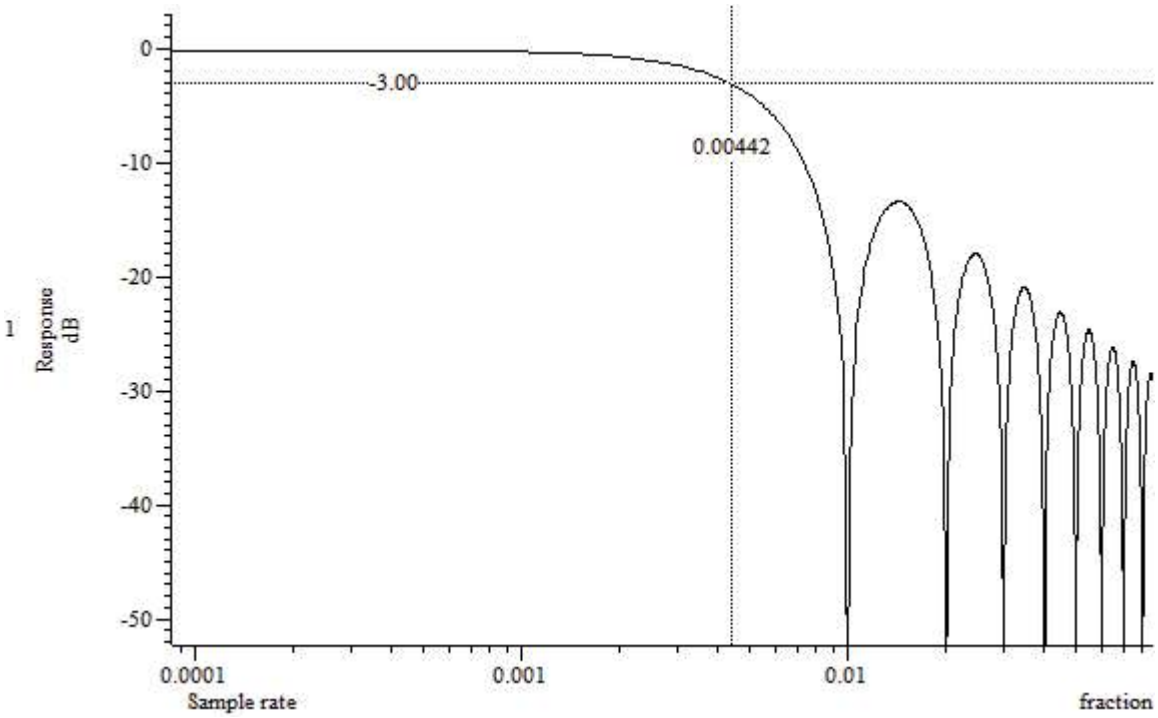


In this example, the bottom trace (Events) has been duplicated and then de-bounced with a Minimum interval of 0.004 seconds (4 ms) in the second trace. All events with another event before them closer than 4 ms are

removed. The third (Level) trace has also been duplicated and de-bounced with the same interval. The first group of events Get reduced to a pulse, the second group also get reduce to a pulse, which is then discarded as it lasts less than 4ms. The third and fourth groups become amalgamated into a single pulse as they are not separated by 4ms. The fifth and sixth groups are interpreted as a pulse with a noisy start and a noisy end.

More about smoothing

The smoothing function is equivalent to an FIR filter with n coefficients, each of value 1/n. This generates a filter with the following characteristics:



Frequency response of smoothing filter (n=100).
Frequency expressed as a fraction of the sampling rate.

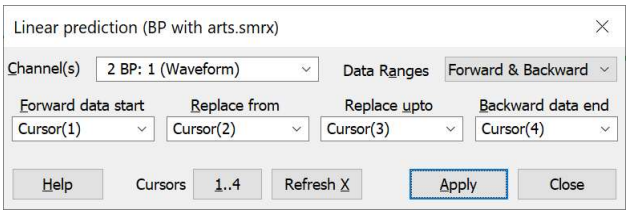
This is not a particularly good filter, but it suffices for many purposes and has the advantage that it can be realised efficiently. The position of the zeros is determined by the number of coefficients and by the sampling rate. If the sampling rate is `s` and there are `n` coefficients, the zeros occur at `i*s/n` Hz, where `i` is 1, 2, 3... If we consider the first section of the filter, the 3 dB point is at approximately 0.44 of the position of the first zero. If we express this in terms of the smoothing dialog, if you enter a time period t, the first zero occurs at a frequency given by `1/(2*t+si)` where `si` is the sample interval of the channel. If `si` is small compared to `t` then this is `0.5/t` Hz. The 3 dB point occurs at `0.44/(2*t+si)` or `0.22/t` Hz if `si` is small compared to `t`.

The graph was made with the following script. You can adjust the frequency response by changing the `np%` parameter. The frequency axis is shown as a fraction of the sampling rate.

```
const np% := 100;       'Data points used for smoothing
const bins% := 4097;    'size of the response array
var coef[np%];          'array to use to simulate the smooth filter
arrconst(coef, 1.0/np%);'Set coefficients all the same
var rv%;
rv% := SetResult(bins%, 0.5/(bins%-1), 0, "Frequency response",
       "fraction", "dB", "Sample rate", "Response");
FIRResponse([], coef);  'calculate the response
DrawMode(1,2);          'draw as line
YRange(1,-40,0);        'set y range in dB
XAxisAttrib(1);         'Log the x axis
XRange(0.0001, 0.5);    'set a sensible range
WindowVisible(1);       'make it visible
```

Linear Predict

The Analysis menu Linear Predict... command is enabled in Time views with suitable channels and in Result views and is also available when you right-click on a suitable channel in a Time or Result view. Linear prediction replaces a (short) section of a waveform with data that is based on a forward linear prediction of the previous data points and a backwards linear prediction from the following data points. This can be used to repair data that has been damaged due to an artifact of some kind. You can find more information about Linear Prediction in the documentation for the `LinPred()` script command.



The command opens a dialog that allows you to select a channel or channels to process, the time range to replace and the previous and following data to use to generate the replacement. The script equivalent of this command is `ChanLinPred()`. The dialog fields are:

Channel(s)

Select a Waveform or RealWave channel to process or type in a list of channels. The channels must be writeable, so you cannot select virtual channels or disk-based channels in Read-only files. If you select a disk-based channel, any change you make is undoable to allow you to try out the command and undo the result if you do not like it. Changes made to memory channels are not undoable.

You can use this command to fill (short) gaps in memory channels. Due to the way that disk-based Waveform and RealWave channels are stored, you cannot use this command to fill a gap in a disk-based channel. Instead, create a memory buffer copy, fill the gaps, then save the memory buffer as a disk-based channel. You can use this command to replace NaN values in a RealWave disk-based channel.

Data Ranges

Normally you will predict both forward and backward to replace damaged data. We also allow you to use only the forward or backwards predictions (perhaps to fill in a gap at the start or end of a memory channel). You can choose from: Forward & Backward, Forward only and Backward only.

The following fields will usually hold cursor-related values. When the dialog opens, and any time that you add, delete or renumber vertical cursors, the dialog will reassess the available cursors and may change the ranges. If there are 4 cursors in the visible area, they will be assigned to the fields in position order. If there are fewer than 4 cursors in the visible area, all available cursors in the view are assigned in position order. If there are fewer than 4 cursors in the view, other positions (`XLow()` and `XHigh()`) are used and the results may be less useful.

Replace from, Replace upto

These two fields define the area of data to be replaced. The Linear prediction algorithm pays no attention whatsoever to the data in this range. Time view disk-based channels must have contiguous (gap free) data in this time range; this is not an issue for memory channels. There is an (arbitrary) limit of 1024 points on the number of points that we allow this algorithm to replace. If you attempt to replace more than this number, no change is made to the data.

There is a practical limit to the number of generated points; each generated point makes a contribution to the following points, so if the implied linear filter is unstable, after sufficient points the result become nonsense. This is particularly obvious if you try to replace a section of a linear ramp, which starts to go very wrong around 200 generated points.

Forward data start, Backward data end

The forward prediction data runs from the Forward data start position up to (but not including) the Replace from position. The backward prediction data runs from the Replace upto position up to (but not including) the Backward data end position. To be used for prediction, the data must be contiguous (hold no gaps). If there is a gap in the forward data, only the contiguous data backwards from the Replace from time is used. If there is a gap in the backwards prediction data, only the data from the Replace upto time up to the gap is used.

The forward and backward data generate two predictions, one forwards and one backwards. These are merged together by weighting the contribution of each point based on the relative number of forward and backward points and the position in the replace area. The weight given to each forward predicted point falls as the position increases, the weight given to each backwards predicted points increases as the position increases.

Apply
This button is enabled if the positions are set in ascending order and all selected channels have between 1 and 1024 replacement points and the selected training data is contiguous with the replacement area. Clicking the button will attempt to replace the selected data for the current channel list. You can undo changes made to Time view disk-based and Result view channels; changes made to memory channels do not record. This action records as the `ChanLinPred()` script command if recording is enabled.

Close
This closes the dialog.

1..4
Click this button to hide all cursors and create cursors 1, 2, 3 and 4 and position them equally spaced across the visible data area of the current Time or Result view. This action records as the `CursorSet(4)` script command if recording is enabled.

Refresh X
This rebuilds the X position fields to reflect the current cursor positions. You might wish to use this if you drag or fetch cursors so that the cursor order changes.

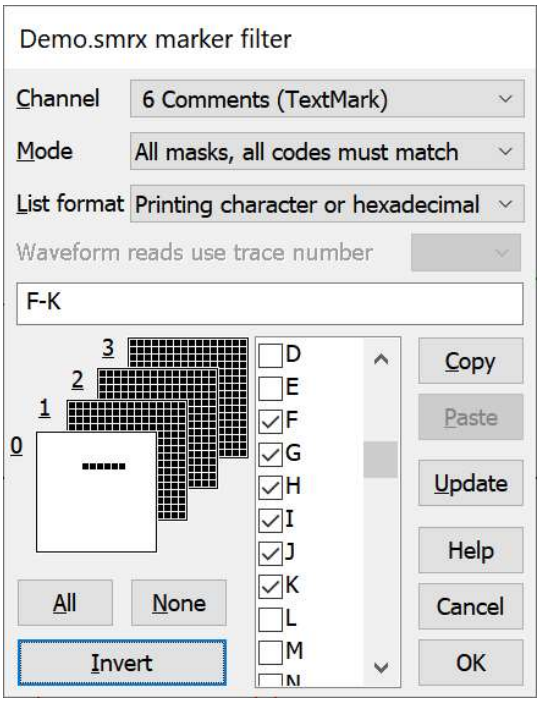
Please read...
Although this command can sometimes achieve seemingly magical results, all it is really doing is continuing a sequence of numbers while maintaining the same data statistics (auto-correlation and power spectrum) as the training data used to generate the generate the forwards and backwards predictions.

There is some skill in choosing the length of data to generate the predictions. If the source data is clearly cyclical (for example a blood pressure signal or an ECG), choosing an integral number of cycles before and after the replacement area will generally give reasonable results. The more cycles you choose, the more 'smoothed' the prediction will be, which may not be what you want if there is a trend in the cycle length as the position increases. In general, there should be more training data than the replaced data.

If the data is not obviously cyclical, the replaced data will have an auto-correlation (and power spectrum) that is similar to the training data, so should blend into the signal, which may be better visually than replacing the artifact with a straight line or a fitted curve.

Please remember that this does not tell you what the real data was... it only tells you what the data might have been if it continued the training data in a sensible way. It can be very useful when cleaning up data for visual inspection or fixing occasional glitches that prevent easy automated analysis, but taking measurements from predicted data is at best suspect and likely bogus.

Marker Filter



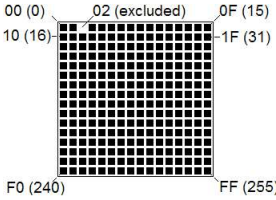
You can filter any channel that holds marker, WaveMark, TextMark or RealMark data. Each of these channels has a *marker filter* that selects the data items to display and use in calculations. Each marker data item has four marker codes that are matched against the marker filter. If you set a marker filter so that it would not pass all data, the channel number is displayed in red.

You can specify a marker filter using text (and in other places, for example in the Gate Settings dialog) or by selecting the current layer mask and checking the codes in the check box list.

Text version of filter

The dialog displays the text version of the filter above the 4 filter masks. You can edit the text version and when it is legal, the filter is applied to the masks and the check box list. If you make a change to the check box list, the text updates to match. If you edit the text to an illegal state, an error message appears at the bottom of the dialog.

Masks



The dialog shows the marker filter as four masks numbered 0 to 3. Each mask has 256 elements in a 16x16 grid, one for each possible code value. The contents of the front-most mask are displayed in the scrolling list in the centre of the dialog. Click on a mask element to bring the mask to the front and scroll the list to the element. If you edit the text version of the filter, the first mask changed by the edit comes to the front of the stack.

The top row of each mask represents code values 0 to 15 (hexadecimal codes 00 to 0F), the second row 16 to 31 (10 to 1F) and so on down to the bottom row, which represents values 240 to 255 (F0 to FF). If a code value is included in the filter, the corresponding element is black. When values are excluded, the element is white. The mask gives a quick indication of the state of the filter. There are three buttons that act on the entire mask:

All	This includes all code values in the filter (checks all the list boxes)
None	This excludes all code values from the filter (clears all the list check boxes)
Invert	This excludes included values and includes excluded values

Channel

This field is the standard channel selector. You can select marker-based channels only. If you change the channel selection the filter description held by the dialog updates to match the new channel. If you change the filter by editing the text or the check box list, changes are held in the dialog; use the `Update` button to apply the changes to the current channel.

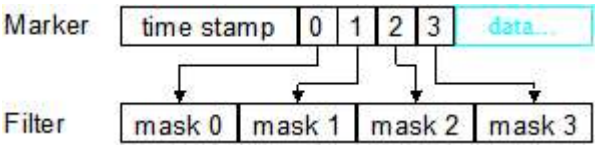
Copy/Paste

The **Copy** button copies the current filter held by the dialog, as text, to the clipboard. The Paste button is enabled if the clipboard holds a valid text description of a filer. Click the button to apply the clipboard description to the dialog; this does not apply the filter to the channel.

List format

The list shows each the code value as two hexadecimal or one to three decimal digits, the single character equivalent or as a combination as set by the *List format* field. Characters are appropriate for keyboard markers, numeric digits are used for the non-printing codes or can be forced for all codes. If you type a character, the list scrolls to the entry that starts with the typed character. To include marker codes, check the boxes. There are two modes in which to use the marker filter:

Mode 0 (AND): All masks, all codes must match



For a data item with marker codes *a*, *b*, *c* and *d* to be included, mask 0 must have code *a* checked, mask 1 must have code *b* checked, mask 2 must have code *c* checked and mask 3 must have code *d* checked. Most users of this mode set mask layers 1, 2 and 3 to All and use the first layer to select data values. You can think of this as the AND mode; to accept data marker code 0 must be in the mask 0 AND marker code 1 must be in mask 1 AND marker code 2 must be in mask 2 AND marker code 3 must be in mask 3.

Mode 1 (OR): One mask, any code can match

Only mask 0 is used, the rest are greyed out. For a data item with marker codes *a*, *b*, *c* and *d* to be included, mask 0 must have one or more of the codes *a*, *b*, *c* and *d* checked. You can think of this as the OR mode; to accept data marker code 0 OR marker code 1 `OR` marker code 2 OR marker code 3 must be in mask 0. There is one exception; for marker codes 2 to 4, the code 00 is ignored. To accept code 00, it must be the first marker code.

Mode 1 is often used when sorting spike shapes (WaveMark data) and you discover a WaveMark that is the result of a collision between two spikes. You can set the first marker code to the code for the first spike and the second to the code for the second (leaving the third and forth codes as 00), then the spike will appear on screen and in analyses when either of the codes are included in the mask.

Waveform reads use trace number

If the Marker channel holds multi-trace WaveMark data, you can also choose which trace is used when the channel is used as if it were a waveform. You can choose to specify traces 0 to 3 for tetrode data and traces 0-1 for stereotrode data. You can also choose Unset, which will use the first trace (trace 0). The script language equivalent of this is the `MarkTrace()` command. Also see the Draw Mode dialog Single trace option. In a virtual channel, you can use the `Ch(n, t)` command to select a trace. If the channel does not hold multi-trace data (as in the example image, above), this field is grey.

Cancel, Update, Help, OK

The buttons on the right are:

Copy	Copy the current dialog filter to the clipboard, as text.
Paste	Set the dialog filter state to match the text filter held on the clipboard.
Update	Update the channel display to correspond with the new filter
Help	Display this information
Cancel	Close dialog, cancelling changes since last update
OK	Close dialog, accept new filter

Specify Marker Filter as text

In addition to setting a Marker Filter interactively, from version [10] it is also possible to specify it using a text string in the Gate settings dialog and from the Sampling Configuration dialog Mode tab. The `ProcessGate()` script command and from [10.06] the `SampleTrigger()` and `MarkMask()` script commands can specify an entire marker filter as text. See the Marker Filter dialog for a description of the filtering process.

Filter specifications can be very simple, for example:

```
01
```

specifies a filter that matches code 01 in the first mask and anything in the other three. They can also be more complicated, for example:

```
00-08|[a,e,i,o,u]|##|!a-z
```

specifies a filter that matches codes 00 to 08 in the first mask, any lower case vowel in the second, anything in the third and anything except lower case letters in the last layer.

Set a filter in AND mode (all codes must match)
The basic syntax for a marker filter in AND mode is:

```
mask{|mask{|mask{|mask}}}}
```

where items in curly braces are optional. The text `mask` stands for the specification of a single mask and the vertical bar separates masks. A mask that is omitted on the right is assumed to match all possible codes.

Set a filter in OR mode (one code, any code can match)
The basic syntax in OR mode is:

```
|mask
```

In this case, there can only be one vertical bar and it must be the first character to signal OR mode. If you want to set AND mode and accept anything for the first code and set a mask for the second, use the syntax:

```
##|mask
```

Defining a mask

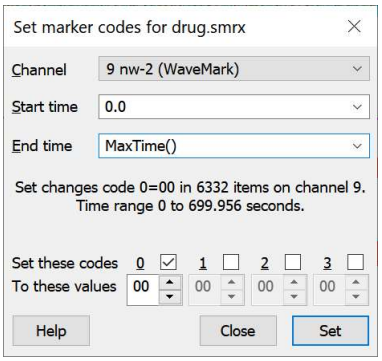
A mask is a text shorthand to describe the list of marker codes that match. The following are acceptable ways to describe a mask (only items marked with a * are compatible with Spike2 versions before [10.00]:

`A`	*A single ASCII character stands for itself. Any printing character can be used (including space), so spaces are significant in Marker Filter specifications.
`0f`	*Two hexadecimal digits (0-9, a-f or A-F) stand for a single code in hexadecimal.
`A-F`	A range of ASCII characters. The range must be in ascending order.
`00-3F`	A range of hexadecimal codes. The range must be in ascending order.
`00-a`	You can define a range with a mix of ASCII and hexadecimal, but the range must be in ascending order.
`##`	Shorthand for 00-FF, which means match anything.
`[1,3,07,a-f]`	A list of the items previously specified in this table. The comma character is used as a separator. You can include a comma in the list by placing it in a position where it cannot be interpreted as a separator: `[,,a]` means match a comma or `a`. You can use a list to match a character that would otherwise have a special meaning, for example | which is not treated as a mask separator inside a list.

You can also use `!` to generate a range that matches anything not matched by what follows the `!`, for example:

`!A`	Match anything except the ASCII character that follows.
`!0f`	Match anything except the hexadecimal code that follows.
`!A-F`	Match anything except the range that follows.
`!00-3F`	Match hexadecimal codes `40-FF`.
`!##`	Match nothing. This is legal, but of no use.
`![1,3,07,a-f]`	Match anything that is not in the list.
`[!,|]`	Match `!` and `|` as ASCII characters.

Set Marker Codes



You can open this dialog from the Analysis menu, by right clicking on a marker channel and from the Edit WaveMark dialog. From here you set selected marker codes for marker, WaveMark, TextMark and RealMark channels. Each data item has four marker codes that can be used to filter the data in the channel. Usually only the first marker code is used, see the Marker Filter command for more information. The Channel field sets the channel to process and the Start time and End time fields identify the markers to be set. If a marker filter is set for the channel, only data items that are in the filter are changed by this command. The four check boxes 0 to 3 select the codes to change. You set the value for each code as either two hexadecimal digits or one printing character.

Example

To change all the marker items in a channel with a first marker code of `02` to have a first marker code of `1A` do the following:

1. Open the marker filter dialog and set the channel to display only marker code 02.
2. Open this dialog, select the channel and set the time range as 0.0 to Maxtime().
3. Check the `1` box to set the first marker code and edit the text under the check box to 1A. Make sure the other check boxes are clear.
4. Read the text in the centre of the dialog as a check that this is the action you wish to take. This action cannot be undone.
5. Click the Set button to change the marker codes.

All the code `02` items will disappear as they have been coded as `1A`, so you will need to change the marker filter settings if you want to see the result.

The first time you open this dialog in a Spike2 session, the message area has the additional text:

```
(Use Ctrl+Alt+Drag to select spikes with a line.)
```

To remind you that you can select spikes with the mouse.

Selecting WaveMark data with the mouse

Whenever WaveMark data is displayed as a waveform, you can select one or more events using the mouse and then change the marker codes with a dialog very similar to the one displayed above except that the Channel, Start time and End time fields are omitted as the selection of events is done by line crossings. To select the events, hold down `Ctrl+Alt`, then click and drag a line to intersect the WaveMark items that you want to select. If the channel has multiple traces you can use the Single trace option of the Draw Mode dialog to show a single trace.

New WaveMark

This creates WaveMark data channels from waveform data or existing WaveMark data. You can also activate this option by right-clicking on a waveform or WaveMark channel and selecting the New WaveMark command from the context menu.

New Stereotrode, Tetrode

Use this to create a new WaveMark channel with 2 or 4 traces. To enable this option, select 2 or 4 waveform channels with the same sample rate. This option opens the New WaveMark dialog with the selected channels as the source (see Spike sorting for details). The channels are used in the order that they are selected. The first selected channel sets the sample rate that the selected channels must match.

If the selected channels have the same units but different scales, we will scale the data to match the channel with the largest scale factor. If the difference in scales between the channels exceeds a factor of 20 you are

warned and offered the chance to abandon the operation. You can edit channel scaling in the Channel Information dialog.

If the selected channels have different units, we display a warning dialog. If you choose to continue we pretend that all the channels have the same scales and units, being the values for the first channel in the list. You can edit the units for a channel in the Channel Information dialog.

Edit WaveMark

This Analysis menu option reclassifies WaveMark data both manually and automatically. You can also activate this option by right-clicking on a WaveMark channel and selecting the Edit WaveMark command from the context menu.

Digital filters

This displays a pop-up menu in which you can choose the FIR Digital filtering dialog or the IIR Digital filtering dialog. From these dialogs you can create digital filters and apply them to waveform or RealWave channels. You can also open these dialogs by right clicking on a suitable channel and selecting a digital filter option from the context menu.

10: Window menu

Window menu

The Window menu has permanently present commands plus a list of up to 9 windows that belong to the Spike2 application. If you select one of the windows in the list, the window is brought to the front and made the current window. If there is more than 9 windows you can use the Windows... menu command to choose from all possible windows.

Duplicate window

This command is only available within a time window and creates a duplicate window with all the attributes (list of displayed channels, event display modes, colours, cursors and size) of the original window. Once you have created the new window, it is independent of the original. However, the data channels within it are the same data channels as in the original window, so any changes made to the data in one window will cause all duplicated windows to update. Duplicating a window allows you to have different views of the same data file with different scales and drawing modes and different sets of channels.

You can close all windows associated with a data document using the control key plus close. This will remember the position and state of all windows associated with the document.

Hide

This command makes a window invisible. This is often used with script windows and sometimes is used to hide data windows during sampling when only the result views are required. Closing the Log window is equivalent to hiding it as the Log window always exists.

Show

This option lists all hidden windows in a pop-up menu. Select a window from the list to make it visible.

Window Title

The data document maintains both a file name and a window title. If the file has been saved, the window title is usually set to the name of the file (without the file path). When you open a new window, the title is set to the type of the window followed by a sequence number, for example Data2 or Grid23. The window title is mainly cosmetic, but it is used to suggest a file name when you save an unsaved document.

You can change the window title with the script `WindowTitle$()` command, and also from the Window Title dialog. The dialog is available from the Window menu and also from the context menu that appears when you right-click on the window title.

Title
This field shows the original title when the dialog opens and you can edit it to a new title.

OK
This button is enabled if you make a change to the text. Click the button to apply the change and close the dialog. Note that changes made to a title are not undoable (but see Reset).

Cancel
Close the dialog without making any change to the title.

Reset
This sets the title back to the default window title. If the associated document has been saved, the default title is the file name. Otherwise, the default title is the type of the window (Data, Text, Grid...) plus a sequence number.

Tile Horizontally

This command arranges all the visible screen windows so that no window is overlapped by any other. Iconized windows are arranged along the bottom edge of the window. The command attempts to arrange window so that they are wider than they are high. Tiling takes into account the space used for title bars of iconized windows, so to use the full application window area you should hide any iconized windows first. This command may have the same result as Tile Vertically, depending on the number of windows.

Tile Vertically

This command arranges all the visible screen windows so that no window is overlapped by any other. Iconized windows are arranged along the bottom edge of the window. The command attempts to arrange windows so that they are taller than they are wide. Tiling takes into account the space used for title bars of iconized windows, so to use the full application window area you should hide any iconized windows first.

Cascade

All windows are set to a standard size and are overlaid with their title bars visible. Any iconized windows are left in the iconized state, and they are arranged along the bottom edge of the window, as for the Arrange Icons option.

Arrange Icons

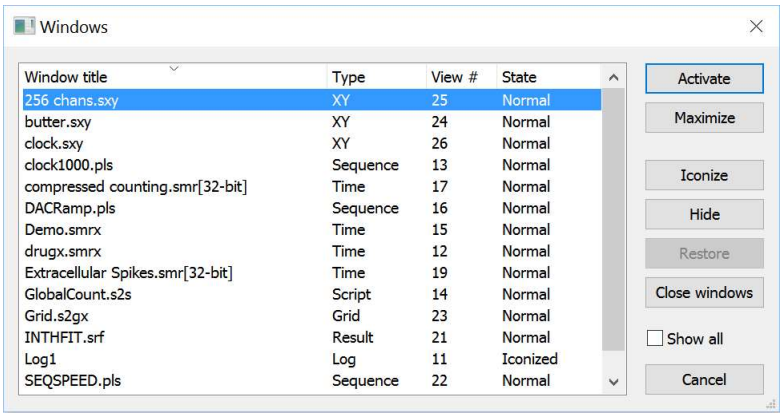
You can use this Window menu command to tidy up the windows that you have iconized in Spike2. The icons are lined up along the bottom edge of the application window.

Close All, Close All and Link

This command closes all windows in the Spike2 application. You are asked if you want to save the contents of any text windows that have changed or any newly sampled data window. You can avoid being asked if you want to save modified result and XY windows with an option in the Edit menu Preferences.

If you hold down the `Ctrl` key before clicking on the Menu, this option becomes Close All and Link. In this case, time views will also save sufficient information about attached result and XY views to restore them when the time view is opened. This extra information is saved in the resource file associated with the time view (and this can be a lot of information). This is equivalent to the File menu Close and Link command applied to all open files.

Windows



This dialog lists all the document-related windows that are open and lets you apply common window operations to one or more of the windows. You can sort the list based on the window title, type, view number (as seen by the script language) and window state by clicking the title bar at the top of the list.

The operations available depend on the type and numbers of the selected windows. For example, the Activate and Maximize buttons are only enabled when a single view of a suitable type is selected.

Show all

Normally, the only windows displayed are for windows that own documents (script, output sequencer, and text views, the log view, time views, result views and XY views). However, if you check the Show all box, the window lists all windows that have a view handle that the script language can manipulate.

11: Cursor menu

Cursor menu

The cursor menu creates and destroys vertical and horizontal cursors, changes the display to make them visible, changes their labelling mode and obtains the values of channels where they cross vertical cursors and between vertical cursors. Up to 10 vertical (numbered 0-9) and 9 horizontal (numbered 1-9) cursors can be active in each time or result window. Cursors in separate windows are independent of each other. When a window is duplicated, the cursors are also duplicated, but are then independent of the original view.

Vertical cursors

Vertical cursors mark positions in data, result and XY views. They allow interactive positioning by dragging with the mouse and in a time view they can be made active, which allows them to search for features in data. You can create up to 10 vertical cursors, numbered 0 to 9. If you feel you need more vertical cursors in a time view you could consider using vertical markers. From version [10.10] you can fix vertical cursors in position and prevent the user from deleting them. From version [10.14] you can use cursor positions in virtual channel expressions.

The following commands in the Cursor menu are used to control and interact with the vertical cursors. There are also vertical cursor context commands available by right-clicking on a vertical cursor. The vertical cursor commands available from the menu are:

New Cursor

This command and the  cursor button at the bottom left of time and result views adds a vertical cursor with the current label style and lowest available cursor number (1 to 9) to the window. The keyboard short cut `Ctrl+n` where `n` is a cursor number (including 0 in a time window) creates cursor `n` if it doesn't exist and moves it to the centre of the window. Each view has a default label mode for new cursors set by the Label mode command.

Delete

The delete command activates a pop-up menu from which you can delete one or all vertical cursors. As an aid to identification cursors are listed with their number and position. Deleting a cursor removes it from the window; other cursors are not affected. Deleting cursor 0 just hides it as cursor 0 always exists. To restore cursor 0 use the `Ctrl+0` key combination or the cursor menu Fetch command.

Fetch

This opens a pop-up menu in which you select a vertical cursor to place in the visible x axis. The keyboard short cut `Ctrl+n` where `n` is a cursor number will fetch cursor `n` if it exists or will create it if it doesn't exist. You can use this command to make cursor 0 visible if it is hidden.

Move To

This command opens a pop-up menu from which you can select a vertical cursor to move to. The cursors are listed with their number and position as an aid to identification. The window is scrolled to display the nominated cursor in the centre of the screen, or as close to the centre as possible. This command does not change the x axis scaling. You can also use the keyboard short cut `Ctrl+Shift+n` where `n` is a cursor number.

If cursor 0 is hidden you can still move to it. If you want to make cursor 0 visible use the Fetch command or the `Ctrl+0` key combination.

Position Cursor

The Cursor menu Position Cursor command opens a pop-up menu to select a cursor and then opens a dialog in which you can type or select a cursor position. You can also activate this dialog by right clicking on a vertical cursor and selecting Set Position from the cursor context menu. You are expected to edit the value to a valid dialog x axis expression to set the cursor position. The field to the right of the edit box holds the x axis display mode, one of: s, ms, hms or tod.

Time of Day x axis mode (tod)

When this dialog is used in a time view with the x axis in Time of Day mode, the value displays in a format to match the x axis followed by `tod`. For example:

```
12:00:00tod
```

This allows you to enter values that match the displayed axis and not as a time offset from the start of the file. If you omit the `tod`, the value you set is in seconds from the start of the file. To move the cursor 1 second to the right in the above example you could edit the value to (assuming it is cursor 3):

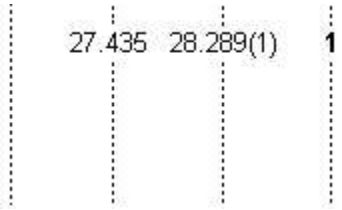
`12:00:01tod` or `12:00:00tod+1` or `C3+1`

Display all

This command has no effect if there are no cursors. With a single cursor, the command behaves as though you had used the Move To command to select it. With multiple cursors, the window is scrolled and scaled such that the earliest cursor is close to the left-hand edge of the window and the latest is close to the right-hand edge (the extra space is 5% of the displayed window unless the cursors are very close together when the space is proportionally more).

If you hold down the `Ctrl` key and select this command, only the current set of active cursors are considered.

Label mode

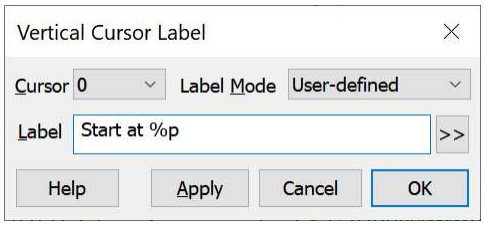


Each cursor has an optional label used to identify it. You can drag the cursor labels up and down the cursor with the mouse to suit the data. There are five cursor label modes: None, Position, Number, Position and Number, and User-defined. You select the most appropriate for your application using the pop-up menu or by right clicking on a cursor and choosing to set the cursor label from the context menu. To avoid confusion between the cursor number and the position, the number is displayed in bold type when it appears alone and bracketed with the position.

Each cursor stores its own mode and label, and the view has a mode that is applied to new cursors. The first four items in the menu set the view mode and the mode of all cursors. You can also set a user-defined label for cursors (but not for the view) with the Set Label command.

Cursor labels can have a `!` appended if they are at an invalid position due to a failed active cursor search. From [10.19] they can have `*` appended if an active cursor search failed and the user supplied a default position.

Set Label



You can open the cursor label dialog from the Label Mode and Horizontal Label Mode cursor menu commands or by right-clicking on a cursor and using the Set Label command in the context menu. The dialog for horizontal cursors differs only in the dialog title.

From this dialog you can set the cursor label mode for one or all cursors. The Cursor field can be set to the number of any cursor in the view, or All. If you choose All, any change applies to all cursors and sets the view mode except in User-defined mode, where the view mode does not change.

The Label Mode field lets you choose one of None, Position, Number, Position and Number, and User-defined as the cursor mode. If you select User-defined, the Label field appears together with the >> button and you can set a label of your choosing.

The OK and Apply buttons are enabled when there are unapplied changes. OK applies the changes and closes the dialog, Apply applies the changes and leaves the dialog open. Cancel closes the dialog.

User-defined labels

User-defined labels display the text you type, except that text sequences introduced by `%` are replaced:

`%p`	Replaced by the cursor position or cursor position difference
`%u`	Replaced by the position units as displayed on the axis
`%q`	Equivalent to `%p %u` (position followed by units)
`%n`	Replaced by the cursor number
`%v(n)`	Replaced by the channel `n` value at the cursor. Not valid for horizontal cursors or in an XY view.
`%w(n)`	The same as `%v(n)` but followed by the units of the value.

You can also stipulate the width (`W`) and the number of decimal places (`D`) used for the position and value by using `%W.Dp` and `%W.Dv(n)`. If you omit the `W.D` when displaying a position or position difference, the displayed precision is based on the current x axis. Channel values use a default format.

For example: `%n at %6.4p %u, %v(2)` might display: `1 at 2.2346 s, 87.128756` if cursor 1 was at 2.2346 seconds and channel 2 had the value 87.128756 at this point. The `%v(n)` option is not allowed for horizontal cursors or for vertical cursors in an XY view. The value returned by `%v(n)` is the same value as displayed on the axis for that cursor and channel.

The >> button pops up the list of replacements, and if you choose one, it replaces any selection in the Label field. If you choose the `%v(n)` option, you are prompted to select a channel to measure. If you select the cursor position difference option you are prompted for the cursor to use and the difference mode.

We allow you to type in quite long labels. Before Spike2 [7.11] labels longer than 19 characters were truncated when you closed a data file. However, very long labels look messy; it is usually better to keep labels short.

Position differences (from [10.08])

You can display the position (`p` and `q` options) relative to a different cursor rather than as the x or y axis position using `%<cur>-{W{.D}}p` and `%<cur>-{W.D}q`, where `<cur>` is the cursor number and items in curly braces can be omitted. You can choose one of three difference modes by changing the character after the cursor number:

`-`	Subtract the nominated cursor position, for example `%0-p` to show the position relative to cursor 0
`+`	Display the distance from the nominated cursor (always positive), for example: `%0+p`
`~`	Subtract the cursor position from the nominated cursor position, for example `%0~p`

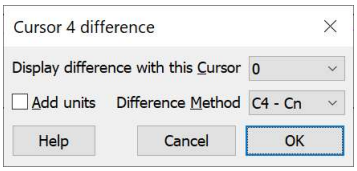
User-defined labels and drop down lists

If a cursor has a user-defined label, it is used as a tool tip when you select the cursor from a drop-down list. To try this, go to the Active cursor dialog. Set several vertical cursors with user-defined labels, select a search mode, then open one of the drop-down lists of x axis positions and hover the mouse over one of the Cursor(n) items in the list. If the cursor has a user-defined label, the label will appear as a tool tip.

Scaling of positions and units

If the associated axis is in an Auto units mode, the displayed label and units match the axis. This is a purely cosmetic effect and does not change the value used in the `Cursor()` or `HCursor()` script command.

Cursor difference dialog



This dialog generates the basic code required to add the difference between the position of the current cursor (displayed in the dialog title) and a different, nominated cursor. The same dialog is used for both vertical and horizontal cursors. However, for horizontal cursors, you can only display the difference with a horizontal cursor attached to the same y axis.

Display difference with this Cursor

This field chooses the cursor to use as the reference. If there is no suitable cursor this field holds None and the OK button is disabled.

Add units

Check the box to follow the measured value by the appropriate units.

Difference Method

There are three difference methods. The Method column is as displayed with vertical cursor 4 as the current cursor. The Cs are replaced with Hs for horizontal cursors.

Method	Code	Effect
C4 -Cn	-	Subtract the nominated cursor
|C4-Cn|	+	Absolute value of the difference
Cn-C4	~	Subtract this cursor from the nominated cursor

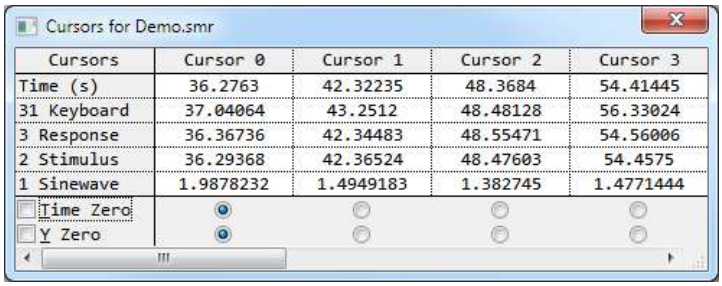
The Code is the character used to separate the cursor number from the rest of the generated label. For example, the dialog above would generate: `%0-p`

Renumber

When created, cursors take the lowest available cursor number. This command renumbers the vertical cursors 1 through 9 so that the lowest numbered cursor is on the left and the highest on the right. Renumbering cursors has no effect on vertical cursor 0. As currently implemented, all other cursor features (labels, active cursor mode) are unchanged by renumber, only the cursor number changes. Because of this, be cautious about using the renumber command with active cursors.

Cursor values

This command opens a new window containing the values at any cursors in the current time or result window. Cells for cursors that are absent, or for which there is no data, are blank. The new window is a top level window and will never go behind another time or result view.



In a result window, the value displayed is the value to be found in the bin to the right of the cursor. If you position the cursor at the far right (where there is no bin to the right), a blank is displayed as the value.

In a time window, the values displayed depend on the channel display mode. If a channel has a y axis the displayed value will be in y axis units. If the data is displayed as a continuous line or as a histogram, the displayed value is where the line crosses the cursor. If the data is displayed as dots, the value is the nearest dot for waveforms and the previous dot for instantaneous frequency. Positions more than one waveform sample interval away from any waveform point have no value.

If the channel has no y axis, in State drawing mode, the value is the state. Otherwise the channel holds an event or marker and the displayed value is the time of the next event or marker at or to the right of the cursor. If there is no data, the field is blank.

The Time zero check box enables relative cursor times. If checked, the cursor marked with the radio button is taken as the reference time, and the remaining cursor times are given relative to it. The reference cursor displays the absolute time, not 0. In the example above, cursor 1 has been set as the reference.

The Y zero check box enables relative cursor values. The radio buttons to the right of the check box select the reference cursor. The remaining channels display the difference between the values at the cursor and the values at the reference. The values for the reference cursor are not changed.

Selecting copying and printing data

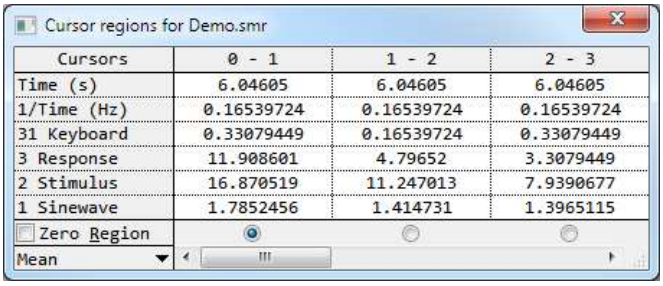
You can select areas of this window and copy them to the clipboard by clicking on them with the mouse. Hold down the Shift key for extended selections. You can select entire rows and columns by clicking in the cursor and channel title fields. Hold down the Control key to select non-contiguous rows and columns. Use `Ctrl+C` to copy selected data to the clipboard. Use `Ctrl+L` to copy selected data to the log view. Right click in the cursor window to display a context menu with command for Copy, Log, Font and Print.

Interaction with script short-cut keys

If this dialog (or the Cursor Regions dialog) has the input focus, user-defined short-cut keys for the script toolbar, interact bar and user-defined dialogs will work. The downside of this is that any keys that are grabbed by the script will no longer be available to the dialog. The main keys that are affected by this are Tab (to move to the next field) and Space (to select/unselect check boxes). As long as these keys are avoided as short-cut keys, users should not notice this change.

Cursor regions

This command opens a cursor region window for the current time or result view. This window calculates values for data regions between the cursors. One column can be designated the Zero region by checking the box and selecting the column with a radio button. The value in this column is then subtracted from the values in the other columns (except in Area(scaled) mode which scales the zero region value to allow for column widths).



The field at the bottom left sets the measurement method; click it for a full list. Cells with no cursors or data are blank.

The dialog fields are updated when cursors move or channel data changes. Calculation of values in this dialog can take a long time for large regions; you may wish to keep this dialog closed unless you are actively using it. In particular, processes that drive cursor 0 can become very slow if this dialog is open. You can interrupt long calculations in time windows with the `Ctrl+Break` key combination.

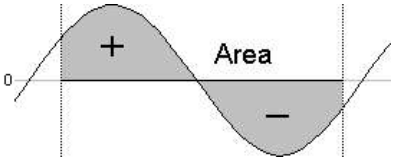
The effect of the selected measurement type depend on the view type and the channel type (time view waveform and event channels, result view channels) and, in some cases, the channel display mode.

The first time you open this dialog in a Spike2 session, it will be in Mean mode. Subsequent use of this dialog will open in the last mode that you set, as this is more likely to be the mode you want.

Waveform channels and result window data

In a time window, waveform channels include WaveMark channels drawn as waveforms, WaveMark or Cubic Spline and waveform and RealWave channels drawn as dots, Cubic spline or Sonogram. The region set by a pair of cursors is the data starting at the first cursor up to, but not including, the data at the second cursor. In a result window, the region set by a pair of cursors starts at the bin containing the left cursor and ends in the bin to the left of the bin containing the right cursor. The values for each mode are:

Area The area between the data points and the y axis zero. Area is positive for curve sections above zero and negative for sections below zero. Use Modulus if you want areas below the y axis to be treated as positive.



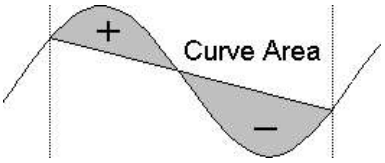
Mean The sum of all the data points between the cursors divided by the number of data points between the cursors. There is also a Mean Absolute value.

Slope The slope of the least squares best fit line to the data between the cursors.

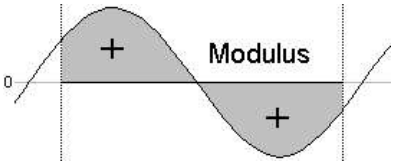
Sum The sum of the data values between the cursors. If there are no samples between the cursors the field is blank.

Area (scaled) The same as Area, but if a zero region is specified, the amount subtracted from the other regions is scaled by the relative width of the regions.

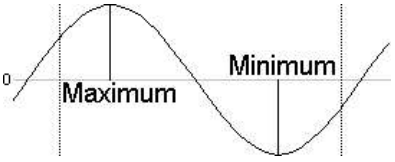
Curve area Each data point makes a contribution to the area of its amplitude above a line joining the end points multiplied by the x axis distance between the data points. The picture makes this clearer.



Modulus Each data point makes a contribution to the area of its absolute amplitude value multiplied by the x axis difference between data points. This is equivalent to rectifying the data, then measuring the area. If a zero region is specified, the amount subtracted from the other regions is scaled by the relative width of the regions.

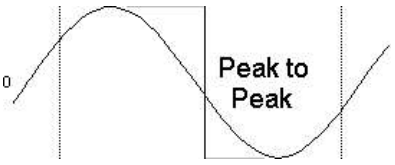


Maximum
Minimum For the maximum measurement, the value is the maximum value found in the time range between the cursors. For the minimum, the value is the minimum value in the time range.



Abs Max The value shown is the maximum absolute value found between the cursors. If the maximum value was +1, and the minimum value was -1.5, then this mode would display 1.5.

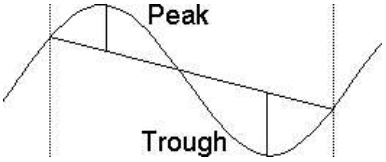
Peak to Peak The value shown is the difference between maximum and minimum values found between the cursors. The peak to peak value is always positive.



SD — The standard deviation from the mean of the values between the cursors. If there are no values between the cursors the field is blank. If there are `n` data points, and the sum of the squares of the differences between the points and the mean value is `DiffSquare`, the result is calculates as `Sqrt(DiffSquare/(n-1))`.

RMS — The value shown is the RMS (Root Mean Square) of the values between the cursors. This is calculated by summing the squares of the values in the range, dividing the sum by the number of values and taking the square root. If there are no values between the cursors the field is blank.

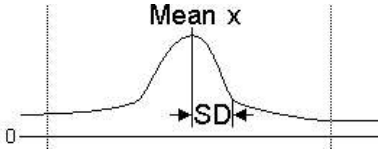
Peak — The value shown is the maximum value found between the cursors measured relative to a baseline formed by joining the two points where the cursors cross the data. This is always greater than or equal to 0.



Trough — The value shown is the minimum value found between the cursors measured relative to a baseline formed by joining the two points where the cursors cross the data. This is always less than or equal to 0.

RMS error — This is the same as the SD except that the normalising factor is `n`, not `n-1`.

Mean in X — This works for waveforms and result views. The value is the mean x value weighted by the data at each point. Although this would normally be used with positive data, it has some meaning with negative data values.



SD in X — This is available for result views only. The result has a useful meaning only when all the data values in the time range are positive. The value is: `Sqrt(Sum(data[x]*(x-xm)^2)/Sum(data[x]))` where the sum is over the `x` positions in the range and `data[x]` is the data value at `x` and `xm` is the value returned by Mean in X.

Mean Abs — This is the mean of the absolute values of the data values in the time range.

SEM — The Standard Error of the Mean of the values. If you measured the mean, then repeated the data capture and measurement under identical circumstance, this is a measure of how different you would expect the mean to be, assuming normally distributed noise. Do not confuse this with the SEM values that are available in result views formed by averaging with error bars enabled.

Median — This is the value for which there are the same number of values less than the value and greater than the value. Medians can be more robust than the mean when the data has outlier points. The Median is slower to compute than the mean and if you attempt to find the median of a huge number of points, Spike2 may give up due to exhausting memory. The Median is not calculated for event channels, regardless of their drawing mode.

Median size — This finds the median of the absolute values of the difference between each value and the median of the range. It is a robust measure of the amplitude variation of the signal in that large signal outliers do not affect it. This is not calculated for event channels, regardless of their drawing mode. This takes twice as long to compute as the Median.

Event channels

Event channels drawn in mean and instantaneous frequency mode are treated the same as waveforms. For events drawn in other modes, all measurement types except those listed below produce a blank field in the window. There is an Edit menu Preferences option in the Compatibility tab to force the old behaviour, in which only the following measurement types produced a result:

Mean — The count of events or markers between the cursors divided by the time difference between the cursors.

Area — The total number of events or markers on the channel between the cursors.

Area (scaled) — The same as Area, but if a zero region is specified, the amount subtracted from the other regions is scaled by the relative width of the regions.

Sum — The same as Area.

The Median is not implemented for event channels, regardless of the drawing mode.

Selecting, copying and printing

Fields from this window can be copied to the clipboard or directly to the Log window. To do this, select the region to be copied and then click the right mouse in the window and use the Copy or Log command. To select an entire row or column, click the mouse in the titles at the top and left of the window. To extend a selection, hold down the Shift key. To select non-contiguous rows or columns hold down the control key and select the rows and columns as required. You can also print the entire window with Ctrl+P and set the font with Ctrl+F.

Horizontal cursors

In many respects, horizontal cursors are similar to vertical cursors. However, they differ significantly in that each horizontal cursor is attached to a channel, or more accurately, to the y axis of a channel. If you change the drawing mode of an event channel with a horizontal cursor to a mode that has no y axis, the horizontal cursor will treat the available vertical space as if it runs from 0 at the bottom to 1 at the top.

If you drag channels with horizontal cursors on top of each other, only the horizontal cursor for the topmost channels (i.e. the channel with a valid y axis) is visible. In the special case of channels drawn with a locked y axis and a group offset of 0, all horizontal cursors for all channels are visible because the y axis is valid for all the channels.

Horizontal cursors positions can be used as part of Dialog expressions and from version [10.14], in virtual channel expressions.

The following commands in the Cursor menu let you control and interact with the horizontal cursors. There are also horizontal cursor context commands available by right-clicking on a horizontal cursors.

New Horizontal

The command is available when a time or result window is the current window, and there are less than 9 horizontal cursors already active. A new horizontal cursor is added to the first (lowest) visible channel with a y axis. If the first channel is part of a group, the cursor is added to the channel that is the head of the group. The cursor is given the lowest available horizontal cursor number and is labelled with the cursor label style for the window. If the x axis scroll bar is visible you can also activate this command by clicking the small button at the left of the x axis scroll bar.

You may find it more convenient to use the Alt+1 through Alt+9 keyboard short cuts to activate horizontal cursors.

Delete Horizontal

The delete command activates a pop-up menu from which you can select a horizontal cursor to remove, or you can delete all the horizontal cursors. The cursors are listed with their number, channel number and position as an aid to identification. Deleting a cursor removes it from the window; other cursors are not affected.

Fetch Horizontal

This activates a drop down list from which you can select an existing horizontal cursor to move it to the first visible channel that has a Y axis. The vertical position in the channel is determined by the cursor number. It is usually much easier to use the keyboard short cuts Alt+1 through Alt+9 to add a specific horizontal cursor to the first suitable channel.

If a horizontal cursors is hidden (this can happen if you use a script), fetching the cursor will make it visible.

Move To Horizontal

This command opens a pop-up menu from which you can select a horizontal cursor to move to. The cursors are listed with their number and position as an aid to identification. The window is scrolled vertically to display the nominated cursor in the centre of the y axis range.

Position Horizontal

This command opens a pop-up menu in which you can choose a horizontal cursor and then opens a dialog in which you can set the position and channel for the cursor. You can also open the dialog by right clicking on a horizontal cursor and selecting Set Position from the cursor context menu.

Display all Horizontal

This has no effect if there are no horizontal cursors. If there is a single cursor, the command behaves as though you had used the Move To Horizontal command and selected it. When there are multiple cursors, the channel y axis range is adjusted such that the lowest cursor is at the bottom edge of the channel area and the highest is at the top edge.

Horizontal Label mode

Each cursor has an optional label used to identify it. You can drag the cursor labels to the left and right with the mouse to suit the data. There are five cursor label modes: None, Position, Number, Position and Number, and User-defined. You select the most appropriate for your application using the pop-up menu or by right clicking on a cursor and choosing to set the cursor label from the context menu. To avoid confusion between the cursor number and the position, the number is displayed in bold type when it appears alone and bracketed with the position.

Each cursor stores its own mode and label, and the view has a mode that is applied to new cursors. The first four items in the menu set the view mode and the mode of all cursors. You can also set a user-defined label for cursors (but not for the view) with the Set Label command (which is the same as for the Vertical cursor).

Renumber Horizontal

When you create a horizontal cursor, they take the lowest available cursor number and you can also drag cursors over each other. In some circumstances you want the cursors numbered in order on the screen, for example, when taking the differences of cursor values. This command renumbers the cursors, with cursor 1 at the bottom. Where cursors share the same channel, they are ordered so that the lower numbered cursor has the lower y axis value.

Active cursors

In a time view, vertical cursors can be *active* or *static*. An active cursor can seek to a position based on the position of other cursors and data in a channel. Active cursors can automate data analysis, leading to new data channels, XY trend plots or tabulated output. The active cursor state is controlled with the Active Cursors dialog or with the `CursorActive()` script command.

Cursor 0

Vertical cursor 0 is special. It always exists and cannot be deleted, but it can be hidden. It is the iterator for XY view and data channel measurements; a movement of cursor 0 causes all other active cursors to recalculate their positions. This calculation is done in order of rising cursor number. Cursor 1-9 recalculate their position if

cursor 0 moves. To hide cursor 0, right click on it and select Hide cursor 0 in the context menu. Script users can cause a nominated range of cursors to search with the `CursorSearch()` command.

You can also make horizontal cursors active so that they move when a vertical cursor on which they depend moves.

Note that cursor 0 can be driven by Spike shape dialogs, by Multimedia replay, Offline Waveform Output, by user dragging and by Measurement processing. The Spike shape dialogs and Multimedia replay windows can be set to track cursor 0, so changes made by active cursor searches can affect them. There is addition control over active cursor searches from the script language.

Valid and invalid cursors

Active cursor positions are either *valid* or *invalid*; invalid cursors display an exclamation mark (`!`) at the end of the label. A position is invalid if a search fails and the Position if search fails field is empty or does not contain a valid expression. From [10.19] the label has an asterisk (`*`) appended if the position was set by the Position if search fails field. Expressions that use positions of invalid cursor are also invalid. The measurement system rejects invalid positions. Cursor positions are validated by operations that move them to a specific place such as dragging. Script users can test if a cursor has a valid position with the `CursorValid()` command.

Cursor search order

Cursor searches are done in ascending order of cursor numbers. Usually, cursor 0 is moved (or told to search), triggering a search of active cursors 1 to 9 (but a script can tell any range of cursors to search). Search ranges can be set in terms of other cursor positions. Such ranges are evaluated when the cursor searches. When cursor 3 searches, any expression used in the search that refers to cursors 0 to 2 will get the new position of these cursors, references to cursors 4 to 9 will get the position before they search.

Previous cursor position [8.09]

Each time a cursor moves we save the old position. You can access this with the `CnX` and `CursorX(n)` dialog expressions and the `CursorX()` script command. This can be useful with iterative searches and also to position a cursor the same distance from cursor 0 (for example) as it was in the previous search. You can use the `HCnX` and `HCursorX(n)` dialog expressions and `HCursorX()` script command with horizontal cursors.

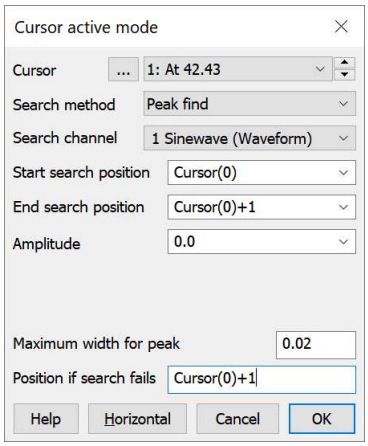
Cursor positions after a failed search

Cursor 0 does not move after a failed search. The remaining cursors are set to the result of evaluating the Position if search fails field. If this field is empty, or if the expression fails to evaluate, the cursor is set to the mid-point of the search region (which may be inconveniently far away). A useful trick is to position a failed search cursor at the same relative position to cursor 0 as for the previous search, for example by setting the Position if search fails for cursor 1 to `C0-C0X+C1`. This takes advantage of the fact the at the time of the cursor 1 search, `C1` is the old cursor 1 position, and `C0-C0X` is the distance that cursor 0 moved.

Warning

Remember that active cursors are set in position in the order of their cursor number. If a cursor search fails, say for cursor 3, then setting the Position if search fails to an expression involving cursors 0-2 will be the new positions (as they will already have searched), but any reference to cursors 4 to 9 will be to the old position.

Active mode



This command opens the Cursor mode dialog (script equivalent `CursorActive()`). The Search method field and the selected cursor determine the dialog contents. An active cursor has an associated Search channel and start and end positions that define the data to search to locate the new cursor position.

Cursor 0 does not have start and end positions. However, it does have a Minimum step; searches start at this distance from the cursor 0 position and continue to the file end for a forward search, or to the file start file for a backward search. Before [8.07], setting a minimum step of 0 could cause searches to become stuck, returning the same position over and over. From [8.07], a search will always return a new position unless it fails.

Cursor 0 has a restricted range of search methods: Peak find, Trough find, Rising threshold, Falling threshold, Within thresholds, Outside thresholds, Peak slope, Trough slope, Turning point, Data points, Expression, Gap start and Gap end.

The search positions can be a fixed time, but more usually they will be expressions that involve the positions of other active cursors. Active cursor positions are evaluated in sequence from cursor 0 to cursor 9. For cursor `n`, an expression that refers to an active cursor less than `n` uses the new position. An expression that refers to a cursor greater than or equal to `n` uses the old position.

If the End position for search is less than the Start position for search, searches are backwards. If a search is backwards, read *previous* for *next* and *last* for *first* in the descriptions of the cursor modes. Where search modes depend on data values, for example maximum and minimum, and the data channel does not have a y axis, then the value is taken as the interval between data items.

Several modes use a slope. These modes have the value Width for slope measurement, in seconds and use the data points from Width/2 before the current position to Width/2 after to calculate the slope unless there are more than 200 points, in which case 100 points before and after are used. The contribution of each point to the slope is proportional to the distance of the point from the current position. Because the slope at any point requires data around it, the slope within Width/2 of the ends of the data and any gaps is not to be relied on as the missing data is assumed to be zero.

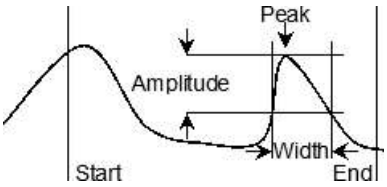
The **...** button next to the cursor number opens the Cursor Label dialog where you can set a user-defined label (sometime useful to make it clear which cursor is in use).

The active cursor modes are:

Static

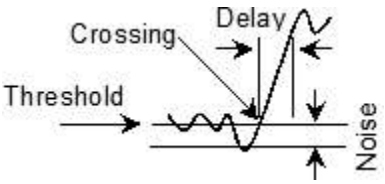
A new cursor starts out in Static mode. It stays where you put it and is not changed by a change in the position of a lower numbered cursor. The cursor position is always valid.

Peak find, Trough find



The Amplitude field defines how much the data must rise before a peak and fall after it (or fall before a trough and rise after it), to detect a peak/trough. The Maximum width for peak field rejects peaks that are too broad (set it 0 for no width restriction). For waveform channels, the peak position is located by fitting a parabola through the highest point and the points on either side. In the diagram, which shows a peak search, the first peak is not detected because the data did not rise by Amplitude within the time range. The cursor position is valid if a peak is detected, invalid if not.

Rising threshold, Falling threshold,Threshold



The data must cross Threshold from a level that is more than Noise rejection (hysteresis) away from it and stay crossed for a time of at least Delay after level crossing. For the Rising threshold mode, the data must increase through the threshold, for Falling threshold mode it must fall through the threshold. In Threshold mode the crossing can be in either direction. The picture shows a rising threshold. For waveform channels, the crossing point is found by linear interpolation of the data points on either side of the threshold crossing. The cursor position is invalid if a crossing does not occur.

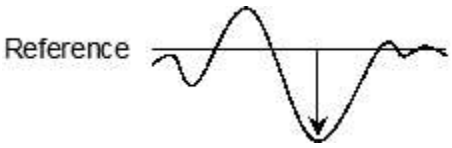
Outside thresholds, Within thresholds

These two modes are similar to the rising and falling thresholds modes except they search for data the lies outside two threshold levels or within two threshold levels. In addition to the Threshold level and Noise rejection (hysteresis) fields, there is a Threshold level 2, which sets the second threshold level. The data must cross a threshold from a level that is more than Noise rejection away from it for a period of at least Delay.

Maximum value, Minimum value

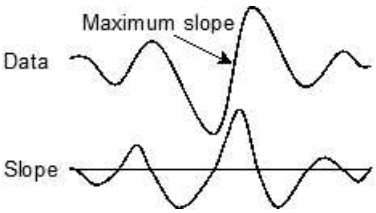
The result is the position of the maximum or minimum value. It is invalid if there is no data in the search range.

Maximum excursion



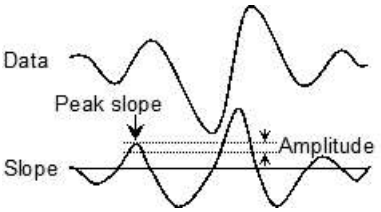
There is an extra field in this mode for the Reference level. The cursor is positioned at the point that is the maximum distance in the y direction away from the reference level. The result is invalid of there is no data in the search range.

Steepest rising, Steepest falling, Steepest slope (+/-)



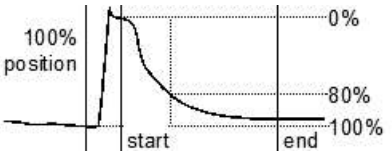
These modes are for waveform channels only. They have the extra field Width for slope measurement that sets length of data used to evaluate the slope at each data point. The result is the position of the maximum, minimum or maximum absolute value of the slope. The result is invalid if there is no data or not enough data to calculate a slope or if the channel is not waveform or WaveMark.

Slope peak, Slope trough



These analysis modes find the first peak or trough in the slope within the search range that meets the Amplitude specification. The Width for slope measurement field sets the length of data used to evaluate the slope at each data point. The Amplitude field sets how much the slope must rise before a peak and fall after it (or fall before a trough and rise after it), for it to be accepted as a peak. The Amplitude units are y axis units per second.

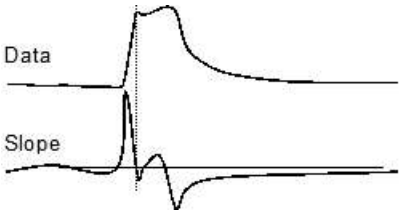
Repolarisation %



This mode finds the point at which a waveform returns a given percentage of the distance to a baseline inside the search range. The start of the search range defines the position of 0% repolarisation. The 100% position, which can be anywhere, and Width fields identify the 100% level. The Repolarise % field (drawn at 80% in the picture) sets a threshold level in percent relative to the 0% and 100% levels. The position is the first point in the search range that crosses the threshold. The result is invalid if the threshold is not crossed.

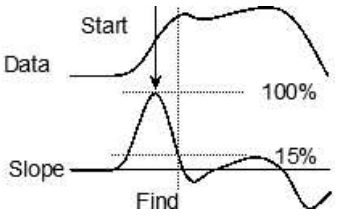
Turning point

This finds the first point in the search range where the waveform slope changes sign. Put another way, it finds a localised peak or trough. The Width for slope measurement field sets the data range to calculate the slope. The picture shows how this can find the top of a sharp rise where Maximum would get the wrong place. To use this, set a cursor on the peak slope and start the search from that point. The result is invalid if no point is found.



Slope%

Use this to find the start and end of a fast up or down stroke in a waveform. The Width for slope measurement field sets the time width used to calculate the slope. The Slope% field sets the percentage of the slope at the start of the search area to find. Set a cursor on the maximum or peak slope, then use that as the start point and search for the required percentage. A value of 15% usually works reasonably well. The result is valid if the slope value is located.



Data points

The Point count field sets the number of data points to move by in the search range on the referenced channel. The result is the time of this data point. The cursor position is invalid if there is no data item in the search range.

Expression

The new position is obtained by evaluating the Position field, which normally holds an expression based on cursor positions, for example `Cursor(0)+1` or `(C0+C1)/2`. If the expression fails to evaluate the cursor is marked as invalid; Position if search fails is not used and the cursor does not move.

Gap start, Gap end

These search modes were added at Spike2 [10.19]. The new position is found by searching the selected channel for a gap in the data of at least the width set by the Minimum width field. If the channel is waveform based and the width is less than or equal to the channel sampling interval, the gap is set to one clock tick more than the sample interval (so the command finds discontinuities in the waveform data). If you search for Gap start, the position found is the last data point before the gap. If you search for the Gap end, the position found is the first data point after the gap. The width of a gap is the time difference between the first point after the gap and the last point before it.

Horizontal

Click this button to apply the current values, close the dialog and display the Active horizontal cursor dialog. This button is disabled if there are no horizontal cursors.

Search Right, Search Left

If cursor 0 is active, these two command cause the cursor to search for the next or previous position that satisfies the active mode. If the cursor 0 mode is Expression, the cursor goes the same way for both commands.

The keyboard shortcut for these commands is `Ctrl+Shift+Left` and `Ctrl+Shift+Right` (`Left` and `Right` are the ← and → arrow keys).

Note that if a cursor is Fixed, it will still move in response to a cursor search command.

Actions on search success and failure

If all cursor searches succeed, all cursors move to their new positions and all cursors are marked as valid.

If the cursor 0 search fails, no cursors move and cursor 0 is marked invalid and its label has ! appended to flag a failure.

If the search for any other cursor fails and Position if search fails is not set or is invalid, the cursor is set to the centre of its search range, the cursor is marked invalid and `!` is appended to the label. If Position if search fails is set to a valid expression, it sets the cursor position and the cursor is marked as valid and (from version [10.19] onwards) the label has `*` appended as a warning that the search failed.

Active horizontal cursors

Normally, horizontal cursors are static; they stay where they are put. Using the Active horizontal cursor dialog, horizontal cursors in time views can be made active; they will move to a level found by a measurement (normally taken from the channel on which the cursor is located). This search is repeated whenever cursor 0 is iterated or a vertical cursor is moved on which they depend. This repositioning is normally carried out after all the vertical cursors whose position is used have been repositioned. Active horizontal cursors can be used as a simple way of marking levels within your data; they are also useful for extending the capabilities of analyses that generate measurements from data files.

Link to vertical cursors

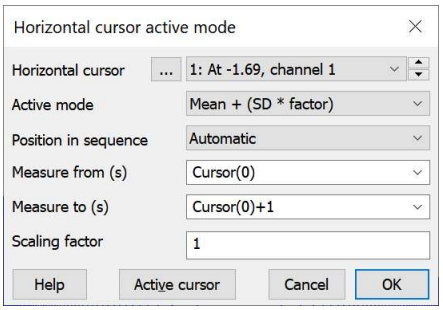
Normally, active horizontal cursors are automatically repositioned after all vertical cursors upon which they depend have been repositioned. Sometimes it can be useful to override this order so mechanisms are provided to force a horizontal cursor to be repositioned before or after the vertical cursors.

Valid and invalid horizontal cursors

Active horizontal cursor positions are either *valid* or *invalid*; invalid cursors have an horizontal exclamation mark at the end of the label. The cursor position is invalid if the active search fails or is based upon an invalid expression. Expressions and measurements that use invalid cursor positions are also invalid, the measurement analyses all reject points that come from invalid measurements. Horizontal cursor positions are made valid by a successful active mode search or by any operation that moves them to a specific place such as dragging.

Active horizontal mode

The Cursor menu Active horizontal mode command opens the Horizontal cursor active mode dialog. Each horizontal cursor is attached to a channel and has an active mode. The mode determines where the cursor positions itself after vertical cursor 0 has been commanded to seek. Modes vary from Static (where the cursor stays where it is) to modes that involve calculations of the cursor level based on time ranges of the horizontal cursor channel.



The fields displayed in the dialog depend on the selected mode. The fields are:

Horizontal cursor

This control selects a horizontal cursor. The control displays the cursor number, label and the channel on which the cursor is placed as "n: label on chan c" where n is the cursor number, label is the cursor label and c is the channel number. The `'...'` button to the left of the cursor selector opens the horizontal cursor label mode dialog for the current cursor to make it easy to change the label.

Active mode

This field sets the active mode of the cursor, which in turn determines the other fields to display. The simplest modes are:

Static — When you add a new horizontal cursor, it starts off as Static, that is, not an active cursor. In this state, the cursor stays where you put it; it is not changed by a change in the data or the position of other cursors.

Expression — The horizontal cursor position is set by an expression string such as `"HCursor(2)+1"`. If you use an expression such as `At(c1)` and do not supply a channel, the optional channel number is the channel of the horizontal cursor.

Value at point — The channel value at a point (set by the Position field). If this is a vertical cursor position, the horizontal cursor will track the intersection of the vertical cursor with the data channel.

The next set of modes take a measurement from a time range (set by the Measure from and Measure to fields). All times are set in seconds. Measurements are always taken from the channel where the horizontal cursor is currently placed. The modes are:

Mean level — The mean channel data value within the specified time range.

Maximum value — The maximum channel data value within the specified time range.

Minimum value — The minimum channel data value within the specified time range.

Maximum excursion — The absolute maximum channel data value (the largest value ignoring the sign) within the specified time range.

Median — The median channel data value within the specified time range. This can be a more robust measure of the centre of the signal than the mean as it is not affected by large outliers. Added at [11.00].

The final modes calculate a mean channel data value within the specified time range and then adds to it a measure of the channel spread multiplied by a specified factor (set by the Scaling factor field, which can be negative).

Mean + SD*factor — The mean level of the channel data plus the standard deviation of the data times the scaling factor.

Median + size*factor — The median of the channel data plus the channel 'size' times the scaling factor. The size is measured as the median of the absolute differences of the channel data from the channel median. This is similar to using the mean and standard deviation, but is less susceptible to large outliers. Added at [11.00].

Position in sequence

This field is present for all active modes. It controls when Spike2 repositions the horizontal cursor with respect to any active vertical cursors. The order is important, as active cursor measurements may depend on other active cursor positions. You can choose from:

Automatic — After all the vertical cursors upon which it depends. Active vertical cursors are positioned in ascending numerical order, so if the highest vertical cursor used in the calculating the position of this horizontal cursor in `n`, vertical cursor `n+1` can use the new position of this cursor to calculate its position.

At the start — Before the active vertical cursors move. This allows you to find a level with respect to the old vertical cursor positions.

At the end — After all vertical cursors have been positioned. This means that any active vertical cursors that use horizontal cursor values will all use the previous positions, not the new ones.

When following these rules, if multiple horizontal cursors would update at the same time, they update in rising numerical order.

Note that from Spike2 8.09, each time a horizontal cursor moves, it saves the place it moved from, and this position is available with the script command `HCursorX()` and with the Dialog Expression commands `HCursorX()` and `HnX`.

Measure from, Measure to, Position

These can be any View-based time expression, and will usually involve the positions of vertical cursors, particularly when cursor 0 is being used to iterate through features. If Measure to is less than Measure from then they are reversed so that they still specify a sensible X axis range.

Scaling factor

This field is used with commands that calculate a level as a measure of the signal mean plus some multiple of a measure of the signal width. This will usually be a number in the range -5 to +5, but we do not place any restriction on the range of allowed values.

Active cursor
Click this button to apply the current values, close the dialog and display the Active vertical cursor dialog.

Cursor context menu commands

If you right-click on a horizontal or vertical cursor, a context menu opens. This holds commands that can be applied to items in the view that are close to the mouse position. In particular, it holds the item Cursor n for vertical cursor n or HCursor n for horizontal cursor n. Move the mouse pointer over one of these commands to open a new context menu.

This menu contains commands to set the cursor position, set the cursor label, copy the cursor value to the clipboard and delete the cursor. For vertical cursors, you can open the Active cursor mode dialog for that cursor. Horizontal cursors also have the option to copy the cursor position relative to any of the other horizontal cursors.

Lock to Cursor
This command is available for both horizontal and vertical cursors. In both cases it is a short cut to setting a vertical or horizontal active cursor mode. To stop a cursor being locked, open the Active cursor mode dialog for that cursor and set the cursor to Static or use the Unlock command from the Lock to Cursor menu.

Cursor type	Action
Vertical	This command is available if there are any visible lower-numbered vertical cursors. The available cursors appear in a list and if you select one, the current cursor is set to keep itself at the same separation when the selected cursor moves. If you select cursor `n`, this sets the current cursor to Expression active mode and sets the expression to `Cursor(n)+separation`, where `separation` is the current distance between the two cursors. If the cursor is already locked to one of the cursors, the cursor it is locked to is marked in the list.
Horizontal	This command is available if there are any visible vertical cursors. If you select one, the horizontal cursor is set to an active mode to track the intersection of the vertical cursor with the horizontal cursor channel. That is, if you select vertical channel `n`, the horizontal cursor is set to Value at point mode and the position to `Cursor(n)`. If the cursor is already locked to one of the cursors, the cursor it is locked to is marked in the list.

Delete/Hide
You can Delete vertical cursors or Hide cursor 0 (which always exists). This command can be disabled with the `CursorFlags()` script command.

Fix/unFix position
This command was added at version [10.10]. It fixes the position of the nominated vertical cursor for interactive use (scripts and active cursor commands will still move it). Fixed cursors display an F after the cursor number. When a cursor is fixed, this command changes to unFix position. You can fix or unfix a cursor and disable this (and other) context commands with the `CursorFlags()` script command. The View menu Standard Display command removes fixed cursors regardless of values set by the `CursorFlags()` script command.

12: Sample menu

Sample menu

The sampling menu controls the sampling configuration, and can start, stop and pause data capture. To capture data you need a CED 1401 interface (a Micro1401 or a Power1401). You can also capture data from other devices using the Talker interface. See the Sampling data chapter for more sampling information.

Sampling configuration

This command opens the Sampling Configuration dialog that sets the sampling parameters used when you select the File menu New command. You can load and save the configuration from the File menu Save Configuration and Load Configuration commands. See the Sampling data chapter for details.

Clear configuration

This command deletes all sampling window positions, associated result views for on-line processing, display trigger settings and WaveMark templates. It preserves the channel lists and all values visible in the Sampling Configuration dialog. If you want to reset the information in the Sampling Configuration dialog (list of channels, sample mode...), use the Reset button in the Sampling configuration dialog Channels Tab.

Script language equivalent

There is no exact script equivalent of this command, and it does not record and action. However, the effect of this followed by sampling can be achieved by using the `FileNew(0, mode%)` without including 2 in the `mode %` value.

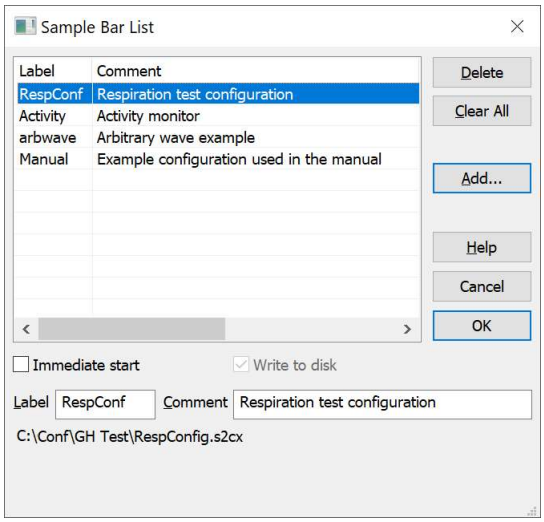
Sample Bar

You can show, hide and manage the Sample Bar from the Sample menu. 

You can show and hide it by right-clicking on any toolbar or on the Spike2 background. The Sample bar is a dockable toolbar with up to 20 user-defined buttons. Each button is linked to a Spike2 configuration file. When you click a button or use `Alt+`key (defined in the label), the associated configuration file loads and a new data file opens for sampling. Right-click a button to modify the button behaviour, open the associated configuration in the Sampling configuration dialog, remove the button from the bar or to open the Sample Bar List dialog.

Sample Bar List dialog

The Sample menu Sample Bar List… command opens the Sample List dialog from where you can control the contents of the Sample bar.



The Add button opens a file dialog to select one or more Spike2 configuration files (`*.s2cx`) to add to the bar. If you have old-format configuration files (`*.s2c`), you can convert them with Spike2 version 8.

If a configuration file holds a label or comment, it is used, otherwise the first 8 characters of the file name form the label and the comment is blank.

Select an item in the list to edit the label and comment. Changes you make here to the label and comment apply to the Sample Bar, but do not change the Label and Comment stored in the configuration file. To change that, double-click an item to open the Sampling Configuration dialog Automation tab, edit the Label and Comment fields, close the Sampling Configuration dialog and save the configuration from the File menu. If the label includes an ampersand (&), the next character becomes a short-cut key. To re-order buttons in the bar, drag items in the list. Delete removes the selected list item. Clear All deletes all items. Spike2 saves the Sample bar state in the registry. Each login to Windows has its own registry settings, so each has its own Sample Bar settings.

Immediate start

Check this to sample immediately on a click of the selected Sample bar button rather than wait for the Sample control toolbar Start button. This control is also available by right-clicking on the buttons in the Sample bar. Checking this button also enables the Write to disk field.

Write to disk

If you have checked the Immediate start box you can also control whether the Write setting of the Sample control toolbar is checked when sampling begins. If you have not checked Immediate start, this field has no effect and the state of the Sample control toolbar Write check box is unchanged.

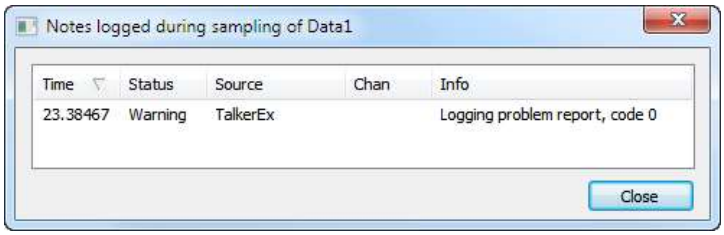
Conditioner Settings

Spike2 supports several programmable signal conditioners. These devices amplify and filter waveform signals, and can provide other specialist functions. If suitable conditioners are installed in your system, this menu command opens the conditioner dialog., see the *Programmable signal conditioners* chapter.

Sampling controls

This command hides and shows the Sample control toolbar. The controls within this toolbar are duplicated in this menu as the Start sampling, Write to disk, Abort sampling and Reset sampling commands. This toolbar is normally made visible (if hidden) when you open a data file for sampling. There is an Edit Preference to prevent this control bar being made visible every time you open a window for sampling. There is more information in the *Edit menu* chapter.

Sampling Notes



The Sample menu Sampling Notes command opens the resizeable Notes dialog that lists Warning and Error messages that were generated either by a 1401 or a Talker during sampling or to inform you that a waveform sample rate differs significantly from the desired rate set in the Sampling Configuration. The list of messages is cleared each time sampling starts. The number of notes in the table is also displayed in the Sample Status bar.

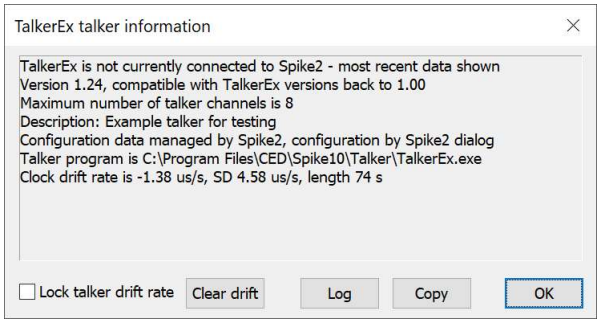
If you right-click on the dialog you can access three context commands:

Copy	Copy the selection to the clipboard as text
Copy for spreadsheet	Copy the selection in a format suitable for pasting into a spreadsheet. Each field is surrounded by double quotation marks and fields are separated by Tab characters.
Select All	Select all lines. To select sub-sections of the table: click to select a single line, `Shift`+click to select from the last selection and `Ctrl`+click to add the current line to the selection.

Talkers

The Sample menu Talkers command opens a pop-up menu with a Licensing... entry followed by a list of all Talkers that are currently connected or that Spike2 has seen in the past and has not been told to forget. The Talker name in the list is followed by an exclamation mark (!) if the talker is not currently connected (so is not available for sampling). If you select a talker name in the list, a further pop-up menu lists the following options:

Info...



This opens up the Talker Information Dialog. This lists information about the talker, including the talker interface specification version, current connection status, the maximum number of channels of data that are available, a brief description of the talker, the talker version number and any talker compatibility information that is available. Talkers that support the interface specification version 2 or later allow use of the `TalkerReadStr()` and `TalkerSendStr()` script commands. For example, with the `SoundCard` Talker connected on the local system the information might look like this:

Lock talker drift rate

If you check this box, the saved Talker drift rate will not be updated after each sampling session. All talkers we have generated (so far) usually have much the same drift rate with respect to the same 1401 (or to the PC clock if no 1401 is used) as long as the hardware is allowed to warm up before use. Changes made take into account the length of the sampling run compared to the duration of sampling used to generate the previous value. This check box was added at [10.17].

Clear drift

The Info... dialog displays the current Talker drift rate and the period over which it was measured. If you click this button you will clear all memory of previous sampling and set the initial talker drift back to 0.0 ms/s. You would use this if you had made a change to the system that invalidates the saved drift. For example, if you used a different 1401 interface or had changed the Talker or PC hardware, or if you suspected that something had gone badly wrong during a sampling session that had invalidated the timing in some way. A drift rate of 1 part in 10^5 is not uncommon between electronic clocks (0.01 ms/s). Because clearing the drift information is destructive, you are asked to confirm that this is what you want. This command was added at [10.17].

From version [10.19], if the Talker supports device qualification, when you click the button you will get to choose which device you are clearing, or All Devices.

Log, Copy
These buttons copy the talker information to the Log view and to the Clipboard. A CED engineer might ask you to use these commands if you are having problems with a Talker. You might also want to save this information as part of your experimental record.

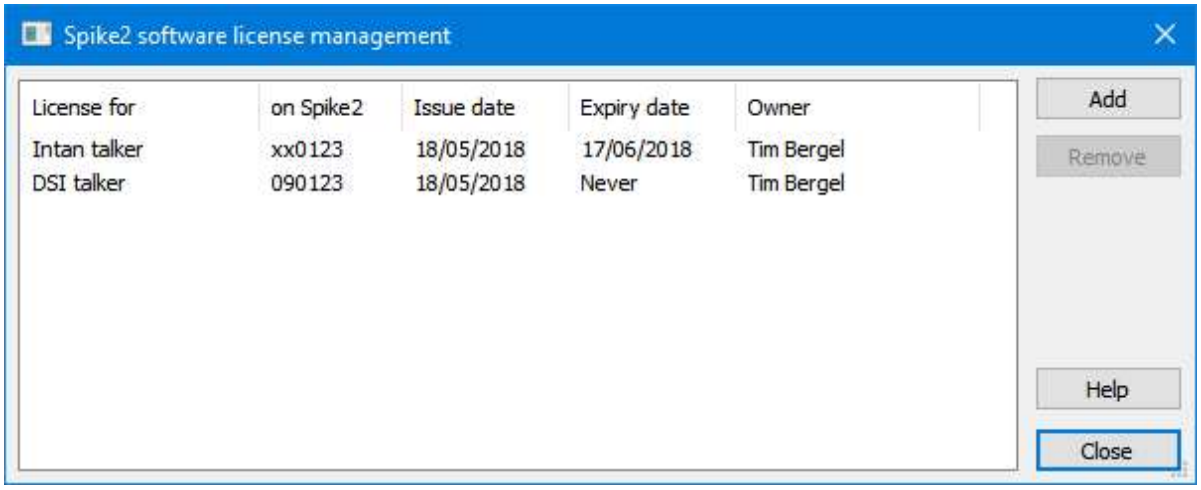
Run
This option is enabled if Spike2 knows the location of the Talker `.exe` file on the same machine as Spike2 is running and the Talker is not currently connected to Spike2. This command will attempt to launch the Talker program with the same command line as was used the last time it was connected to Spike2. This command was added at version [10.16].

Disconnect
This command will temporarily disconnect the Talker from Spike2. If a disconnected talker is still running, it may reconnect itself spontaneously (this depends on the talker design). In some cases you may need to use the Talker application 'Connect' command. Talkers written by CED will normally try to reconnect if disconnected.

Forget
Spikes maintains a list of all talkers that it has seen so that you can set a sampling configuration without the Talker connected. The Forget option deletes all information about a Talker; you must reconnect to use it again. You cannot forget a connected Talker. See the *Sampling data* chapter for more about Talkers.

Manage licences

The CED Talker system allows Talkers to be linked to specific copies of Spike2 by the Spike2 serial number. Some Talkers are free, but others are available as paid for additions to Spike2. To allow us to provide updates for these that can be downloaded from our web site, we have implemented a Licensing scheme so that users will have access to the latest versions as soon as they become available. This will also allow us to issue trial licences that will expire after a time period.



The Sample menu->Talkers->Licensing... command opens a dialog from which you can view the currently installed software licenses. Each installed license is shown on a separate line displaying: the Talker name, owner, issue and expiry dates. Installed licenses can be selected by clicking on the relevant line of information.

If a license has expired the expiry date is replaced by "Expired" in red text. Spike2 will warn you that an installed license has expired or is going to expire soon each time it starts up.

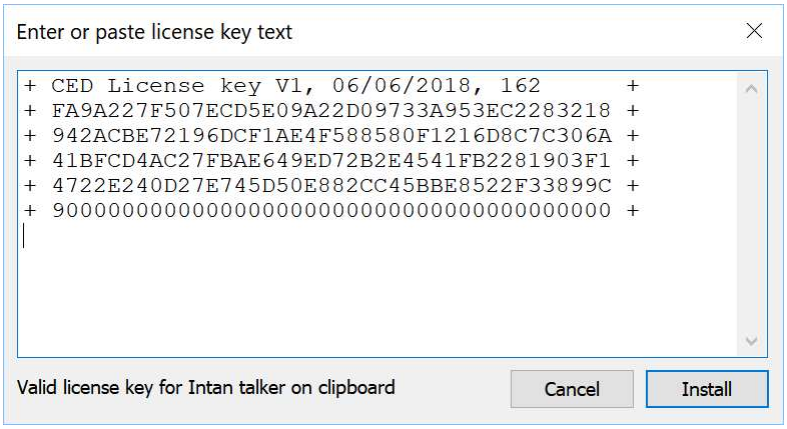
The Add button provides you with a separate dialog that is used to install a new license using license key text that will have been emailed to you by CED.

The Remove button will remove the currently selected installed license. Note that this option is permanent and cannot be undone except by reinstalling the original license key or a replacement key obtained from CED.

Add licence

This dialog opens when you click the Add button in the Talker licensing dialog. It used for entering license key text which, if correct and valid, will install the software license held within the key.

You can paste (or type) the license key text into the rectangular area occupying most of this dialog. If the Windows clipboard holds a valid licence key when the dialog opens, it is pasted automatically into the text area (as in the example below - which is not a valid key, in case you were wondering!)



Information regarding the validity of the current text is displayed at the bottom of the dialog. The Install button becomes enabled when the licence key area holds a valid license. Click on Install to add the licence to Spike2.

No licence detected

A talker program has attempted to connect to Spike2. This connection attempt has been blocked because the talker requires licensing and a valid license for this talker has not been found or has expired.

The Log view window holds more details of the problem and the name of the talker in question.

In order to make use of this talker you will need to obtain and install a suitable license key from CED. 30 day trial licenses are provided free of charge while full licenses are charged-for. The Talker licence system was instigated at Spike2 version 9.05. Talkers supplied before the release of 9.05 did not require a licence key. Please contact CED if your paid-for Talker refuses to load due to a lack of a licence key.

Talker Licences

Spike2 incorporates a licensing system for optional software components that are charged-for by CED. At the time of writing, this only applies to some Talkers, allowing us to provide updates over the web.

The licenses currently installed on a copy of Spike2 can be viewed using a dialog provided by the Sample->Talkers->Licensing... menu, this dialog is also used to add new licenses and to remove existing ones if required.

Licenses are either permanent or they have an expiry date after which they will become invalid. Temporary licenses with an expiry date are normally generated free of charge by CED to allow the licensed software to be evaluated; permanent licenses are normally charged for.

Some talker licenses can be used with all versions of Spike2 (so a license issued for a previous major version will still work correctly when you upgrade), other licenses are specific to a given Spike2 version so that when you upgrade you will also have to upgrade your talker license(s).

In order to obtain a software license from CED you will need to contact us. An easy way is to send an email to sales@ced.co.uk or info@ced.co.uk requesting a license. You will need to tell us:

- The talker for which you require a license

- The name of the person or organisation that will own the license
- The Spike2 serial number to which the license will apply
- Whether you require a 30-day trial license or a permanent one

If this is a permanent licence requiring payment, you will need to follow the ordering rules for your organisation.

Once your license has been generated a license key will be emailed to you. This can be installed into your copy of Spike2 using the Add button in Sample menu Licensing... dialog and pasting the key text into the text area provided. Please can you make sure that you save the email with the license key text somewhere safe as you might need it again should you reinstall Spike2.

CED software licenses are only usable with copies of Spike2 bearing a specific serial number, other copies of Spike2 will refuse to allow the license to be installed. So it is very important that you correctly specify the serial number of your copy of Spike2 when you request a license from CED.

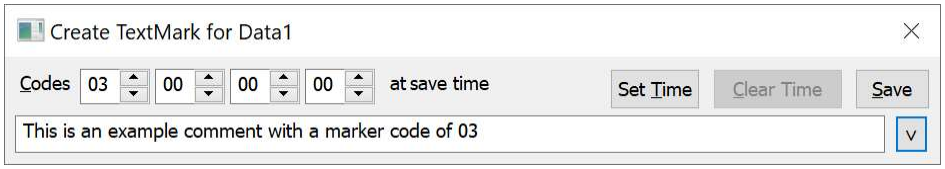
Sequencer controls

This command hides and shows the sequencer control panel that is available during sampling if an output sequence is active. There is more information about the sequencer control panel in the *Sampling data* chapter.

Create a TextMark

When you are sampling data, this menu command (or Toolbar button or short cut `Ctrl+T`) opens the Create TextMark window. To use this command you must enable a TextMark channel in the Sampling Configuration. This dialog is also available from the Sampling Configuration TextMark setup dialog, with slightly different options.

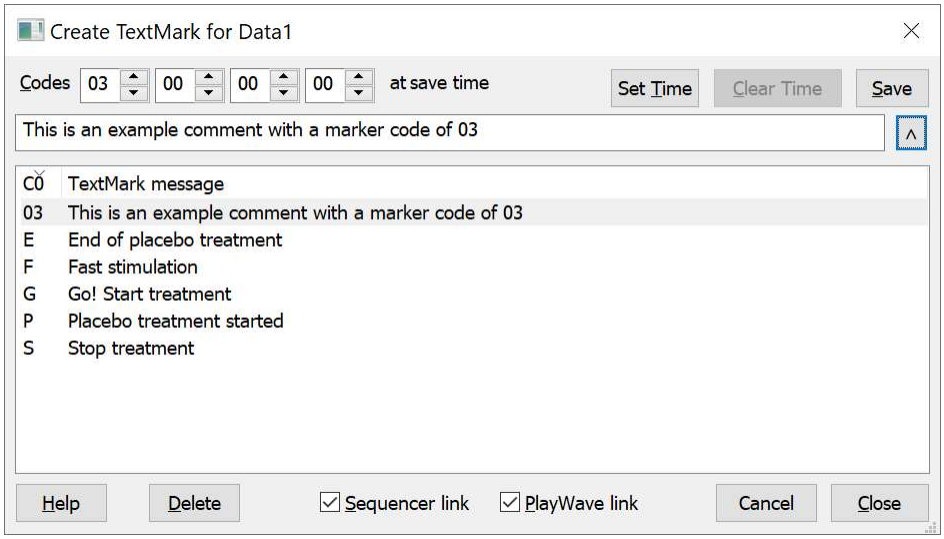
You can always add markers by hand even if you enabled serial line input of TextMark data in the Sampling Configuration. However, you cannot add a marker at a time before or equal to the last saved marker time. If the serial line is adding markers at a fast rate you may not be able to use the Set Time button as if the serial line adds a marker after the set time, this will block attempt to add earlier markers. The script equivalent of this dialog is the `SampleText()` command.



You can set the four marker codes to either a single printing character or to two hexadecimal digits. The four Code fields will not allow you to enter illegal code combinations. You can use the up/down spinner controls to adjust the marker code, or type codes into the fields.

You commit the comment to the file when you press the Save button. The Set Time button fixes the time associated with the comment, the Clear Time button clears the time. If you have not set a time, the time of the comment is the time when it was saved.

When you press Save, we also add the marker text and codes to the list of recently saved TextMark items. You can access this list by clicking the arrow at the right of the text area. You can force the dialog to open with the list displayed by holding down the `Shift` key when you open it, otherwise the dialog opens in the last used state in the current Spike2 session. This list is populated by TextMark items set in the Sampling configuration when configuring the TextMark channel, or if there are none, by the most recently-used TextMark items held in the Registry.



When you select a marker from the list, the text and codes associated with that marker are restored. You can modify the list of markers ahead of sampling by opening a file ready to sample, opening this dialog and setting or deleting markers before you start to sample. You can choose which Codes to display in the list by right-clicking in the list area to open a context menu. This is as described for the View menu TextMark list.

If this channel is a trigger for Triggered sampling mode in the sampling configuration dialog, the trigger happens when you click the Save button and the Save Time and Clear Time buttons are replaced by the message This channel is a trigger.

Double-clicking an item in the list is equivalent to the Save button; if you are sampling it adds the selected TextMark to the file.

Save

This adds the marker defined in the upper part of the dialog to the list, and if sampling has started, saves the TextMark and code in the file. Items are identified in the list by matching the TextMark message (the match is case insensitive) and ignoring the Codes. If a new message matches on in the list, the item in the list is replaced. Note that the list is sorted by the currently active column (indicated by a v above the column title). You can choose which column to sort on by clicking the column headings.

Delete

This button allows you to remove the currently selected items from the list.

Sequencer link, PlayWave link

Prior to [11.00], the only way to trigger the output sequencer to jump to a particular action, or to trigger the arbitrary waveform output to play a waveform was by using Keyboard markers. Now you can choose to activate the sequencer and/or a PlayWave area with the first code associated with the TextMark. These setting only apply to markers added from this dialog. If you have enabled serial line input of TextMark data in the Sampling configuration, this has separate controls to enable sequencer and PlayWave links.

Close

This closes the dialog and saves the current state to the sampling configuration. It also saves the text and codes of the last used 10 different TextMark items to the registry.

Cancel

This closes the dialog without saving any changes.

Display of comments in a file

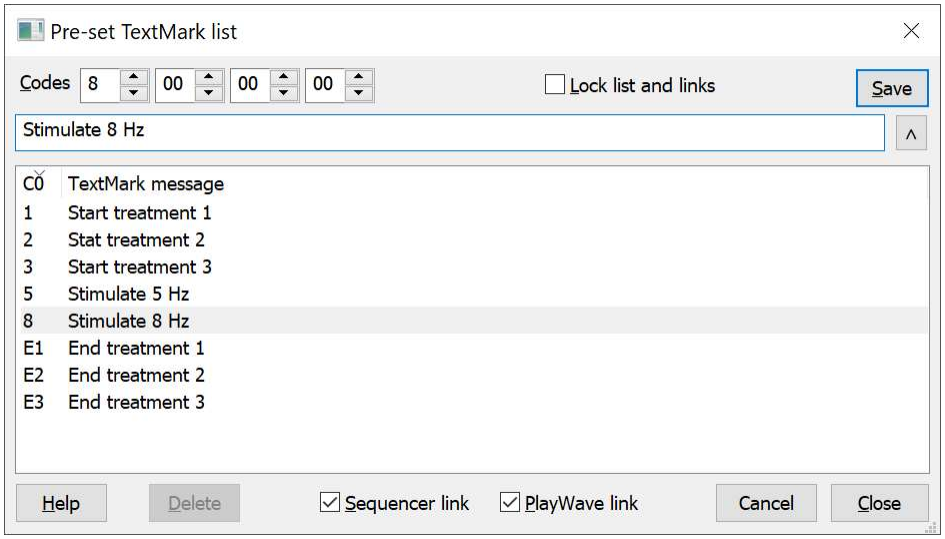
The standard way to display TextMark data is in Dots mode, and comments are shown in the file as a small rectangle. The colour of the rectangle depends on the first marker code associated with the marker. If the code is 00, the rectangle is yellow. Otherwise, codes use the same colours as WaveMark data, set by the View menu Change Colours command. Other modes are available, such as Text and State.

You can see the text associated with a TextMark by moving the mouse pointer over the rectangle and waiting a moment. The text appears as a tool tip. Any movement of the mouse pointer hides the text.

To edit the comment text and marker codes, double click the rectangle to open the Edit TextMark dialog.

Dialog use from Sampling Configuration

This dialog is also used from the Sampling configuration TextMark channel settings to pre-set a list of TextMark items and enable links between the first marker code and sequencer jumps and arbitrary waveform output for use during sampling:



The Set Time and Clear Time buttons are not available and the action of the Save button is to add an item into the list. Double-clicking an item in the list has no effect beyond selecting the item. The remaining controls operate in the same way as in the Create a TextMark dialog.

Lock list and links

If you check this box, the user may not change the sequencer and PlayWave links during sampling and if you provide a pre-set list of markers, the user may not change it (but they can apply other markers). If you do not provide a pre-set list, the list for sampling is pre-set to the last-used set of markers and can be modified. Setting the Lock check box prevents an update of these settings in the last-used configuration when sampling ends.

If you do not check this box, the user can enable and disable links and change the list during sampling and any changes are saved in the last-used configuration when sampling ends.

Script equivalent

The script equivalent of pre-setting TextMark items is the `SampleTextMark(code%|code%[], text$);` command. The script equivalent of the check boxes is the `SampleTextMark(flags%);` command.

Change Output Sequence

This command is used during data sampling to replaces the current text output sequence. It displays a File Open dialog in which you can select an output sequence file. This file will replace the current sequence.

This will fail if there is no current sequence, or if the sequence or table size is larger than the current size or the size reserved in the Sequencer tab of the Sampling configuration dialog. If a sequence is replaced, it runs from the first instruction. Any variables that are declared with initial values are updated as are all initialised table values. The Sampling configuration is not updated with the name of this file.

The script language equivalent of this is `SampleSequencer(mode%, src$)` with `mode%` values 4 and 6.

Update graphical sequence online

It is possible to modify and update a graphical sequence online, but this can only be done interactively, not by a script.

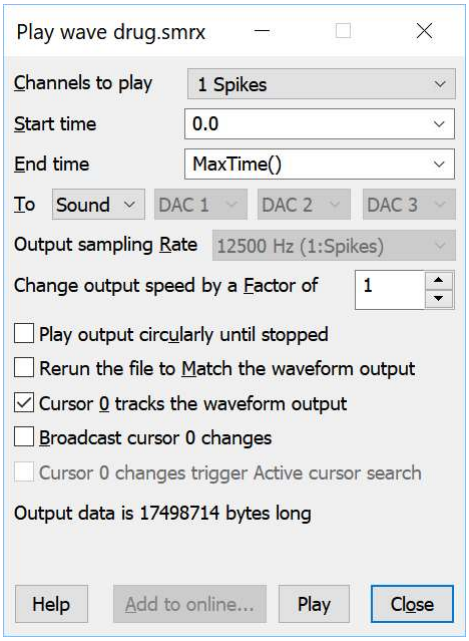
Graphical Sequence Editor

This command is enabled if you started to sample with a graphical sequence defined. It opens the Graphical Sequence Editor and you can then adjust the sequence and apply your changes while sampling. You must have started to sample (not just opened a file, ready to sample), before you can access this command.

You cannot modify basic sampling parameters, such as the time base and the digital and DAC channels that you are using. You also cannot extend the resources used by the sequencer beyond those used by the original sequence unless you have reserved instruction and table space in the Sequencer Tab of the Sampling Configuration dialog.

If you require complete control over all aspects of the output sequence, you can use a Text Sequence and the Change Output Sequence command. However, you must then cope with the complexities of the text sequencer.

Offline waveform output



When a data file is active, the Sample menu Waveform Output… command opens the Waveform output dialog. The script language equivalent of the majority of this dialog is the `PlayOffline()` command.

Channels to output

You can play up to 4 waveform or WaveMark channels through the 1401 DACs, or up to two channels through a sound card. To output multiple channels, select them in the time view, otherwise you can select single channels. You can change the channel selections after you open this dialog. Spike2 fills gaps in waveform data and gaps between WaveMark data items with zeros.

Time range

The Start time and End time fields set the file time range to play. You can link these to cursors in the time view. The values used are those set when you click Play.

To

You can select either a 1401 DAC for each channel, or if you select Sound in the first drop down list, the first (and second) channel is played through the sound card (mono for one channel, stereo for two channels).

Output sampling Rate

You can choose the sampling rate of any of the channels for output and the output device will get as close to this rate as it can. The 1401 sets the rate based on dividing down a 10 MHz clock; the rates achievable through a sound card depend on the card. All channels are replayed at the same speed. If any channel has a different sample rate, Spike2 resamples the data to have the same rate as set by this field (using linear interpolation).

Change output speed by a Factor of

You can speed up or slow down output by a factor of up to 4. This will change the pitch of sound or the repeat rate of a repetitive signal. The actual output rate depends on the capabilities of the output device.

Play output circularly

If you do not check this box, the waveform plays once only.

Rerun the file to Match the waveform output

If this box is checked when you click Play, the file will Rerun (simulate sampling) with the file time taken from the play position. This can be very useful when using Spike2 in teaching or for presentations, especially when using sound card output.

Cursor 0 tracks the waveform output

Check this box to have cursor 0 track the current playing position. Other Spike2 actions can also drive cursor 0: Spike shape analysis, Measurement processing, Active cursor searches and optionally by multimedia replay. From version [10.01], a Play command with the cursor 0 link set cancels the cursor 0 drive link for any other driver.

Broadcast cursor 0 changes

If you check the *Cursor 0 tracks the waveform output* box, this field is enabled. If you check it, each change made to cursor 0 for tracking purposes will be broadcast to all linked dialogs and to any multimedia windows associated with the time view. This will cause any open Cursor Values or Cursor Regions dialog box to update, which may make your system slow down.

Cursor 0 changes trigger Active cursor searches

If the *Broadcast cursor 0 changes* box is enabled and checked, this field is enabled. Each change to cursor 0 will cause an active cursor search.

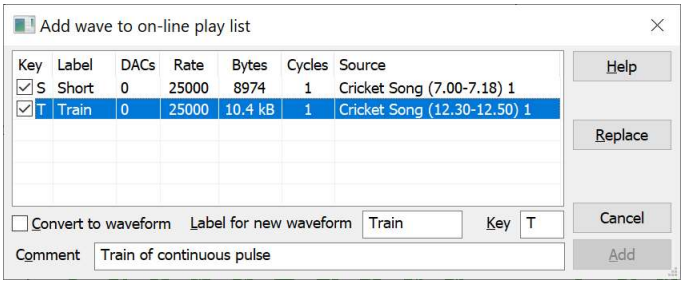
Play and Stop

This button starts output (and changes to Stop, which stops it). The output pays no attention to the scale and offset values associated with the channel. The 16-bit data values recorded for the file are played out. If you have an 8-bit sound card and try to play a low level signal, the result may be disappointing (or just silence).

Add to online…

This option opens a new dialog that adds the selected waveforms to the on-line waveform output list.

Add to online



This option adds the selected waveforms to the on-line arbitrary waveform output list so that they can be replayed during data acquisition. The data file must have been saved to disk. You are allowed up to 62 (20 with Micro2 and Micro3) waveforms. As long as the list of waveforms is not full and the current waveform is less than 32 MB is size you can Add the waveform to the list. You can also Replace an existing waveform in the list. You can give each waveform a short name that labels the buttons for interactive replay of waveforms during data acquisition. There are more controls for the list of waveforms for replay in the Sampling Configuration dialog in the Play waveform tab.

The script language equivalent of this dialog is the `PlayWaveAdd()` command.

Label for new waveform

If you select an item in the list, this field is filled in for you. This sets a text label that is used to label the Play waveform toolbar buttons. The text is purely for display purposes and should identify the waveform. The label is limited to 7 characters.

Key

This is the Key code that identifies the wave, and is the marker code that Spike2 adds to the Keyboard Marker channel when you activate a wave from the Play waveform toolbar. Most users set marker codes as the ASCII characters `0-9`, `A-Z` and `a-z`. You are not allowed to set a code of 00 and codes 01 and 02 will not work as

keyboard triggers as they have special meanings. You can also use this key interactively during sampling to play the wave (unless disabled in the sampling configuration).

If you set a key that matches one of the items in the list, the item becomes selected, the Replace button is enabled and the Add button disabled. Likewise, if you select one of the items in the list, the Key field is set to match the item.

Comment

Since Spike2 version [10.20], each waveform has a comment of up to 80 characters. If set, this comment is used to provide a ToolTip in the Play waveform bar and is accessible from Scripts with the `PlayWaveComment$()` command.

Add

This button is enabled when the Key field is empty, or if it is set to a valid marker code that is not already in use. When you activate this option, the current waveform is added to the end of the list of waves. The Label... field and Key fields set the associated label and key. If the Key field is empty, the first unused key from the list: `0-9`, `A-Z`, `a-z` is used.

Replace

This button is enabled when the Key field holds a marker code that matches an item that is already in the list. When you activate this option, the current waveform replaces the wave with this marker code and the key remains in the same position in the list.

Convert to waveform

Waveforms can be saved as either a data file, time range and a list of channels or by converting the current channel list into the memory image that would be played through the output DACs on line. If a waveform has been converted the Source column in the list starts with "Wave from". The advantages of converting are:

- The output is not affected if the original file is moved, deleted or changed
- The (usually small) time for the conversion is saved each time you sample data

However, there are also disadvantages to converting:

- It can take a lot of memory to hold the converted data which can slow Spike2 down and may cause your system to run out of memory
- If the sampling configuration is saved, the converted data is also saved, which makes the configuration files much larger and can slow down system operation.
- If this was a virtual channel and you want to be able to change the waveform by editing the virtual channel expression.

To prevent the system running out of memory, there is a limit on the amount of memory (32 MB) that any one converted wave may use. If any channel in the play list is a WaveMark or a memory channel or a duplicate channel or a virtual channel, Spike2 will check the Convert to waveform box, as there is no guarantee that these channels will exist (or have the same marker filter) when sampling is requested. If you really want to fetch the data from the duplicate, memory or virtual channel (which will work ONLY when the file is open in Spike2 as this is the only time these channels are accessible), then clear the check box.

If a channel has a channel process applied to it, you must either leave the data file open with the channel process applied or you must convert the wave. This is because a channel process only has effect while the data file is open in Spike2.

13: Script menu

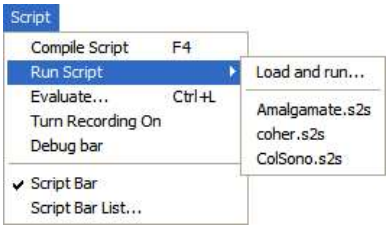
Script menu

The script menu gives you access to the scripting system. From it you can compile a script, run a loaded script, evaluate a script command for immediate execution and record your actions as a script.

Compile Script

This option is enabled when the current window holds a script that is not already compiled. It is equivalent to the compile button in the script window. Spike2 checks the syntax of the script, and if it is correct, it generates a compiled version of the script, ready to run.

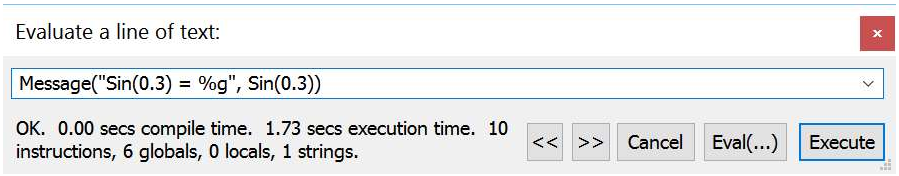
Run Script



This option pops-up a list of all the scripts that have been loaded so that you can select a script to run. Spike2 compiles the selected script and if there are no errors, the script runs. If you run a script twice in succession, Spike2 compiles it for the first run, and uses the compiled result for the second run, saving the compilation time. If a script stops with a run time error, the script window is brought to the front and the offending line is highlighted.

You can also select the Load and Run… option from which you can select a script to run. The script is hidden and run immediately (unless a syntax error is found in it).

Evaluate



This Script menu Evaluate... command (short-cut key `Ctrl+L`) opens the Evaluate a line of text window where you can type in and run one line scripts (of up to 255 characters in length). You can cycle round the last 10 script lines with the << and >> buttons. The `Eval(...)` and `Execute` buttons are disabled when a script is running as only one script can be compiled and/or run at a time. The `F1` (Help) key can be used to lookup the text word nearest to the text caret or selected text in the evaluate window.

Execute button

The Execute button compiles and runs your script in the same way as if it were saved as a script and run.

Eval(...) button

Eval(...) does the same, and displays the value of the last expression on the line at the bottom left of the Evaluate window and also in the Log view. It does this by examining the line of text and attempting to find the last expression on the line, then inserts the `Eval()` command around the expression. So a line that you type as:

```
statement1;statement2;expression
```

is transformed into

```
statement1;statement2;Eval(expression)
```

before being passed to the script system.

If the expression is an assignment, for example:

```
var x; x := Sin(1)
```

this is transformed into:

```
var x; x:=Sin(1);Eval(x)
```

Version [10.16] onwards also recognises +=, -1, *= and /= as assignments. From version [10.15], if the original text already includes **Eval()**, we do not change the expression and offer it to the compiler as it is. Prior to this version, the **Eval()** option did not work if the original text included a comment.

Note that if the **expression** does not have a value, for example:

```
Draw()
```

this results in the message: `a proc has no return value at 'Draw'` as there is nothing to evaluate.

From version [10.16] a wider range of expressions can be evaluated without generating an error including:

```
var x:= 2          'now evaluates as 2
var a[10]; a[1]:=3 'now evaluates as 3
var y:=1; y += 4   'now evaluates as 5
```

That is, if the line includes an assignment, we evaluate the variable to the left of the assignment operator.

Turn Recording On/Off

You can record your actions into the equivalent script. Use this command to turn recording on and off. When you turn recording off, a new script window opens that holds the commands that are equivalent to your actions since you last turned recording on. When recording is enabled REC appears in the status bar.

Caveats

Note that not all actions are recordable and that there is no guarantee that a recorded script will run. Some actions depend on the overall state of the system and this may well not be the same as when the script was recorded. That said, the majority of actions do record and the recording can often be used as the skeleton for a script. We do make some attempts to not record actions that have no effect. For example, if you drag a window around, we only record the final position before you perform some other action; we do not record every intermediate position.

If you come across an action that does not record and that would be useful to record and for which a script equivalent exists, let us know and we will consider implementing it.

Recording is not compatible with running a script, so if you start a script running, any recording will stop. This can be unexpected. For example, if you set running a script as part of the sampling configuration, turn recording on, then sample data (expecting to record the actions during sampling), as soon as the condition occurs to run the script, the recording will stop and a new window will appear holding the recorded actions up to that point.

Created channels and views

If you record actions that create new views or memory, virtual or duplicate channels, we assign variables to hold the new view handles and channel numbers. We do this because for these actions, the new channel number or view handle may not be the same if you repeat the action. For example, if you record the action of creating a memory channel and then set the channel comment, the recorded script is:

```
var v12% := ViewFind("Demo.smr");
var ch1%; 'MemChan created channel
FrontView(v12%);
ch1% := MemChan(2,0,0,0);
ChanShow(ch1%); 'Make it visible
ChanComment$(ch1%, "Comment for recordng example");
```

By recording the generated memory channel in a variable, the `ChanShow()` and `ChanCommanet$()` commands can act on the new channel, regardless of the number it was assigned. If you do not create a channel as part of the recording, it will be referred to by the channel number, not by a variable. When you stop recording, all knowledge of recorded channel numbers is cleared.

Special characters

When recording literal strings, such as channel comments and file paths, we have to treat some characters specially so that when the text is processed by the script compiler it ends up unchanged. In particular, the

backslash character must be doubled, otherwise it will be treated as an escape character and the effect will depend on the following character. Likewise, any " characters in the text must be 'escaped', otherwise they will terminate the string and cause the compiler to report an error. Similarly, if the recorded string holds any end of line characters, we convert them to "\r\n" and split the line. As an (extreme) example, if we had to record the text:

```
This is an "odd" comment for<CR><LF>C:\MyPath\Demo.smrx
```

where <CR><LF> stands for the Carriage Return, Line Feed character combination used to mark end of line, it would be recorded as:

```
"This is an \"odd\" comment for\r\n"
"C:\\MyPath\\Demo.smrx"
```

Which will generate the original text after being processed by the script compiler.

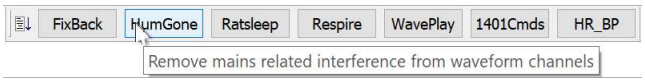
Timing

Recording does not take account of how long you take to do something. For example, if you record the actions: open new data document, start sampling, wait 100 seconds, stop sampling, the recorded script will not have the 100 second wait. If you want to include this you will need to edit the result. To insert time delays you can use a toolbar or dialog idle routine or the `Yield()` command.

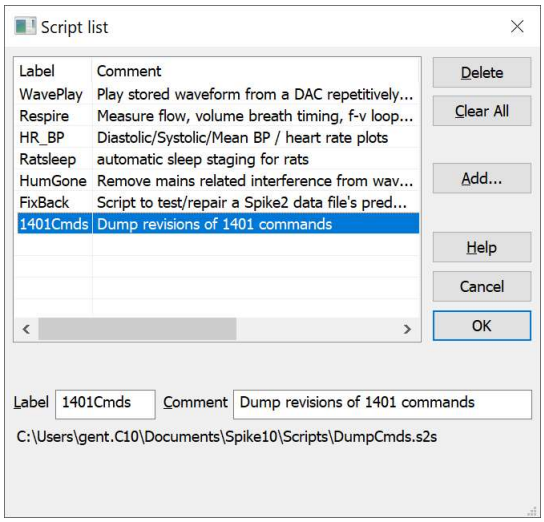
Debug Bar

You can show and hide the debug bar from this menu when the active view is a script. You can also show and hide the debug bar by right clicking on the title bar of the Spike2 window, or on the application window.

Script Bar



You can show and hide the Script Bar and manage the Script Bar contents from the Script menu. You can also show and hide the Script Bar by clicking the right mouse button on any Spike2 toolbar or on the Spike2 background. The Script Bar is a dockable toolbar with up to 40 user-defined buttons. Each button is linked to a Spike2 script file. When you click a button, the associated script file is loaded and run. There is also a user-defined comment associated with each button that appears as a tool-tip when the mouse pointer lingers over a button.



You can right-click on a button to open a context menu from where you can open the script in the script editor, remove the button from the bar or open the Script Bar List dialog.

The Script menu Script Bar List… command opens the Script List dialog from where you can control the contents of the Script Bar. This dialog is resizeable, making it easier to display long lists; if you stretch the dialog horizontally you can view the file path for each item (also available as a Tool tip).

The Add buttons opens a file dialog in which you can choose one or more Spike2 script files (`*.S2S`) to add to the bar. If the first line of a script starts with a single quote followed by a dollar sign, the rest of the line is interpreted as a label and a comment, otherwise the first 8 characters of the file name form the label and the comment is blank. The label is separated from the comment by a vertical bar. The label is be up to 8 characters (and can include an ampersand for keyboard activation, see below) and the comment up to 80 characters. A typical first line might be:

```
'$1401Cmds|Dump revision of 1401 commands
```

You can select an item in the list and edit the label and comment. This does not change the contents of the script file. Double-click an item to open the associated script file in a window.

You can re-order buttons in the bar by dragging items in the list. You can sort the list by clicking on the list header to sort by the contents of the column. If you click a column head a second time, the sort order reverses. The sort direction is indicated by a small arrowhead in the centre of the column.

The Delete button removes the selected item. Clear All removes all items from the list.

The dialog can be resized to display long lists. If you increase the dialog width, you will expose an additional column holding any comment associated with the items. You can change the column widths by dragging the division between columns in the column header. You can auto-size the column width to the contents by double-clicking the column divider in the column header.

Spike2 saves the Script bar state in the registry when it closes and loads it when it runs. Each different login to Windows has its own registry configuration, so if your system has three users, each has their own Script bar settings. Alternatively, you can have different experimental configurations by logging on as a different user name. The script language `ScriptBar()` function has the same functionality as this dialog.

Keyboard short-cuts for Script and Sample bars

You can activate a script in the Script bar from the keyboard. To do this, add an ampersand (&) to the script label before the keyboard character you want to use to run the script. The key is not case sensitive, so `&x` and `&X` are equivalent. Then, as long as the script bar is visible and there is not a script already running and there is not a modal dialog open (a dialog that requires you to close it before you can do anything else), you can type `Alt`+keyboard character to simulate clicking the script button.

You must be aware that any key you use overrides the use of that key by the menu system as you get first use of any key. For example, if you set a label to `"&FixChan"`, you have disabled the use of the keyboard to access the File menu.

The Sample bar also supports `Alt` key activation, but the script bar has higher priority. To activate a Sample bar configuration, the Sample bar must be visible, there must be no open sampling document and a script must not have disabled the Sample menu. Keys defined for the Script bar take precedence over Sample bar keys.

14: Help menu

Help menu

Spike2 has comprehensive Help built in and usually accessible with the F1 key or by clicking on a Help button. The Help menu gives you some control over the help system and also contains the About Spike2 command.

 The Help button in the system toolbar opens the main Help index, from which you can search based on the program structure. You should be aware that Spike2 has context sensitive help. If you use the F1 key, or if the current dialog has a Help button, using these will give you help on the current item. If you are writing a script or editing an output sequence, you can right click on a text item or use the F1 key to get help on the current text selection.

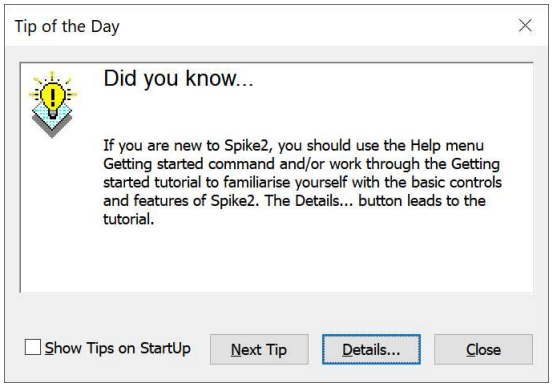
Using help

Spike2 supports context sensitive help that duplicates the contents of this manual and the script language manual. You can activate context sensitive help with the F1 key from most dialogs to get a description of each field in the dialog. You can use the help Search dialog to lookup topics that are not covered by the index.

In a script window you can obtain help by placing the cursor on any keyword in the script and pressing F1. To get help on a script function, type the function name followed by a left hand bracket, for example `MemChan(`, then make sure that the cursor lies to the left of the bracket and in the function name and press F1.

The help system has indexes, hypertext links, keyword searches, help history, bookmarks and annotations. If you are unsure of using help, once you have opened the help window, use the Help menu Using help command for detailed instructions.

Tip of the Day



You can show the Tip of the Day window when Spike2 starts by checking the box. Of course, if you hate this sort of thing, clear the check box and you never need see it again!

If the Details… button is enabled, it opens the on-line Help at a page that holds more information about the topic in the window. The tips are held in the file `sonview.tip` in the same folder as Spike2. If you have a useful tip that you would like to share with others, send it to us and we will very likely include it in the next release.

The list of tips restarts from the beginning with each update of Spike2 that changes the tip file. The first tip is an introduction for new users. This is followed by tips for the latest version, followed by some general advice, then tips for older versions, in order of increasing age.

View Web site

If you have an Internet browser installed in your system, this command will launch it and attempt to connect to the CED web site (ced.co.uk). The site contains down-loadable scripts, updates to Spike2 and information about CED products.

Getting started

This command runs a demonstration script that gives a quick tour of Spike2 basics. The option is disabled if the script is not present or if you are sampling data.

Other sources of help

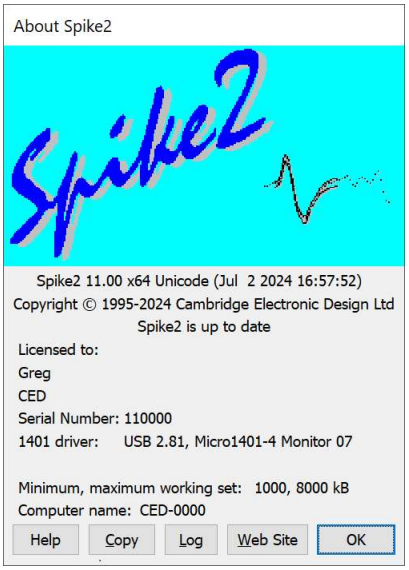
If you are having trouble using Spike2, please do the following before contacting the CED Software Help Desk:

1. Try looking in the on-line help system for more information. Use Search to find related topics.
2. If the problem is in sampling data, run the Try1401 program provided as part of Spike2. This checks out the 1401, device driver and interface.

If none of the above helps, FAX, email (softhelp@ced.co.uk) or call the CED Software Help Desk. Please include a problem description, Spike2 serial number and program version and a description of the circumstances leading to the problem. It would also help us to know the type of computer you use, how much memory it has and which version of your operating system you are running.

About Spike2

The About Spike2 command is found in the Help menu. It opens an information dialog that contains the serial number of your licensed copy of Spike2, plus your name and organisation. The dialog also display information about the 1401 device drive and your 1401. If you contact us for assistance we will often need the information in this dialog.



The Copy button writes the Spike2 version and serial number, your user name and organisation (that was entered when you installed Spike2), the device driver and 1401 information and your working set size to the clipboard. It also includes information about the file paths that Spike2 will use, the Spike2 installation folder, any installed exporter filters and a list of the Talkers that Spike2 knows about. You can paste this into an email or other text document; this information can help CED engineers diagnose problems. Typical output is:

```
Spike2 11.00 x64 Unicode (Jul  2 2024 16:57:52)
Serial no: 110000
User name: Greg
Organisation: CED
1401: USB 2.81, Micro1401-4 Monitor 07
Operating system: Windows 10 x64 build 19045
Computer name: CED-PC02
Working set: 1000, 8000 kB
User data: C:\Users\user.domain\Documents\Spike11\
Shared public data: C:\Users\Public\Documents\Spike11Shared\
User application data: C:\Users\user.domain\AppData\Local\CED\Spike11\
Application data: C:\ProgramData\CED\Spike11\
Installation path: C:\Program Files\CED\Spike11\x64\Release\
Found 0 *.sxl exporter(s) in C:\Program Files\CED\Spike11\x64\Release\Export\
Talker repository: C:\Users\user.domain\AppData\Local\CED\Spike11\sp2talks.datx
holds 5 Talker(s), *=loaded
 TalkerEx   1.26 (1.00) Example talker for testing
*SoundCard  1.04 (1.00) Records the input from a sound device
 XKeys      1.13 (1.00) Keyboard markers from external system
 MouseTalk  1.13 (1.00) Talker generating mouse X and Y co-ordinates
 MKeys      1.13 (1.00) Marker channel from external keyboard
```

The two numbers after the Talker names are the Talker version and the oldest talker configuration it is compatible with.

The Log button writes the same information as generated by Copy to the Log view.

The Web Site button will attempt to open your web browser at the CED web site on the *Latest software versions* page.

Spike2 version information

The top line of text identifies the version of Spike2 that you are running, the target platform (x86 for a 32-bit build or x64 for a 64-bit build) and the date when we built the code. This is followed by our copyright statement. The third line is present if Spike2 can contact the CED website. If there is a more recent version available, there will be a message of the form: Version 11.nn is available for download. If your version is up

to date, we display: Spike2 is up to date. If we could not contact the site, or if the information we read makes no sense, the line is blank. If there is an update available you can click the Web Site button to go to our general update page to collect it.

1401 device driver

If there is a 1401 device driver installed, the driver revision is displayed. If the driver is older than Spike2 expects, you will be warned. Spike2 displays the type of 1401 and the monitor version if a 1401 is connected and powered up. If you have installed Spike2 correctly, you should not have any problems as Spike2 comes with the latest 1401 drivers available at the time we generated the release. If you really need to update the drivers independently of Spike2 you use the 1401 Windows installer link on our web site. If the 1401 is synchronized to another 1401 as a slave unit, the monitor revision is followed by `Sync`.

64-bit Windows 10 and driver signing

Microsoft require device drivers for 64-bit versions of Windows 10 to be signed for each version of Windows 10 that they have been tested to run with. This signing is checked the first time that the driver is installed; once you have a working driver you can update Windows 10 and it will continue to work, even if the driver is not signed for the new version of Windows 10. However, if you install on a new version of Windows 10 64-bit that has not seen a 1401 before you will need an up to date driver.

If the Spike2 installation detects that you are installing onto a newer version of Windows 10 than the driver is signed for it will warn you that this could be a problem. If the installed Spike2 cannot communicate with the 1401 (no device driver found), you should update the 1401 support using the 1401 Windows installer link on our web site.

1401 firmware revision

If the 1401 monitor or 1401 firmware is not the most recent at the time this version of Spike2 was released, an asterisk follows the version. If it is so old that it compromises data sampling, two asterisks follow the version.

All 1401 interfaces supported by Spike2 version 11 have firmware in flash memory. Flash updates and instructions for applying them are available as downloads from the CED web site; you can update the flash firmware without opening the 1401 case. There are downloads for all current 1401 types on our web site.

To update to the latest firmware, you should follow the website link for your 1401, go to the download and follow the detailed instructions to update your 1401. The update is done by the Try1401 program that is installed at the same time as Spike2, so you already have all the tools you need to complete the task.

1401: Monitor or firmware too old or interface problem

This message appears if we detect that the 1401 monitor ROM is too old to run Spike2 safely or if there is a problem with 1401 communications such that the 1401 is detected but communications are garbled.

Working set size

There is information about the Working Set Size at the bottom of the About box. The two numbers describe the minimum and maximum physical memory that the operating system allows Spike2 to use when memory is running low. If the kB is followed by an asterisk (kB*), this means that it is likely that you do not have the *SeIncreaseWorkingSetPrivilege*. If you suffer from error -544 when you sample data, these numbers are important. See the Technical Support chapter for more information.

Computer name

This is the name of your computer that can be used by a Talker on a different computer on the same network to connect to Spike2.

15: Script language

Script language

Introduction to scripting

For many users, the interactive nature of Spike2 may provide all the functionality required. This is often the case where the basic problem to be addressed is to present the data in a variety of formats for visual inspection with, perhaps, a few numbers to extract by cursor analysis and a picture to cut and paste into another application. However, some users need analysis of the form:

1. Find the peak after the second stimulus pulse from now.
2. Find the trough after that.
3. Compute the time difference between these two points and the slope of the line.
4. Print the results.
5. If not at the end, go back to step 1.

This could all be done manually, but it would be very tedious. A script can automate this process, however it requires more effort initially to write it. A script is a list of instructions (which can include loops, branches and calls to user-defined and built-in functions) that control the Spike2 environment. You can create scripts by example, or type them in by hand.

Hello world

Traditionally, the first thing written in every computer language prints “Hello world” to the output device. Here is our version that does it twice! To run this, use the File menu New command and select Script Document, then click OK to create a new, empty script window. Type or copy the following into the script window:

```
Message("Hello world");
PrintLog("Hello world");
```

Click the  `Run` button to check and run your first script. If you have made any typing errors, Spike2 will tell you and you must correct them before the script will run. The first line displays “Hello world” in a box and you must click on OK to close it. The second line writes the sane text to the Log view. Open the Log view to see the result (if the Log window is hidden you should use the Window menu Show command to make it visible).

So, how does this work? Spike2 recognises names followed by round brackets as a request to perform an operation (called a *function* in computer-speak). Spike2 has more than 400 built-in functions, and you can add more with the script language. You pass the function extra information inside the round brackets. The additional information passed to a function is called the function *arguments*.

Spike2 interprets the first line as a request to use the function `Message()` with the argument `"Hello world"`. The message is enclosed in double quotation marks to flag that it is to be interpreted as text, and not as a function name or a variable name.

An argument containing text is called a *string*. A string is one of the three basic data types in Spike2. The other two are *integer* numbers (like 2, -6 and 0) and *real* numbers (like 3.14159, -27.6 and 3.0e+8). These data types can be stored in *variables*.

Spike2 runs your script in much the same way as you would read it. Operations are performed in reading order (left to right, top to bottom). There are also special commands you can insert in the script to make the it run round loops or do one operation rather than another. These are described in the script language syntax chapter.

Spike2 can give you a lot of help when writing a script. Move the text caret to the middle of the word `Message` and press the `F1` key. Help for the `Message()` command appears, and at the bottom of the help entry you will find a list of related commands that you might also find useful.

Views and view handles

Within the script language we refer to Spike2 application windows and files through *View Handles*. A view handle is an integer number that identifies a view. Views fall into one of the following categories:

- Data views: Time, result and XY windows
- Text views: Script, output sequencer, text and log windows
- Grid views: A rectangular array of cells that hold text (usually for tables)
- External views: text and binary files on disk without a window
- Other views: the Spike2 application window, dialog based views, Info windows and toolbars

The first three categories of view are special in that when the user clicks on them to make them active (see below), the menus available for user interaction change to match the view. One of these categories of view is always active.

Whenever you use a built-in function that creates a new view, the function returns the view handle of the new view. View handles run from 1 to 32767; each time you open a new view you get a different handle until all 32767 have been used, at which point we start from the lowest unused handle again. The only exception to this is when you set a sampling configuration to automatically sample a sequence of data files; in this case, all files in the sequence share the same view number (only one of these files is open at a time). The view handle is used with the `View()` command to set the current view and with the `FrontView()` commands to set the active view and the view that should be on top of all windows.

All views have a *parent* window and an *owner* window. A child window is positioned relative to its parent and is usually only visible within the parent rectangle. The parent and owner are often the same thing. For example, a Time view is a child of the application (so the application window is the parent) and it is owned by the application. However, a spike shape dialog is owned by a Time view, but is a child of the Windows Desktop (so it can be moved anywhere on the Desktop, independently of the position of the Spike2 application). You can find the view handle of the owner of a view with the `ViewLink()` command.

There is also a concept of the *input focus*. The view with the input focus gets the first shot at deciding what to do with keyboard characters and mouse messages. Use `FocusHandle()` to get the view with the input focus.

The current view

There is always a *current view*. The current view is important to the script language as it is the view that script commands are applied to unless the script command defines a different view. There is always a current view. Even if you close all windows the Log view, used for text output by the `PrintLog()` command, remains.

You get the view handle of the current view with the `View()` command with no arguments; use the `View(handle%)` command to set the current view. Do not confuse the current view (which is the view that script commands will use) with the Active view (the one with the highlighted title bar, usually the last window where the user clicked the mouse). The current view and the Active view will often be the same, for example after the `FrontView()` script command has set it, but can be different, for example after the `View()` command has been used.

Whenever a script creates a new view, it becomes the current view. Views are usually created invisibly so that they can be configured before appearing (which prevents unsightly screen flashes). You can use `WindowVisible(1)` to display a new window.

When debugging a script, you can see the current view handle by opening the Global variables window.

The front (active) view and input focus

The `FrontView()` script command with no arguments returns the Time, Result, XY, Grid or text-based view that is the *Active* view of the application. The Active view has a highlighted title bar and if the application has the input focus, either it or one of its children or a view it owns has the input focus. The Active view determines the contents of the Spike2 main menu. It determines which set of menus are visible to the `MenuCommand()` script command.

You can also use the `FrontView(handle%)` command to bring any script-controllable view to the front (or as near to the front as is possible) and make it the current view. The `FocusHandle()` command reports the handle of the view with the input focus. If you use the `FrontView(handle%)` command to bring a Time,

Result, XY, Grid or text-based view to the front, this also makes it the active view, just as if the user had clicked the mouse on it.

The Spike2 application window

This view always exists and the handle can be obtained with the `App(0)` command. You can use this view handle to position, show and hide the application window. You cannot close this window with `FileClose()`, use the `FileQuit()` command to do this. The application window is a top-level window and is a child of the Desktop.

Time, result, XY, Grid and text-based views

These are all views with an associated document that can be saved as a file on disk. They are special in the sense that one of them is always the Active view of the application. Use `FrontView()` to return the Active view.

Text-based views are: simple text windows, the Log view, output sequencer views and script views. The running script view is hidden from most script commands, however you can obtain its view handle with `App(3)` so you can show and hide it. Closing the Log view hides it as it always exists. You can get the Log view handle with `LogHandle()`.

You can see a list of Time, Result, XY, Grid and text-based view handles by using the Window menu Windows... command.

External views

These are used by scripts to store text and binary data and have no associated window. They can be the script current view but can never be the application Active view as they have no visible window. External text views can be useful when you want to write a lot of text line by line or read a text file line by line to process it. External binary files allow random access to their contents and allow you to create and read arbitrary format files (but you have to have detailed knowledge of the contents).

Dialog-based views and toolbars

These cover all the script-controllable views that do not fall into one of the other categories: Cursor windows (see `CursorOpen()`), Multimedia views (see `MMOpen()`), Spike shape views (see `SSOpen()`), the Spike monitor window (see `SMOpen()`), the sampling control panel, Sequencer control panel and the Sample Status bar (see `SampleHandle()`) and Info windows. The `App()` command gives you access to: the System toolbar, the status bar, the Edit toolbar, the Play wave bar, the Script bar, the Sample bar, the sample control panel, the Sequencer control panel and the Sample Status bar.

Writing scripts by example

To help you write scripts Spike2 can monitor your actions and write the equivalent script. This is a great way to get going writing scripts, but it has limitations (and not all actions get recorded). Scripts generated this way only repeat actions that you have already made. The good point of recording your actions is that Spike2 shows you the correct function to use for each activity.

For example, let us suppose you use the Script menu Turn Recording On option, open a data file, select interval histogram analysis mode of channel 3, process all data in the file and end with the Script menu Stop recording command. Spike2 opens a new window holding the equivalent script (we have tidied the output up a little and added comments):

```
var v6%,v7%;      'declare integer variables to hold view handles
v6% := FileOpen("demo.smr", 0, 3);    'Open file, save view handle
Window(10, 10, 80, 50);               'Set a window position
v7% := SetInth(3, 100, 0.01, 0);      'Create invisible INTH view
WindowVisible(1);                     'make the INTH visible
Process(0, View(-1).Maxtime(), 0, 1); 'Add data to INTH view
```

Some of this is fairly straightforward. You can find the `FileOpen()`, `SetInth()`, and `Process()` functions described in this manual and they seem to map onto the actions that you performed. However, there is extra scaffolding holding up the structure.

In the first line, the `var` keyword creates two integer variables, `v6%` and `v7%`. These variables hold view handles returned by `FileOpen()` and `SetInth()`. Spike2 generates the names from the internal view numbers (so

your script may not be exactly the same). The result of the functions is copied to the variable with the *:= assignment operator*. In English, the second line of the script could be read as: *Variable v6% is assigned the result of the FileOpen command on file demo.smr* .

The `SetInth()` command makes a new window to hold an interval histogram of the data in channel 3 with 100 bins of 0.01 seconds width and with the first bin starting at an interval of 0 seconds. The `WindowVisible(1)` command is present because the new window created by `SetInth()` is hidden. Spike2 creates invisible windows so that you can size and position them before display to prevent excessive screen repainting.

The `View(-1).` syntax accesses data belonging to a view other than the current view. The current view when the `Process()` command is used is the result view and we want to access the maximum time in the original time view. The negative argument tells the script system that we want to change the current view to the time view associated with this result view. The dot after the command means that the swap is temporary, only for the duration of the `Maxtime()` function.

Using recorded actions

You can now run this script with the Run button at the top right of the script window. The script runs and generates a new result view, repeating your actions. Now suppose we want to run this for several files, each one selected by the user. You must edit the script a bit more and add in some looping control (`while...wend`). The following script suggests a solution. Notice that we have now changed the view handle variables to names that are a little easier to remember.

```
var fileH%, inthH%;              'view handle variables
fileH% := FileOpen("", 0, 1);    'blank for dialog, single window
while fileH% > 0 do              'while fileH% is greater than 0
   inthH% := SetInth(3, 100, .01, 0);WindowVisible(1);
   Process(0, View(-1).Maxtime(), 0, 1);
   Draw();                       'Update the INTH display
   fileH% := FileOpen("", 0, 1); 'ask for the next file
wend;                            'The end of the while loop
```

This time, Spike2 prompts you for the file to open. The file identifier is negative if anything goes wrong opening the file, or if you press the Cancel button. We have also included a `Draw()` statement to force Spike2 to draw the data after it calculates the interval histogram. There is a problem with this script if you open a file that does not contain a channel 3 that holds events or markers of some kind. We will deal with this a little later.

However, you will find that the screen gets rather cluttered up with windows. We do not want the original window once we have calculated the histogram, so the next step is to delete it. We also have added a line to close down all the windows at the start, to reduce the clutter when the script starts.

```
var fileH%, inthH%;
FileClose(-1);                   'close all windows except script
fileH% := FileOpen("", 0, 1);    'use a blank name to open dialog
while fileH% > 0 do              'FileOpen returns -ve if no file
   inthH% := SetInth(3, 100, .01, 0);WindowVisible(1);
   Process(0, View(-1).Maxtime(), 0, 1);
   View(fileH%).FileClose();     'Shut the old window
   Draw();                       'Update the INTH display
   fileH% := FileOpen("", 0, 1); 'ask for the next file
wend;
```

This seems somewhat better, but we still have the problem that there will be an error if the file does not hold a channel 3, or it is of the wrong type. The solution to this is to ask the user to choose a channel using a dialog. We will have a dialog with a single field that asks us to select a suitable channel:

```
var fileH%, inthH%, chan%;       'Add a new variable for channel
FileClose(-1);                   'close all windows to tidy up
fileH% := FileOpen("", 0, 1);    'use a blank name to open dialog
while fileH% > 0 do              'FileOpen returns -ve if no file
   DlgCreate("Channel selection"); 'Start a dialog
   DlgChan(1, "Choose channel for INTH", 126); 'all but waveform
   if (DlgShow(chan%) > 0) and   'User pressed OK and...
      (chan% > 0) then           '...selected a channel?
      inthH% := SetInth(chan%, 100, .01, 0);WindowVisible(1);
      Process(0, View(-1).Maxtime(), 0, 1);
```

```
      View(fileH%).FileClose();  'Shut the old window
      Draw();                    'Update the display
   endif
   fileH% := FileOpen("", 0, 1); 'ask for the next file
wend;
```

The `DlgCreate()` function has started the definition of a dialog with one field that the user can control. The `DlgChan()` function sets a prompt for the field, and declares it to be a channel list from which we must select a channel (or we can select the No channel entry). The `DlgShow()` function opens the dialog and waits for you to select a channel and press OK or Cancel. The `if` statement checks that all is well before making the histogram.

Recording sampling

If you record opening a file for sampling, Spike2 will generate the code that creates a sampling configuration for the channels used by the 1401 and for any online processes that generate channels, Result and XY views. The following is the result of turning recording on in the Script menu, using the main toolbar Sample now command (or the File menu New command and selecting a data file), then clicking the Start sampling button and then the Stop sampling button and finally turning recording off in the Script menu:

```
var v26%; 'Data view created by FileNew()
var v11%:= LogHandle();
FrontView(v11%);

SampleClear(); 'Set standard state (32 channels, 64-bit smrx file)
SampleOptimise(0, -1, 9); 'No optimise, set 1401 type, (do first)
SampleUsPerTime(10);
SampleTimePerAdc(1000);
SampleStartTrigger(-1); 'Set trigger to start sampling
SampleSeqCtrl(1, 0); ' Seqencer jump control

'Channel recording definitions
SampleWaveform(1, 0,100); ' chan, port, ideal rate
SampleDerived(2, 1); 'Derived from channel 1
SampleProcess(2, 0, 0, 0, 6, 10.0365); 'IIR Low pass Butterworth 10.04 Hz, sixth order
SampleCalibrate(2," Volt",0.5, 0); 'scale and offset
SampleComment$(2,"Derived: ");
SampleTextMark(80);
SampleTitle$(30,"Title30");
SampleComment$(30,"Comment30");
SampleTitle$(31,"Title31");
SampleComment$(31,"Comment31");
SampleOptimise(1,1,9,1,50);  'Set optimise mode (do this last)
SampleMode(1); 'Continuous sampling

'Create data file without resources (mode=1). Set mode=3 and
'delete any procedure definitions below to use resources.
v26%:=FileNew(0, 1); 'Create Data view
View(SampleHandle(1)).WindowVisible(1); 'Show sampling controls

'Process setup
MeasureX(102, 0, "Cursor(0)", "0"); ' The time of the item
MeasureY(100, 1, "Cursor(0)", "0");
ChanDelete(2001); 'Ensure target channel unused
MeasureToChan(2001, "Measure 1", 7, 4, 1, 0, "0.0", 0, 1, "", 0);
ProcessAuto(1, 0, 1, 0, 0.5, 2001);

Window(0, 0,100, 80.0216);

SampleStart();  'Begin sampling immediately
'Time delays are not recorded
SampleStop();
```

What gets recorded will depend on the current sampling configuration. To use this code you would need to insert some kind of time delay between the `SampleStart()` and `SampleStop()`. The easiest method (for the purpose of demonstration) is to add a `Yield(secs);` command (replace `secs` by the desired sampling

duration). Normally one would use the Toolbar to give the user control over sampling while allowing sampling to continue. This code has generated a template for setting up the sampling configuration, but recording has its limitations and the generated script is verbose.

Another way to do this is to save the desired sampling configuration to a `.s2cx` file and load it:

```
var ok%:=FileOpen("E:\\Users\\Fred\\Documents\\100HzLoPass.s2cx",6);
if ok% then Message("Failed to open configuration"); halt endif;
var sh%:=FileNew(0, 3); 'Create Data view visible, use configuration
if sh% < 0 then Message("Failed to open sampling file"); halt endif;

XRange(0, 3);          'Set data range to display
Window(0, 0, 100, 80); 'Position window
SampleStart();         'Begin sampling immediately
Yield(3);              'Lazy way of waiting (should use Toolbar)
SampleStop();          'Stop
```

This example loads a sampling configuration file and creates a new data file based on it by setting the `mode%` argument in `FileNew(0, mode%)` to 3 (make view visible, use loaded configuration and resources). You can adjust the sampling configuration after loading it by adding script commands between the loading of the configuration file and the creation of the data file.

Differences between systems

If you need to write code that can run on different systems, you should try to avoid system dependent features, or if this is impossible, make use of the `System()` function to find out where you are. For example, the Macintosh versions of Spike2 stopped at version 2, so if you use any Spike2 version 3 or later features, your script will not run on a Macintosh.

In the Windows environment, beware of restrictions on where files and programs can be stored. To improve security, Microsoft have imposed more restrictions on the use of features and folder with each version of Windows.

Notation conventions

Throughout this manual we use the font of this sentence to signify descriptive text. Function declarations, variables and code examples print in a monospaced font, for example `a := View(0)`. We show optional keywords and arguments to functions in curly braces:

```
Func Example(needed1, needed2 {,opt1 {,opt2}});
```

In this example, the first two arguments are always required; the last two are optional. Any of the following would be acceptable uses of the function:

```
a := Example(1,2);       'Call omitting the optional arguments
a := Example(1,2,3);     'Call omitting one argument
a := Example(1,2,3,4);   'Call using all arguments
```

A vertical bar between arguments means that there is a choice of argument type:

```
Func Choice( i%|r|str$ );
```

In this case, the function takes a single argument that could be an integer, a real or a string. The function will detect the type that you have passed and may perform a different action depending upon the type.

Three dots (`...`) stand for a list of further, similar items. It is also used when a function can accept an array with any number of dimensions:

```
Func Sin(x|x[]{[]...});
```

This means that the function `Sin()` will accept a real value or and array with one or more dimensions.

Other sources of script information

This continues with information about the script window and debugging, followed by a reference for the script language and then documentation for the built-in script functions, first with functions grouped by topic, and then a full alphabetical list.

When you are in the script editor, there are several features designed to help you write script code. These include pop-up help tips, right-clicks to take you to the help for a specific function, pop-up help when adding arguments to functions, automatic formatting and Auto-complete of typed names.

There are example and utility script provided with Spike2. These are copied to the `scripts` folder within the folder that contains Spike2.

Our web site at `ced.co.uk` has example scripts and script updates that you can download.

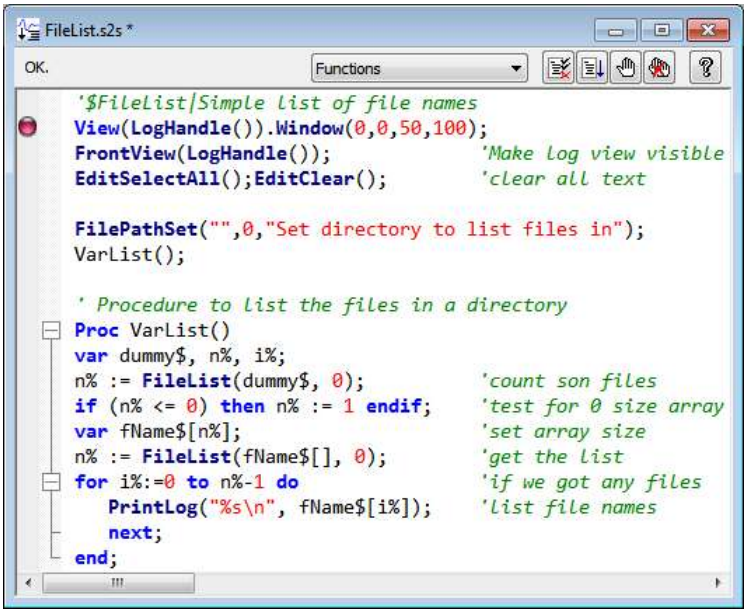
There is also a manual that has been used for our Spike2 training courses, held at CED and around the world. This Training Day manual contains many annotated examples and tutorials. Some of the scripts in this manual are useful in their own right; others provide skeletons upon which you can build your own applications. You can get a PDF of this manual from our web site on the Downloads page (follow the links Software manuals, Spike2 V10, Training).

Script window and debugging

You use the script window when you write and debug a script. Once you are satisfied that your script runs correctly you would normally run a script from the script menu without displaying the source code. You can have several scripts loaded at a time and select one to run with the Script menu Run Script command.

Script editor

If a script is read from a read only medium or is write protected you cannot make changes to it in the Spike2 script editor.



The script window is a text window with a few extra controls including a folding margin that allows you to fold away inner loops, function and procedures. The folding margin can be hidden; see the Edit menu Preferences for details of configuring the script view folding margin.

To the left of the text area is a margin where you can set break points (one is shown already set), bookmarks and where the current line of the script is indicated during debugging. Above the text area is a message bar and several controls. The controls have the following functions:

This control is a quick way to find any `func` or `proc` in your script. Click on this to display a list, in alphabetical order, of the names of all user-defined routines in the current script and in files included by it. Select one, and the window will scroll to it. To be located, the keywords `func` and `proc` must be at the start of a line and the name of the routine must be on the same line. From version [10.08], the box displays the name of

the user-defined `func` or `proc` that contains the text caret. You can use a comment that starts `'!` to hide a routine from the Function list and from generating pop-up help.

Compile

The script compiler checks the syntax of the script and if no errors are found it creates the compiled version, ready to run. If the script has not been changed since the last compile and no other script has been compiled, the button is disabled as there is no need to compile again. Spike2 can have one compiled script in memory at a time.

Run

If the script has not been compiled it is compiled first. If no errors are found, Spike2 runs the compiled version, starting from the beginning. Spike2 skips over `proc ... end;` and `func ... end;` statements, so the initial code can come before, between or after any user-defined procedures and functions. This button is disabled once the script has started to run.

Set break point

This button sets a break point on the line containing the text caret or clears a break if one is already set. A break point stops a running script when it reaches the start of the line containing the break point. You can also set and clear break points by clicking the mouse pointer in the margin or by right clicking the line and using the Toggle break command. A break point is indicated by a red marker in the gutter (between the optional line number and folding margins). You can set and clear break points by clicking in the gutter, or using the context menu (right-click) in a script view.

Not all statements can have break points set on them. Some statements, such as `var`, `const`, `func` and `proc` compile to entries in a symbol table; they generate no code. If you set a break point on one of them the break point will appear at the next statement that is "breakable". If you set break points before you compile your script, you may find that some break points move to the next breakable line when you compile.

Once a script has been compiled, all breakpoints in it and in any included files are remembered. You can close and open the files and the break points will still be visible. However, break points are reassessed when you click the run button to start a script. This means that if you want to break in an included file you must have the file open and the break point set when you hit run. Once the script has started to run you can close any or all of the script files and break points will still work and automatically open the source file when they are hit.

Clear all break points

This button is enabled if there are break points set in the script. Click this button to remove all break points from the script. Break points can be set and cleared at any time, even before your script has been compiled.

Help

This button provides help on the script language. It takes you to an alphabetic list of all the built-in script functions. If you scroll to the bottom of this list you can also find links to the script language syntax and to the script language commands grouped by function. Within a script, you can get help on keywords and built in commands by clicking in the keyword or command and pressing the `F1` key or by right-clicking on a name and selecting Help for name from the context menu.

You can also get a pop-up call tip to appear if you hover the mouse pointer over a user-defined or built-in `proc` or `func` name. If you hover over the name of a `var` or `const` item, the pop-up text displays the line that defined it unless it was hidden.

Go to user-defined proc or func or to a var or const definition

You can navigate to the start of a user-define `proc` or `func` or `var` or `const` definition by right-clicking on the name and selecting Go to name from the context menu. This works, even if the named item is defined in an included file (as long as the included file can be located). Items can be hidden from this system by adding a comment that starts `'!` to the item definition.

Go to next or previous proc or func

If you right-click in the text, and no name is found to navigate to, you are offered the next and previous line that starts with a `Proc` or `Func` declaration. This can be useful in large user-defined functions as it lets you jump quickly to the start or end.

Syntax colouring

Spike2 supports syntax colouring for both the script language and also for the output sequencer editor. You can customise the colouring (or disable it) from the Script files settings section of the Edit menu Preferences in the General tab. The language keywords have the standard colour blue, quoted text strings have the standard colour red, and comments have the standard colour green. You can also set the colour for normal text (standard colour black) and for the text background (standard colour white).

The syntax colouring options are saved in the Windows registry. If several users share the same computer, they can each have their own colouring preferences as long as they log on as different users.

Editing features for scripts

There are some extra editing features that can help you when writing scripts. These include automatic formatting, commenting and un-commenting of selected lines, indent and outdent of code, code folding, auto-complete of typed words, marking of changed text and pop-up help for built-in and user-defined functions. These are described in the documentation for the Edit menu.

Debug overview

Despite all our efforts to make writing a script easy, and all your efforts to get things right, sooner or later (usually sooner), a script will refuse to do what you expect. Rather than admit to human error, programmers attribute such failures to “bugs”, hence the act of removing such effects is “debugging”. The term dates back to times when an insect in the high voltage supply to a thermionic valve really could cause hardware problems.

To make bug extermination a relatively simple task, Spike2 has a “debugger” built into the script system. With the debugger you can:

- Step one statement at a time
- Step into or over procedures and functions
- Step out of a procedure or function
- Step to a particular line
- Enter the debugger on a script error to view variable values
- View local and global variables
- Watch selected local and global variables
- Edit variable values
- See how you reached a particular function or procedure
- Set and clear break points

With these tools at your disposal, most bugs are easy to track down.

Preparing to debug

A Spike2 script does not need any special preparation for debugging except that you must set a break point or include the `Debug()` command at the point in your script where you want to enter the debugger. Alternatively, you can enter the debugger “on demand” by pressing the `Esc` key; you may need to hold it down for a second or two, depending on what the script is doing. If the `Toolbar()`, `DlgShow()` or `Interact()` commands are active, hold down `Esc` and click on a button. This is a very useful if your script is running round in a loop with no exit! You can disable this with the `Debug(0)` command, but we suggest that this feature is added after the script has been tested! Once you have disabled debugging, there is no way out of an infinite loop.

You can also choose to enter debug on a script error by checking the Enter debug on script error box in the General tab of the Edit menu Preferences option. Depending on the error, this may let you check the values of variables to help you fix the problem.

When your script enters the debugger, the debug toolbar opens if it was not visible and debug windows that were open when the last debug session ended are restored. The picture shows the toolbar as a floating window, but you can dock it to any side of the Spike2 window by dragging it over the window edge and releasing.

Stop running the script (`Ctrl+Alt+Shift+F5`). There is no check that you really meant to do this; we assume that if you know enough to open the debugger, you know what you are doing! You can use the `Debug()` command to disable the debugger. This also cancels any chained script set with `ScriptRun()`.

Display the current script line (`Ctrl+Alt+F2`). If the script window is hidden, this makes it visible, brings it to the top and scrolls the text to the current line.

If the current statement contains a call to a user-defined `Proc` or a `Func`, step into it, otherwise just step (`Ctrl+F5`). This does not work with the `Toolbar()` command which is not user-defined, but which can cause user-defined routines to be called. To step into a user-defined `Func` that is linked to a `Toolbar()` command, set a break point in the `Func`.

Step over this statement to the next statement (`Shift+F5`). If you have more than one statement on a line you will have to click this button once for each statement, not once per line.

Step out of a procedure or function (`Alt+F5`). This does not work if you are in a function run from the `Toolbar()` command as there is nowhere to return to. In this case, the button behaves as if you had pressed the run button.

Run to cursor, or more accurately to the start of the line with the text caret (`Ctrl+Alt+F2`). This is slightly quicker than setting a break point, running to it, then clearing it (which is how this is implemented). This button is disabled unless the active view is a script file that is part of the currently running script.

Run the script (`Ctrl+Shift+F5`). This disables the buttons on the debug toolbar and the script runs until in reaches a break point or the end of the script.

Show the local variables for the current user-defined `func` or `proc` (`F7`). If there is no current routine, the window is empty. You can edit a value by double clicking on the variable. Elements of arrays are displayed for the width of the text window. If an array is longer than the space in the window, the text display for the array ends with … to show that there is more data. You can right-click on a variable to add it to the Watch window.

Show the global variable values in a window (`Ctrl+F7`). You can edit a global variable by double clicking on it. The very first entry in this window lists the script current view by handle, type and window name. You can right-click on a variable to add it to the Watch window.

Display the call stack (list of calls to user-defined functions with their arguments) on the way to the current line in a window (`Ctrl+Shift+F7`). If the `Toolbar()` function has been used, the arguments for it appear, but the function name is blank.

Open the Watch window. This displays globals and locals that were selected in the global or local variables windows. Local variables are displayed with values when the user-defined `Proc or Func` in which they are defined is active.

Displays a list of objects that are currently opened in a debug window. Objects are opened by double-clicking them in a Local, Global or Watch window. Up to 10 object windows can be open at once.

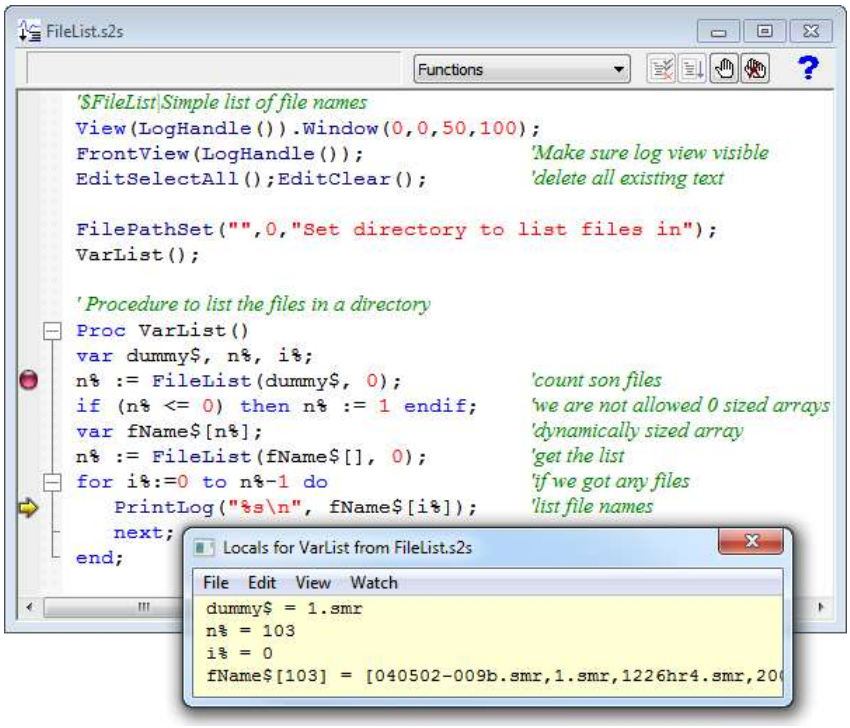
The debug toolbar and the locals, globals, watches and the call stack window close at the end of a script; Spike2 remembers their positions and will restore them the next time they open. If you move one of these windows off screen, click the button to activate them again to move them to the nearest visible position.

The buttons in the debug toolbar are disabled if they cannot be used. If you forget what a particular button does, move the mouse pointer over the button. A “Tool tip” window will open next to the button with a short description; if the Status bar is visible, a longer description can be seen there.

Enter debug on error

There is an option in the Edit menu Preferences dialog General tab that allows you to enter the debugger if an error occurs. After an error you can inspect the values of local and global variables and the contents of the call stack, but you are not allowed to continue running the script.

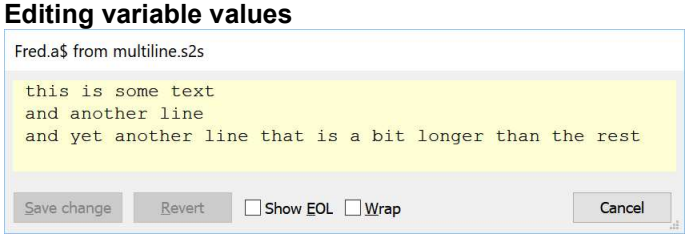
Inspecting variables



If the Watch, Object, Locals or Globals windows are open, they display a list of variables. The Globals window also displays information about the current view. If there are more variables than can fit in the window you can scroll the list up and down to show them all. Simple variables are followed by their values. If you double click one, a new window opens in which you can edit the value of the variable. Variables in the local, global and object windows are displayed in the order they are declared in the script. Variables in the watch window are displayed in the order that you add them, but you can use the Watch menu to sort them into alphabetic order.

If you double click on an array, a new window opens that lists the values of the elements of the array. You choose the element by editing the index fields, one for each dimension.

Editing variable values

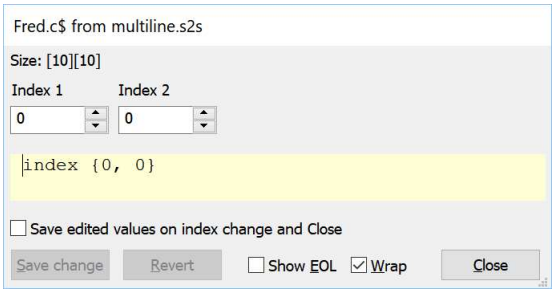


When you edit a numeric value, the edit window is always a single line edit. If you edit a string value, the edit window will attempt to resize to make the entire string visible, and can cope with multi-line strings. There is a limit on the length of a string that can be modified (currently 512 characters). There are several controls you can use:

Show EOL Strings only. Check this box to display the end of line character(s). Script strings that are created using `\n` to mark the end of line will display the LF character marking. You can add more line

breaks by typing `Ctrl+Enter` to add more LF characters or `Enter` to add CR LF character pairs (equivalent to `\r\n` in a script).

Wrap	Strings only. If this box is checked, strings that do not fit in the text edit box will wrap around, otherwise they run off the end of the edit window. If the string selected for editing is a single line, the Wrap box is initially checked. If the text contains any line end characters, the Wrap box is initially unchecked.
Save change	This button is enabled if you have modified the value. Click the button to save your change.
Revert	This button is enabled if you have modified the value. Click the button to revert to the initial value.



When you are editing an array element, there are extra controls to select the array index. There is also the Save edited values on index change and Close check-box. If you do not check this box, edits are saved only when you click the Save change button. If the variable is an array of Objects, there is an addition Object button. Click the button to display the current object in a new Object window, allowing you to edit object members.

Menu commands

The variable windows contain a menu with the following commands:

File	Close	Closes the debug window
Edit	Copy	Copy selected text to the clipboard
	Log	Copy selected text to the Log view
	Select All	Select all windows text
	Find...	Open the Find text window
	Find Again	Repeat the last find operation
	Find Last	Repeat last find operation searching backwards
	Toggle Bookmark	Set/clear bookmark in Global variable window
	Next Bookmark	Jump to the next bookmark in the Global variable window
	Previous Bookmark	Jump to previous bookmark in the Global variable window
	Clear All Bookmarks	Remove all bookmarks from the Global variable window
View	Font	Open the Font dialog to change the window settings
	Hide const	Remove items declared as `const` from the display
	Names only	In Locals and Object windows, only display item names, not the `Object` or `Proc`/`Func` name.
Watch	Add to the Watch window	Add selected Global and Local items
	Delete from the Watch window	Remove selected Watch window item
	Delete all watched variables	Remove all items from the Watch window
	Delete 'Not found' variables	Remove watched items that are not in the current script
	Sort variables into alphabetic order	Sort Watch window items into alphabetic order

Most commands have keyboard short cuts listed in the menu. The Watch menu items are also available on a right-click context menu, where appropriate.

Watch window

You can add variables to the watch window by right-clicking on them in the locals, globals or object windows and choose the option to copy the selected variables to the watch window. In the watch window, right-click to

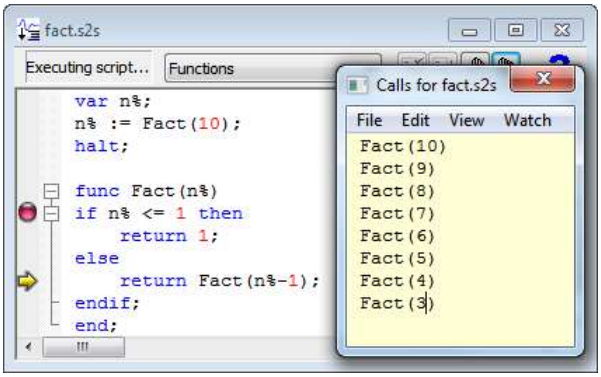
see available options to control the watched variables. The watch window remembers the watched variables between debugging sessions. If a variable does not exist in the current script, it is still remembered, but is marked as not existing.

Spike2 remembers the list of watched variables between sessions.

Object windows

If you double click on an object displayed in a window, or click the Object button in the array edit window, a new window opens holding the object members. You can have up to 10 object windows open at the same time.

Call stack



The call stack can sometimes be useful to figure out how your script arrived at a position in your code. This is particularly true if your script makes recursive use of functions. A function is recursive when it calls itself, either directly, or indirectly through other functions. Recursive functions can be very useful, but must be used with care.

A tricky fault to diagnose is a script that has mutually recursive functions. If there is no way out, the script gets deeper and deeper into the call stack until it runs out of memory. The call stack window can help to detect such problems. The example to the right demonstrates the use of a recursive function to generate a factorial (but see `LnGamma()` and `BinomialC()` for large factorials). Excessive call stack use is trapped and will stop a runaway recursive script (unless it runs out of memory before hitting the stack limit).

Script language syntax

A Spike2 **script** is composed of lines of text. Each line can be up to 511 characters long (was 240 before [10.18]). This is an arbitrary limit - most lines will be much shorter. Scripts are usually written in the script editor, but can be written with any editor and imported. The script language keywords, numbers and names of variables and constants use ASCII characters. However, comments and strings can include Unicode characters.

A script consists of program **statements**. Statements that generate looping or branching constructs include other statements within them. A statement is terminated by a semicolon or by a keyword that is part of an enclosing statement. Statements are not terminated by an end of line character, so a statement can be spread over multiple lines. Two semicolons in a row generate an empty statement, which generates no code.

Within a statement, **white space** of any length is treated as a single space character, and white space between script tokens is ignored unless is it is required to separate two tokens. White space consists of the space and tab characters and end of line characters.

A **token** is an indivisible entity such as a script keyword, the name of a constant, variable, procedure or function or a numeric or string constant. White space can be added anywhere without changing the meaning of a script except in the middle of a token.

Keywords and names

All keywords, user-defined functions and variable names in the script language start with one of the letters `a` to `z` or `A` to `Z` or `_` (underscore) followed by the characters `a` to `z`, `A` to `Z`, `0` to `9` and underscore. Keywords and names are not case sensitive (so `Fred` and `fred` are equivalent), however users are encouraged to be consistent in their use of case as it makes scripts easier to read. We reserve starting a symbol name with underscore for CED use. There is nothing to stop you using it, but we will use a leading underscore for future constant names, so your use might collide with ours. Underscore was added as an acceptable character at Spike2 versions [8.03, 7.16].

Variables and user-defined functions use the characters `%` and `$` at the end of the name to indicate integer and string type. The `%` or `$` is considered part of the name.

User-defined names can extend up to a line in length. Most users will restrict themselves to a maximum of 20 or so characters.

The following keywords are reserved and cannot be used for variables or function names:

```
and    band     bor    break   bxor    case
const  continue diag   do      docase  else
end    endcase  endif  for     func    halt
if     mod      next   not     object  objend
or     proc     repeat resize  return  step
then   to       trans  until   var     view
wend   while    xor
```

You cannot use the name **`self`** inside a user-defined object member function as this name is reserved for the reference to the object instance passed automatically as the first argument of the member function.

Further, names used by Spike2 built-in functions cannot be redefined as user functions or global variables. They can be used as local variables (not recommended).

Data types

There are three basic data types in the script language: real, integer and string. The real and integer types store numbers; the string type stores characters. Integer numbers have no fractional part, and are useful for indexing arrays or for describing objects for which fractions have no meaning. Integers have a limited (but large) range of allowed values.

Real numbers span a very large range of number and can have fractional parts. They are often used to describe real-world quantities, for example the weight of an object.

Strings hold text and automatically grow and shrink in length to suit the number of text characters stored within them.

You can also define your own types with the `object` keyword.

Real data type

This type is a double precision floating point number. Variables of this type have no special character to identify them. Real constants have a decimal point or the letter `e` or `E` to differentiate from integers. White space is not allowed in a sequence of characters that define a real number, so a number cannot start on one line and end on another). Real number constants have one of the following formats where `digit` is a decimal digit in the range 0 to 9:

```
{-}{digit(s)}digit.{digit(s)}{e|E{+|-}digit(s)}
{-}{digit(s)}.digit{digit(s)}{e|E{+|-}digit(s)}
{-}{digit(s)}digitE|e{+|-}digit(s)
```

Real numbers have a precision of more than 16 decimal digits. The maximum magnitude that can be stored is `1.7976931348623158e+308`. If your script has a floating point literal value that exceeds the maximum magnitude, the compiler will flag an error.

The minimum non-zero magnitude is `2.2250738585072014e-308`. The smallest number that can be added to 1.0 so that the result is not 1.0 is `2.2204460492503131e-016`. There is no limit on the number of digits after the decimal point. However, supplying more than 17 digits of precision has no effect.

The following are legal real numbers:

```
1.2345 -3.14159 .1 1. 1e6 23e-6 -43e+03
```

`E` or `e` followed by a power of 10 introduces exponential format. The last three numbers above are: `1000000 0.000023 -43000.0`. The following are not real constants:

`1 e6`	White space is not allowed	`1E3.5`	Fractional powers are not allowed
`2.0E`	Missing exponent digits	`1e500`	The number is too large

Technical stuff

Real numbers in Spike2 are stored in the double format defined in IEEE 754-1985, which is a standard format used in most computers. Each number uses 64 bits, of which 11 hold an exponent, there is one sign bit, and 52 bits hold the magnitude of the data, making 64 bits. The format manages to squeeze one extra bit out of the magnitude by stating that all normalised numbers start with a 1 bit that need not be supplied.

Maintaining precision in real calculations

Unlike integer numeric expressions, which are exact unless you overflow their value range, the precision of a calculation involving real numbers may be problematic. Multiplication and division do not cause any loss in precision other than a rounding effect in the least significant bit of the result. However, addition and subtraction can cause significant loss. The expression: `1e15 + (1.0/3.0) - 1e15` is not evaluated as 0.33333333... as you might hope, but 0.375 and `1e16 + (1.0/3.0) - 1e16` has the value 0. When taking the difference of numbers of similar sizes, the precision of the result is limited by the precision of the larger number.

You can sometimes reduce the precision loss by rearranging expressions. For example the calculation:

```
var c := a * a - b * b;
```

can be rewritten as:

```
var c := (a + b) * (a - b);
```

which has less precision loss when a and b are similar in magnitude.

When comparing real numbers that are the results of calculations, do not test for exact equality. For example, the following loop may run forever:

```
var a := 0.0;
repeat
    a += 1/999.0;   'Beware: 1/999 evaluates as 0 as integer/integer -> integer
    ...
until a = 1.0;
```

The problem is that 1/999.0 cannot be represented exactly in the real number format, the closest that can be stored is 0.0010010010010010010; this is slightly less than the exact result. It is easy to see that after adding this up 999 times, the result is 0.99999999999999999900, which is not 1. Changing the last line to:

```
until a >= 1.0;
```

guarantees that the loop will terminate (although it may run one more time than intended). As a general rule, never use accumulation of a constant inside a loop, use multiplication instead:

```
var a := 0.0, i%;
for i% := 1 to 999 do
    a = i% / 999.0;   'avoids accumulation of rounding errors
    ...
next;
```

Real numbers can store integral values exactly in the range -9007199254740992 to 9007199254740992 (this is 2 to the power 53). `Ceil()`, `Floor()`, `Trunc()` and `Round()` have no effect on real numbers greater in magnitude than 9007199254740992 as they have no fractional part.

Overflow handling

If you overflow the floating point range, the result is set to floating point infinity, displayed as `inf` or `-inf`, depending on the sign. You can compare floating point values with an infinite value and the results will make sense (`inf` is greater than all non-infinite values and `-inf` is less than all non-minus infinite values). Combining infinity with another value using `+`, `-`, `*` or `/` will usually generate `inf`, though multiply and divide by a negative value will change the sign of the infinity. Infinity times 0.0 is `nan` (Not a Number, see below). `inf % value` is also `nan`. Perhaps unexpectedly, `value % inf` is `value`, which is the correct result. also, `value / inf` is 0.0, but `inf / inf` is `nan`.

Not a Number (nan)

A floating point value can be Not a Number. This can occur as a result of expressions involving `inf`, as described above. Some programs use `nan` to indicate missing data, and this can occur when we import data into a RealWave channel. There are special data channel processes you can add to strip out such data.

Underflow

There is another floating point number problem, which occurs when a value gets too small to be represented with full accuracy. This usually occurs as the result of a multiply or divide operation. In this situation we have chosen to ask the floating point system to 'do the best it can', so if a value becomes smaller than `2.2250738585072014e-308` we just keep going and accept that there is a loss of precision (rather than setting the result to 0.0. which is usually a worse choice).

Integer data type

The integer type is identified by a `%` at the end of the variable name and stores 64-bit signed integer (whole) numbers in the range `-9223372036854775808` to `9223372036854775807`. There is no decimal point in an integer number. An integer number has the following formats (where `digit` is a decimal digit 0 to 9, and `hexadecimal-digit` is 0 to 9 or `a` to `f` or `A` to `F`, with `a` to `f` and `A` to `F` standing for decimal 10 to 15, items in braces {} are optional and `x|X` means one of `x` or `X`):

```
{-}{digit(s)}digit
{-}0x|X{hexadecimal-digit(s)}hexadecimal-digit
```

You may assign real numbers to an integer; it is an error to assign numbers beyond the integer range. Real numbers have any fractional part removed before assignment (-1.99 is -1 as an integer). If you assign an integer to a real number you can lose accuracy as real numbers have typically 53 bits of precision. Put another way, if you assign an integer larger than 9007199254740992 (0x20000000000000, 9.007... x 10^{15}) to a real number you will lose precision. When differencing very large integers, use integer arithmetic for maximum accuracy.

The following are examples of integers:

```
1 -1 -2147483647 0 0xfedcba9876543219 0X100 -0xCED
```

Integer limits and older versions of Spike2

Before Spike2 version 8 the script language used 32-bit integers. If you wish your script to run in version 7 you must limit your use of integers to the 32-bit range.

Feature	64-bit	32-bit
Decimal range	`-9223372036854775808` to `9223372036854775807`	`-2147483648` to `2147483647`
Hexadecimal range	`0x8000000000000000` to `0x7fffffffffffffff`	`0x80000000` to `0x7fffffff`
As a power of 10	`-9.223 x 10`^{18} to `9.223 x 10`^{18}	`-2.147 x 10`^{9} to `2.147 x 10`^{9}
Size in bytes	8	4

Integer overflow

Integer overflow occurs when the result of an integer expression exceeds the range of an integer number. Within the script language, when working with integer variables, we trap integer overflow for multiply and divide, but not for addition and subtraction. There is only one integer division operation that can overflow, which is division of the most negative possible integer (-9223372036854775808) by -1 which yields 9223372036854775808. This is greater than the maximum representable integer value.

Note that the integer array arithmetic functions (for example `ArrAdd()`) do detect overflow and handle it appropriately. Also, the compiler does detect overflow for add and subtract when processing expressions that can be evaluated at compile time, for example: `var i% := 0x7fffffffffffffff + 1;` is detected as an overflow.

Integer literals

When the script compiler finds a decimal number that contains no decimal point or exponent notation (`e` or `E`) it will interpret it as an integer if it is in the range `-9223372036854775808` to `9223372036854775807`. This can be written as -2^{63} to 2^{63}-1. If the magnitude of the number is in the range 2^{63} up to 2^{64}, it is interpreted as an integer with the binary bit pattern as set by the number, but the signed integer value will not be the same. For example, if you enter the value `9223372036854775809`, this will print as `-9223372036854775807`. You can avoid this by adding a decimal point after the value to force it to be recognised as a real number.

If you set an value with magnitude 2^{64} or larger, it will be interpreted as a real number.

String data type

Strings are lists of characters. String variable names end in a `$`. String variables can hold strings of length up to the limit described in Program size limits. Literal strings in the body of a program are enclosed in double quotation marks, for example:

```
"This is a string"
```

A string literal may not extend over more than one line. Consecutive strings with only white space between them are concatenated, so the following:

```
"This string starts on one lin"
"e and ends on another"
```

is interpreted as `"This string starts on one line and ends on another"`.

String encoding and sub-string access

The script language stores strings internally as a sequence of 16-bit numbers in UTF-16LE format. Most characters used for everyday work in European languages occupy 1 16-bit number each. However, some languages require two 16-bit numbers to specify characters (called surrogate pairs). Spike2 provides string functions, for example `Left$()`, `Right$()` and `Mid$()` to give you access to sub-strings. These define sub-strings by the 0-based index into the 16-bit numbers that represent the characters (the first character is at index 0), and by the count of 16-bit numbers. If a particular character requires 2 16-bit numbers any index that starts at the second number of a pair is moved on by 1. Any sub-string that ends on the first of a pair is also moved on by 1. That is, we try to ensure that we do not split a surrogate pair.

Special characters

String literals can hold special characters, introduced by the escape character backslash:

`\"` The double quote character (this would normally terminate the string)
`\\` The Backslash character itself (beware file paths, use: `"C:\\folder\\name.ext"` or `"C:/folder/name.ext"`)
`\t` The `Tab` character (ASCII code 9)
`\n` The `New Line` character (ASCII code 10)
`\r` The `Carriage Return` character (ASCII code 13)

A backslash followed by any other character is interpreted as the following character. For example, `"\x\y\z"` is the same as `"xyz"`. You should beware of this when using literal strings that hold file paths and when working with literal strings in regular expressions (see `InStrRE()`).

Unicode

From Spike2 version [8.03], there are additional escape sequences to allow you to embed Unicode characters into the string literal:

\uxxxx xxxx stands for up to 4 hexadecimal digits that specify a Unicode character in the range `0x0000` to `0xffff`. However, characters in the range `0xd800` to `0xdfff` are not allowed (these are reserved for UTF-16 lead and trail codes, used in *surrogate pairs*, see below). You can use this to input characters that are not easily available from your keyboard. For example, the Greek letter π is `\u03c0`. In reference material, Unicode characters are often written as U+03CO, and the hexadecimal digits after the U+ are the ones you need. You can omit leading 0s as long as the following character is not hexadecimal; you could write "π " as `"\u3c0 "` but "π2" would have to be `"\u03c02"`.
Beware: in many places when text strings are used, a character with code 0 (`\u0000`) will terminate the text string.

\Uxxxxxx Although most common characters in most languages can be represented by a 4 digit hexadecimal code, the full Unicode code range is from `0x000000` to `0x10ffff`. To represent the characters with codes `0x10000` to `0x10ffff` you can use `\U` followed by up to 6 hexadecimal characters. You can use fewer if the following character is not hexadecimal. For example the Unicode character for a music notation treble clef is code U+1D11E, so we could write this as `\U01d11e` or as `\U1d11e` (as long as the next character is not `0-9`, `a-f` or `A-F`).

In addition to using an escape sequence, you can type in all the characters that your keyboard supports and you can also use Input Method Editors.

There is no guarantee that all Unicode codes exist in the fonts on your computer. Characters that are missing from the font are usually rendered in a fall-back font (so may not match the style of the selected font), and failing that, as an empty square box. For example the treble clef character mentioned above is not present on my computer in a fall-back font.

Unicode surrogate characters

Spike2 strings in Unicode mode use the same underlying encoding as Windows does to store characters. Most characters fit in 16-bits of data, and the text strings are organised as arrays of 16-bit numbers. Characters with code points greater than 0xffff are represented by two consecutive characters known as surrogates. The string manipulation routines that use indices, such as `Left$()`, `Mid$()`, `Right$()`, `DelStr$()` and `InStr()`, use an index to these 16-bit elements. The routines that extract strings will never return a string that starts or ends half-way through a surrogate pair. They achieve this by moving on by one 16-bit element to the next. It would be possible for us to hide this from all users by making all indices into strings indices to characters, not to the underlying 16-bit codes. However, there would be a performance penalty for doing this and as surrogate pairs are usually very rarely encountered, it does not feel worth doing. We may review this in the future.

Text files

When we save text files on disk, we generally use UTF-8 format, which has the advantage of looking the same as ASCII when no characters with codes above 127 are used. The `FileOpen()` command, when used with external text files, expects you to use UTF-8 format, but can be forced to use UTF-16LE with the `mode%` argument. Whichever format you use, the `Print()` and `Read()` script commands will take care of any required translation between the external and internal formats.

Object data type

From Spike2 version 10 onwards the script language supports *user-defined objects*. These objects group data types, including other previously-defined objects together into a single item that can be passed into user-defined functions and procedures. You can declare arrays of objects and objects can contain arrays of data items and other objects.

The syntax to define an object type is:

```
object TName
    var variables;
    const constants;
    objType objects;

    func FName(...) end; 'optionally forward declare FName
    proc PName(...) end; 'optionally forward declare PName

    proc PName(args) 'Actual definition of procedure PName
    {code}
    end;

    func FName(args) 'Actual definition of function FName
    {code}
    end;
objend
```

The syntax for declaring variables, constants and other objects inside an object is identical to declaring them as globals or within user-defined procedures and functions, including the use of initialisers. The syntax for declaring user-defined functions and procedures within an object is also identical to declaring them at a global level. If you want to refer to a member `Func` or `Proc` from another member `Func` or `Proc` you must define if before you refer to it. Alternatively, you can forward declare it. A forward declaration uses `(...)` for the argument list as a signal to the compiler that this is a forward declaration. You only need forward declarations if functions are mutually recursive (otherwise you can re-order the functions to place the callee ahead of the caller).

We call items declared within an object, members of the object. Note that member functions and procedures cannot be passed as arguments as they need an object in addition to their name to define them.

Create object variables

The **object...objend** statement creates a new type. You can then create variables of this type or even arrays of variables of this type. There are two ways to make a variable of type **TName**:

```
TName obj1, obj2, aobj[10];
var x, i%, TName obj3, s$;
```

If you start a statement with the object type name, this declares a list of objects or arrays of objects of this type. You can also create an object of the type within a **var** statement, as in the second example.

Refer to members of object variables

Within an object member function or procedure, you refer to object members by their name. Anywhere else, you must provide a naming path that identifies which object variable and which member you are referring to. For example:

```
Object OTest
    var x,y,z;
    var a[10] := {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    proc Dump()
        PrintLog("x=%g, y=%g, z=%g, a=%g\n", x, y, z, a);
    end;
ObjEnd

OTest obj, ao[10];
obj.Dump();
var i%;
for i% := 0 to 9 do
    ao[i%].Dump();
    next;
```

Within the **Dump()** procedure, a member of **Object OTest**, we refer to the member variables by name. However, at the global level, we use the '**.**' symbol to indicate which member of which global object variable we want to access. You are allowed to declare objects and arrays of objects as members of an object as long as a member object is not of the same type, nor does it contain, the type of any enclosing object. That is, you cannot have a recursive object definition; an object cannot contain itself.

You can build a complex hierarchy:

```
Object O1
    var a,b,c;
ObjEnd;

Object O2
    O1 ao1[10][10];
    var a,b,c;
ObjEnd;

Object O3
    var x, O2 obj2;
ObjEnd;

var O3 obj3;

obj3.obj2.ao1[2][1].a := 3;
```

The self variable

Within a member function you can refer to the current object as **self**. The only circumstance in which you should use this is when you wish to pass the entire object to another **func** or **proc** or to refer to an object member that has the same name as a local variable. If the current object has a member called **a**, it is legal to refer to it as **self.a**, but this is less efficient (slower and uses one more instruction) that just using **a**.

Built-in object types

There are currently no built-in object types, and user-defined types cannot be passed to the built-in commands.

Conversion between data types

You can assign integer numbers to real variables and real numbers to integer variables (unless the real number is out of the integer range when a run-time error will occur). When a real number is converted to an integer, it is truncated. You can use the `Round()`, `Trunc()`, `Floor()` and `Ceil()` functions to convert floating point values to integral floating point values. The `Asc()`, `Chr$()`, `Str$()`, `Print$()` , `ReadStr()`, and `Val()` functions convert between strings and numbers.

Variable declarations

Variables store information in a script that can be accessed and modified. Variables must be declared before they can be used and the declaration determines the type of the variable (real, integer, string or user-defined object).There are three categories of variables:

Global — Variables declared outside any function or object. These can be accessed from anywhere in the script after they are defined. They can be hidden by local or member variables with the same name.

Local — Variables declared inside a `Func or Proc` can be accessed only from within the function and after the declaration. They 'hide' global variables with the same name. Within a member function they hide member variables with the same name. Each invocation of a function creates a fresh set of variables (this is important if the function is recursive).

Member — Variables declared inside a user-defined object are visible within member functions and procedures (as long as not hidden by a local variable with the same name) and from anywhere by specifying the object to which they belong.

Variables are created by the `var` keyword. This is followed by a list of variable names and optional initialising expressions and the list is terminated by a semicolon. You can declare variables of type real, integer, string or a previously-defined object type. Arrays are declared as variables with the size of each dimension in square brackets. The first item in an array is at index 0. If an array is declared as being of size `n`, the last element is indexed `n-1`.

```
var myInt%,myReal,myString$;     'an integer, a real and a string
var aInt%[20],arl[100],aStr$[3] 'integer, real and string vectors
var a2d[10][4];                  '10 rows of 4 columns of reals
var square$[3][3];               '3 rows of 3 columns of strings
var x, TObject y, z;             'a real, a user-defined object, a real
```

Currently, the script compiler allows variables (and constants) to be declared inside loops. For reasons of backwards compatibility, variables declared inside a loop are also visible after the loop, but it is possible that we may revisit this in the future, so we suggest that you only declare variables within a loop if you will only use them within the loop.

Before version 7.11, the dimensions of global arrays had to be constant expressions (but see `resize`). The dimensions of local arrays can be set by variables or calculated expressions. Simple variables (not arrays) can be initialised when they are declared. Uninitialised numeric variables are set to 0; uninitialised strings are empty.

```
var Sam%:=3+2, golden:=(1+sqrt(5))*0.5, sal$:="I am \"Sally\"";
```

You can initialize an array if you use a `const` declaration.

Compatibility with previous versions of Spike2

Before Spike2 version 7, the initialising expression could not include variables or function calls. If you want to write a script that is compatible with version 6, the previous example must be written as:

```
var Sam%:=3+2, golden, sal$:="I am \"Sally\"";
golden:=(1+sqrt(5))*0.5;
```

There was a script compiler bug in version 7.11 up to 7.16 and in version 8.00 up to 8.04a that meant that a variable that was declared inside a loop and assigned a constant value would not be reset to the constant value each time around the loop:

```
var i%;
for i% := 1 to 10 do
   var x% := 0;
   x% += 1;
next;
```

This was treated as though the declaration of `x%` was outside the loop, so after the loop, `x%` would be 10 before the bug was fixed and 1 after it was fixed.

var implementation

A `var` statement is implemented in two parts: the declaration part that creates the declared variables with all the properties that are known at compile time and the evaluation part that handles all the information that is not known at compile time and variable initialization. The declaration part is always done, even if the `var` statement is within a conditional statement that is not run. To check your understanding, try to predict what the output of the following program will be:

```
var sz% := 4;
if _Version > 2000 then       'Equivalent to if 0 then
    var fred[sz%], n% := sz%;
endif
Message("fred[%d], n%=%d", Len(fred), n%);
```

The result is `fred[0], n%=0` because both variables were created, but the `sz%` is a variable so is applied at run time, and the `if` statement condition is false (we have not yet reached Spike2 version 20), so the variable initialization was never done. If you change the declaration of `sz%` to:

```
const sz% := 4;
```

The result is `fred[4], n%=4` as all the information was available at compile time.

Note: The script compiler is not smart. It is obvious, to a human, that the variable `sz%` is never modified, and certainly not modified between having 4 assigned to it and being used. This could allow it to be treated as known at compile time, leading to more efficient code. However, this is not done (and if we did it, this would make an incompatibility with older versions of the compiler).

Constant declarations

Constants are created by the `const` keyword. A constant can be of any of the three basic data types, real, integer and string (but not a user-defined object type) and must be initialised in the declaration. From Spike2 version 8.03 onward, constants can be an array.

Before Spike2 version 10, constants had to be set to a *constant expression*. A constant expression is composed of previously defined constant expressions, numbers and the operators add, subtract, multiply and divide (`+-*/`) or is a string constant; its value must be known at script compile time. Now you can initialise a `const` item with values that are not known at compile time.

The syntax and use of constants is the same as for variables, except that you cannot change their value or pass them to a function or procedure as a non-**`const`** reference parameter. Constants are generally used to make the intention of the script writer clear; this name stands for a value that should never be changed. They are also used to remove 'magic numbers' from a script. For example the value 7 used in two different places could refer to two different things, but `const DaysInWeek% := 7, AgeInYears% := 7;` makes it clear what was intended, and makes it easy to keep the two values separate. Further, in a years time, if you want to change the age to 8, there is no question of which values to change.

Examples of declaring simple constants:

```
const Sam%:=3+2, golden:=1.6180339887498948, sal$:="I am \"Sally\"";
```

Constant arrays [8.03]

Constant arrays are useful only if the array is initialised:

```
const day$[] := {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
const prime%[10] := {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
```

The syntax for initialising a `const` array is the same as for a `var`. If you do not initialise the values, for example:

```
const day$[] := {};
const prime%[10] := {};
```

the `const` array has zero values or permanently empty strings. You cannot `resize` a `const` array. A `const` array may only be passed to a `func` or `proc` or built-in function that declares the argument as `const`.

Built-in constants [7.17, 8.03]

The following named constants are pre-defined. They are supported from Spike2 version 7.16 and 8.03 onwards onwards. You will get a compilation error if you attempt to redefine them for anything else:

Name	Type	Use
`_Version`	Integer	The program version number times 100 as a constant. So for version 11.00 this has the value 1100. This is the same value as returned by `App(-1)`, but it is known at compile time. This means that if a new built-in function **`MagicFunc()`** was added at version 11.01 you could write: `if _Version > 1100 then MagicFunc(23) else WorkAround(23) endif;` Where **`WorkAround()`** stands for code to emulate what the new function does. This works in version 8.03 onward and 9.xx (where the new function does not exist) because the compiler first builds an internal representation of the program in which **`MagicFunc()`** is assumed to be a user-defined function that it has not yet been found. Then the compiler notices that the **`then`** condition will never be run, so it converts the entire line to **`WorkAround(23)`**, and then it checks for undefined functions. This is no longer a problem as `MagicFunc()` is no longer used. In version 10.01 onwards the compiler will notice that the if condition is always true, and it will replace the entire line with **`MagicFunc(23)`**.
`_VerMinor`	Integer	Sometimes we release minor releases to fix a bug. These are give letter codes, for example 11.00a. For version 11.00 this is 0, for 11.00a it would be 1, for 11.00b it would be 2, and so on.
`_pi`	Real	The value of the mathematical constant π (**`3.141592653589`**...) as accurately as can be stored in a real number.
`_e`	Real	The value of the mathematical constant e (**`2.718281828459`**...) as accurately as can be stored in a real number.

Constant expressions

There are two types of items that are generated by a `const` declaration: items with a value known at compile time (*constant expressions*) and everything else. This difference matters when you are using the debugger. Non-array constant expressions, such as `const x := 3, a$:="text";` do not generate symbols that are visible to the debugger. The compiler uses the constant value when generating code, which is more efficient at run time than finding the symbol value. However, `var a := 3;const x := a;` does generate a symbol for **`x`** as it is assigned a value from a variable (and the compiler is not smart enough to figure out that this value must be 3).

Arrays of data

The three basic types (integer, real and string) can be made into arrays with from 1 to 5 dimensions. We call a one-dimensional array a *vector*, and a two-dimensional array a *matrix* to match common usage. Declare arrays with the `var` statement:

```
var v[20], M[10][1000], nd[2][3][4][5][6], text$[n%];
```

This declares a vector with 10 elements, a matrix with 10 rows and 1000 columns, a 5-dimensional array with 720 elements and a one-dimensional array of strings with a size set by `n%`. To reference array elements, enclose the element number in square brackets (the first element in each dimension is number 0):

```
v[6] := 1; x := M[8][997]; nd[1][0][0][0][2] := 4.5; text$[1]:="a";
```

From Spike2 [8.03] you can initialise arrays that are declared with a known size or omit the size and set it with the initialisation data:

```
var col$[] := {"Red", "Green", "Blue"};  'is the same as
var col$[3] := {"Red", "Green", "Blue"};
```

You can declare an array with one or more dimensions set to 0! However, such an array has limited utility; you can `resize` it, it can be passed to user-defined `proc` and `func` and some built-in script functions will accept 0-sized arrays (and do nothing) but in most cases, all dimensions must be non-zero to use the array.

From Spike2 [8.03] you can also declare arrays with a `const` declaration (you will almost certainly want to initialise them).

There is a maximum number of elements (product of the sizes of the dimensions) that you are allowed when you declare an array using `var`. This is currently set to 100,000,000 in an attempt to prevent operations that would likely take a very long time. If you need more elements than this, declare the array with a 0-sized dimension, then use `resize` (which we do not limit), or declare the size using a non-constant expression (which the compiler cannot check). However, be prepared for the script to fail if you run out of allocatable memory, which can happen, even if the size is less than the limit.

You will find that above some (large) size, arrays will become slow when accessed randomly. This happens when the operating system decides that your application is using too much memory and pages some out to disk, then loads it in as you use it. If you access such memory sequentially, the effect is not too disastrous as you will be loading new pages every few hundred accesses. However, if you access memory randomly, you will likely trigger a disk read every access. A disk read takes of order milliseconds compared to a fraction of a microsecond for a memory access.

You cannot have two global `var` statements that refer to the same variable. That is, you cannot have code like:

```
var fred[23][32];
...
var fred[23[48];           'This line will generate an error
```

as this will generate an error: `"Identifer is already defined at 'fred'"`. In a `Proc` or `Func`, you can declare an array inside a loop, and change the size of the dimensions each time around the loop. However, since version 7 we urge you to declare arrays outside loops and use `resize` to do any required size changes.

```
Proc BadStyle()
var i%;
for i% := 1 to 100 do
   var arr[i%];            'this has always been allowed...
   ...
next;
end;

Proc BetterStyle()
var arr[0], i%;            'declare the array once
for i% := 1 to 100 do
   resize arr[i%];         'resize it
   ...
next;
end;
```

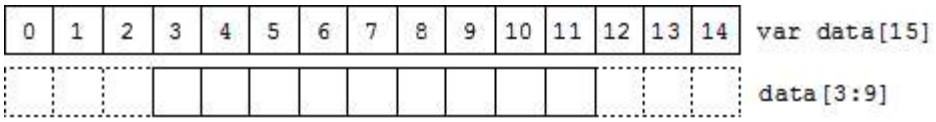
We may make resizing an array using **var** illegal in the future, so please do not do it in new code. Resizing with **var** preserved the original data when the last dimension is changed, but changes to any other dimension do not preserve the data.

Vector subsets

Use `v[start:size]` to pass a vector or a subset of a vector `v` to a function. `start` is the index of the first element to pass, `size` is the number of elements. Both `start` and `size` are optional. If you omit `start`, 0 is used. If you omit `size`, the sub-set extends to the end of the vector. To pass the entire vector use `v[0:]`, `v[:]`, `v[]` or just `v`.

For example, consider the vector of real numbers declared as `var data[15]`. This has 15 elements numbered 0 to 14. To pass it to a function as a vector, you could specify:

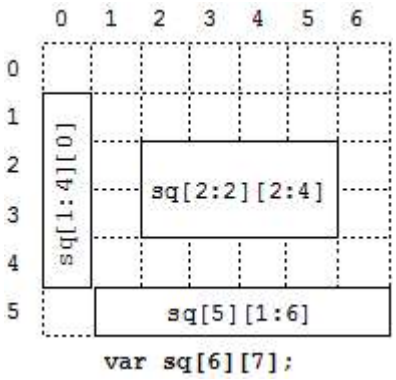
`data` or `data[]`	This is the entire vector. This is the same as `data[:]` or `data[0:15]`.
`data[3:9]`	This is a vector of length 9, being elements 3 to 11.
`data[:8]`	This is a vector of length 8, being elements 0 to 7.



Matrix subsets

With a matrix you have more options. You can pass a single element, a vector sub-set, or a matrix sub-set. Consider the matrix of real numbers defined as `var sq[6][7]`. You can pass this as a vector or a matrix to a function as (`a`, `b`, `c` and `d` are integer numbers):

`sq[a][b:c]`	a vector of length c
`sq[a][]`	a vector of length 7
`sq[a:b][c]`	a vector of length b
`sq[][c]`	a vector of length 6
`sq[a:b][c:d]`	a matrix of size `[b][d]`
`sq` or `sq[][]`	a matrix of size `[6][7]`



This diagram shows how sub-sets are constructed. `sq[1:4][0]` is a 4 element vector. This could be passed to a function that expects a vector as an argument. `sq[5][1:6]` is a vector with 6 elements. `sq[2:2][2:4]` is a matrix, of size `[2][4]`.

N-Dimensional array subsets

With more than 2 dimensions, you can make a subset of any number of dimensions up to the size of the original array. These examples show some of the possibilities for passing a source array with 5 dimensions defined as `var nd[4][5][6][7][8];`

`nd` or `nd[][][][][]`	The entire 5 dimensional array
`nd[0][][][][]`	A 4 dimensional array of size `[5][6][7][8]`
`nd[1:2][][2][][]`	A 4 dimensional array of size `[2][5][7][8]`
`nd[1][2][3][4][]`	A vector of size `[8]`
`nd[][][0][0][0]`	A matrix of size `[4][5]`

Transpose of an array

You can pass the transpose of a vector or matrix to a function with the `trans()` operator, or by adding ` (back-quote) after the vector or matrix name. The transpose of a matrix swaps the rows and columns. To be consistent with matrix mathematics, a one-dimensional array is treated as a column vector and is transposed into a matrix with 1 row. That is given `var data[15]`, `trans(data)` is a matrix of size `[1][15]`.

```
var M[5][3], v[5], W[5][5];
PrintLog(M, M`);               'Print M and its transpose
PrintLog(M[][], trans(M[][])); 'Exactly the same as last line
MatMul(W, M, M[][]`);          'set W to M times its transpose
MatMul(W, v, v`);              'set W to v times its transpose
```

From Spike2 version 6 onwards you can apply the transpose operator to arrays of higher dimensions. The result is an array with the dimensions and indexing reversed. That is, the transpose of `x[2][3][4]` is an array of size `[4][3][2]`. The element `x[i][j][k]` in the original becomes the element at index `[k][j][i]` in the transposed array.

Diagonal of a matrix or array

You can pass the diagonal of a matrix to a function using the `diag()` operator. This expects a matrix as an argument and produces a vector whose length is the smaller of the dimensions of the matrix. Given a matrix `M[10][10]`, `diag(M)` is a 10 element vector.

From version 6 of Spike2 you can take the diagonal of any array with more than 1 dimension. The result is a vector with the length of the smallest dimension of the array. For example, given **var a[4][5][6]**, **diag(a)** is a vector of length 4 holding the elements: **a[0][0][0]**, **a[1][1][1]**, **a[2][2][2]** and **a[3][3][3]**.

Memory usage of arrays

Each element of a real or integer array uses 8 bytes of memory. Each element of a string array uses 16 bytes, plus the memory to hold the text in the string. If you use many, large arrays, it is possible to run out of system memory. When this will happen is difficult to predict as it depend on many system settings, on how much physical memory your system has and how much virtual memory space the system will allocate to you. Once you exceed the limits of available physical RAM, random array access can become very slow as data is swapped between memory and disk. Adding more physical memory to your system can help if this is a problem. Spike2 scripts will stop with an error if you request more memory than the system can allocate.

Changes to arrays at Spike2 versions

Before Spike2 version 6, the maximum number of dimensions allowed was 2.

Prior to Spike2 version 7.11, dimension sizes for an array declared outside a `Proc` or `Func` (a global array) were all constants; from version 7.11 they can be defined with a variable expression. The following did not compile before version 7.11:

```
var n% := 10, x[n%];
```

You can now pass arrays of zero size to user-defined functions; previously this caused a fatal error. You can also pass zero-sized arrays to some built-in functions, for example `ArrConst()`.

Prior to version 8.03, arrays could not be declared **const** and could not be initialised.

From Spike2 version 10 you can have arrays of user-defined objects, and user-defined objects can contain arrays.

Initialise an Array

At the point where an array is declared it can also be initialised to values that are *constant expressions* (known at compile time). The following examples show how you initialise integer arrays, but the same syntax holds for real arrays and for string arrays (but replacing the numbers with literal strings, such as `"data"`). All the data used to initialise the arrays must be known at compile time. The syntax for **var** and **const** arrays is the same, so they are used interchangeably in the examples. Any data in the array that is not explicitly initialised is set to 0 (or an empty string).

If you need to set the values in an array to constant values (other than 0 or empty strings) and `ArrConst()` and its friends will not do the job, it is usually more convenient (less typing) and faster at run time to initialise them than to write out assignment statements.

Note that if you declare the array inside a loop, the array creation and any initialisation is done only once, not each time around the loop. However, if the array size is not a constant expression (known at compile time), the array is resized each time around the loop. Using `var` to resize an array in a loop is deprecated; you should use `resize` in new code.

Vectors

These are the simplest to initialise. The syntax for a vector is:

```
var name[optional size] := {list of constant expression items separated by commas};
```

If the size of the vector is specified, it must be a constant expression (known at compile time) and the list of items must be no longer that the size. If the size is not specified, the vector length is set by the initialisation list size. The items must be constant expressions. You cannot use this syntax to initialize any array element to a variable value.

```
var data%[] := {0, 1, 2, 3, 4};
const c%[10] := {,,,,1,1};
```

In the first example, no array size is specified, so the size is defined by the initialisation data. In the second example, the size is specified, but values are provided only for the fourth and fifth elements. Any unspecified elements are set to 0 (or to an empty string in the case of an array of strings), so the initialiser is equivalent to:

```
{0,0,0,0,1,1,0,0,0,0}
```

If the size of an array dimension is declared, it is an error to give more initialisation data than can fit:

```
var bad%[4] := {1, 2, 3, 4, 5};
const JustAsBad%[4] := {,,,,};
```

Matrix

These are a little more complicated: The syntax for a matrix is:

```
var name[rows][cols] := {{list of col 0 data}, {list of col 1 data}, ... ,{list of cols-1 data}};
```

Where ... means continue in a like manner. Both *rows* and *cols* are optional, but must be constant expressions if they are supplied. If either is missing, the size is set by the largest row or column of the initialisation data. Within the list of row data, you may omit values by having two consecutive commas, and the missing value will be set to 0. You can also set an entire row to 0 (or empty strings) by omitting it:

```
var name[rows][cols] := { , {list of col 1 data}, ... ,{list of cols-1 data}};
```

In this case, row 0 is initialised to zeros. The next example sets up two arrays to hold the same data.

```
var d2d%[2][3] := {{1, 2}, ,{5,6}};
var same%[][] := {{1,2}, {0,0}, {5,6}};
```

3 or more dimensions

The syntax continues in a logical sequence. So to initialise a 3D array:

```
var d3d%[2][3][4] := {{{1, 2},   {3, 4},   {5, 6}},
                      {{7, 8},   {9, 10},  {11, 12}},
                      {{13, 14}, {15, 16}, {17, 18}},
                      {{19, 20}, {21, 22}, {23, 24}}};
```

You can think of this as a 4 x 3 matrix of vectors of length 2. To handle a 4D one:

```
const d4d%[1][2][3][4] := {{{{ 1},{ 2}}, {{ 3},{ 4}}, {{ 5},{ 6}}},
                           {{{ 7},{ 8}}, {{ 9},{10}}, {{11},{12}}},
                           {{{13},{14}}, {{15},{16}}, {{17},{18}}},
                           {{{19},{20}}, {{21},{22}}, {{23},{24}}}};
```

You can think of this as a 4 x 3 matrix of 2 x 1 matrices. We leave the 5D array as an exercise. You can zero fill (or fill with empty strings) any region by omitting the contents of any pair of matching curly braces, you can even omit the curly braces (except the outermost pair). If we wanted to zero the elements with values 1-6 and 13-18 in the `d3d%` array we could have written:

```
var d3d%[2][3][4] := { ,
                      {{7, 8},   {9, 10},  {11, 12}},
                      {},
                      {{19, 20}, {21, 22}, {23, 24}}};
```

In the first case we have omitted everything, and in the second case we have omitted the contents of the curly braces.

Resize array

You cannot change the number of dimensions of an array, but you can change the size of the dimensions (unless the array is `const`).. This is done with the **`resize`** statement (added at version Spike2 version 7), which has a syntax that is very similar to `var`:

```
resize v[24], M[2][3000], nd[6][5][4][3][2], text$[923];
```

When used in this way, the values in the square brackets, which can be expressions or constants, set the new size for each dimension. However, if you want to leave a dimension at the current size, you can use:

```
resize nd[][][][][n%];  'change last dimension only
```

A pair of empty square brackets means that you want to preserve the current size of the corresponding dimension. The **`resize`** statement preserves data in the array (unless you make one or more dimensions smaller, when data is omitted). When you make dimensions larger, new numeric array elements are set to 0; new string elements are set to an empty string ("").

Unlike `var`, which checks the total size of the array (unless the arguments are expressions), there is no compiler-imposed limit on the total array size. However, each individual dimension is limited (currently to 100,000,000 elements). The script will stop with a fatal error if memory cannot be allocated.

In most cases, you will only want to change one dimension to cope with adding more items to an array. It is more efficient to increase the last dimension as in this case it is often possible to extend (or reduce) the memory allocated to the array without physically moving it in memory. If you change any dimension other than the last, the `resize` statement allocates a new array of the required size, copies data into it, replaces the original array with the new one and releases the memory used by the original array. If the new array dimensions are the same as the existing ones, nothing is done and no time is wasted.

Arrays that can be resized

You cannot resize a `const` array. You can always resize a local array (one declared inside the current `func` or `proc`). You can always resize a global array at the global level and even from inside a `proc` or `func` unless a sub-array, transpose or diagonal of it has been passed to a `Proc` of `Func` and is currently in use. You can resize an entire array passed as an argument. You can never resize an array that is a result view.

When you pass an entire array as an argument, the passed object is still the original array. However, if you pass a sub-array, this is a separate object with a link to the original (and uses the same storage space as the original). The original is flagged as having had a reference taken and then cannot be resized.

Some examples may make this clearer:

```
var global[2][3];
Level1(global);            'pass entire global array
resize global[3][3];       'OK

proc Level1(g[][])         'g is entire global array
var local[3][4];
TryResize(local);          'this is OK, passing entire array
TryResize(g);              'this is OK if g is entire array
TryResize(global);         'this is OK, passing entire array
TryResize(local[:2][]);    'will fail as is a sub-array
ObscureError(local[:2][]); 'pass sub-array of local, OK
ObscureError(g[:1][:1]);   'pass sub-array of global, not OK
end;

Proc TryResize(arr[][])
resize arr[][2];
end;

Proc ObscureError(h[][])   'will fail in the resize...
resize global[4][];        '...if h is a sub-array of global
end;
```

When you create a sub-array, transpose or diagonal of an existing array, a temporary array construct is created that depends on the original. If you were to resize the original, all the dependant arrays that referred to it would become invalid, so we do not allow you to make such a change. You cannot resize an array derived from a result view.

Efficiency

If you are adding items to an array, it is very inefficient to increase the array size for each item added. Apart from being very slow, this will cause a pattern of memory allocation that is about the worst possible for the performance of the system. The standard solution in this case is to start with a reasonable size, one that will be big enough for most situations, then when you need more, to allocate a sensible extra portion of space. If you have no idea how big the target is, the best algorithm (best in terms of reducing the number of reallocations and memory fragmentation) is to double the size each time you run out. However, this is also the most wasteful of memory. Increasing by a fixed amount or a fixed proportion of the existing size may work. Do NOT increase by one each time unless the array is very small and is never going to get very big.

Result views as arrays

The script language treats a result view as vectors of real numbers, one vector per channel. To access a vector element use **`View(v%,ch%).[index]`** where `v%` is the view, `ch%` is the channel and `index` is the bin number, starting from 0. You can pass a channel as an array to a function using **`View(v%, ch%).[]`**, or **`View(v%, ch%).[a:b]`** to pass a vector subset starting at element `a` of length `b`. You can omit `ch%`, in which case channel 1 is used. You can also omit **`View(v%,ch%)`**, in which case channel 1 in the current view is used.

If you change a visible result view, the modified area is marked as invalid and will update at the next opportunity.

Efficiency

Generating a reference to a result view as a script is much slower than referring to a script array. If you want to process individual array elements, it is much faster (and less typing) to pass the entire result view as an array to a **`Func`** or **`Proc`** and do the index manipulation there rather than doing it using the **`View(v%,ch%).[index]`** notation. Consider the following ways to set the bins of an 1000 bins result view to the values 0 to 999:

Method 1

```
var i%;
for i% := 0 to 999 do
    [i%] := i%;
next;
```

Method 2

```
ProcessResult([]);

Proc ProcessResult(arr[])
var i%;
for i% := 0 to 999 do
    arr[i%] := i%;
next;
end;
```

Method 3

```
ArrConst([], 1);
[0] := 0;
ArrIntgl([]);
```

Method 4

```
ProcessResult([]);

Proc ProcessResult(arr[])
var i%;
for i% := 0 to 999 do
    arr[i%] := i%;
next;
end;
```

Here are the times, measured in microseconds per array element on an Intel Core i7-4770K @ 3.50 GHz:

Method	µs	What is being done
1	0.992	For each reference, get the array, make an index, tell the view to invalidate the result
2	0.114	Only get the array once, tell the view to invalidate each bin
3	0.0045	Get the array twice, tell the view to invalidate a range twice + one bin invalidate
4	0.0026	Get the array once, tell the view to invalidate a range twice + one bin invalidate

You should avoid manipulating individual elements of a result view. If you must, pass the entire result view to a **`Func`** or **`Proc`** and do it there. Ideally use array arithmetic, and even then, consider passing the entire result view to a **`Func`** or **`Proc`** first.

Closing referenced result view

If you pass a result view as an array or a result view element as a reference argument to a user-defined `func` or `proc`, then close the result view, you will get a fatal script error:

```
A result view was closed that was aliased by a script array or variable
```

If the result view was closed by the user (for instance during `Interact()`) rather than by a script you will get the error when the script resumes. This error is fatal as there is no way for the script to continue if it holds references to something that no longer exists. The following minimal code demonstrates both ways of generating this error.

```
var rv% := SetResult(100, 1, 0, "test", "x");
Err0([]);       'Pass as an array
Err1([10]);     'Pass an element as a reference

proc Err0(data[])
FileClose();    'Error if data[] is the result array
end;

proc Err1(&v)   'v is passed by reference
FileClose();    'Error if v is part of the result array
end;
```

Memory usage of result views

Each channel uses 8 bytes per bin per channel. However, if you are averaging data and want to see the variance of each data point, this costs you an extra 8 bytes per bin, and if you want to keep individual bin counts, this costs a further 4 bytes per bin.

If you are saving event times into a channel for Raster display, the cost is around 100 bytes for each raster line and 8 bytes for every event.

If you average 20 channels of 50000 points per channel with all options on, this will use 20 MB of memory. This is not too bad in itself, but if you are generating a lot of result views and leaving them open, the system may start paging memory to disk, which will make things feel slow.

If you generate a lot of result views you can run out of other system resources, such as window handles.

Statement types

The script language is composed of statements. Statements are separated by a semicolon. Semicolons are not required before `else`, `endif`, `case`, `endcase`, `until`, `next`, `end` and `wend`, or after `end`, `endif`, `endcase`, `next` and `wend`. White space is allowed between items in statements, and statements can be spread over several lines. Statements may include comments. Statements are of one of the following types:

- A variable or constant declaration terminated by a semicolon
- An assignment statement of the form:

`variable := expression;`	Set the variable to the value of the expression
`variable += expression;`	Add the expression value to the variable
`variable -= expression;`	Subtract the expression value from the variable
`variable *= expression;`	Multiply the variable by the expression value
`variable /= expression;`	Divide the variable by the expression value

  The `+=`, `-=`, `*=` and `/=` assignments were added at version 3.02. `+=` can also be used with strings (`a$+=b$` is the same as `a$:=a$+b$`, but is more efficient).
- A flow of control statement, described below
- A procedure call or a function with the result ignored, for example `View(vh%);`

Comments in a script

A comment is introduced by a single quotation mark. All text after the quotation mark is ignored up to the end of the line.

```
View(vh%);  'This is a comment, and extends to the end of the line
'This is a whole line comment that includes Japanese (日本語) text
```

Spike2 does not have a block comment marker that allows comments to start on one line and end on another.

Hiding proc, punc, var and const definitions

You can hide `proc`, `func`, `var` and `const` definitions so that they do not generate tool tips by adding a comment that starts '! to the definition line. This can be useful in an include file where there are definitions that are intended to be used internally. If you do have such hidden declarations, it is usually a good idea to use some kind of naming convention so that there is no danger of re-using the name (as you do not get a pop-up hint that it already exists):

```
'MyInclude.s2s
const mi_HiddenNumber := 2.3; '! Secret number
var mi_InvisibleVar;          '! also hidden
func mi_PrivateAct()          '! for internal use
...
end;
```

Hiding items makes no difference to the script compiler. It only modifies the visibility of the item to the script language user interface.

Expressions and operators

Anywhere in the script where a numeric value can be used, so can a numeric expression. Anywhere a string constant can be used, so can a string expression. Expressions are formed from functions, variables, constants, brackets and operators.

Numeric operators

In numerical expressions, the following operators are allowed, listed in order of precedence:

	Operators	Names
Highest	`trans(), `, [], ()`	Matrix transpose (2 forms), array subscript, round brackets
	`-, not, ~` `  !`	Unary minus, logical not, bitwise not (version 9) You can use `!` for logical not from version 9
	`*, /, mod` `  %`	Multiply, divide and modulus (remainder) You can use `%` for modulus from version 9
	`+, -`	Add and subtract
	`<<, >>, >>>`	Shift left, shift right, bit shift right (all new at version 9)
	`<, <=, >, >=`	Less, less or equal, greater, greater or equal
	`=, <>`	Equal and not equal
	`and, band` `&&  &`	Logical AND, bitwise AND You can use `&&` and `&` from version 9
	`or, xor, bor, bxor` `||  |  ^`	Logical OR, exclusive OR and bitwise versions You can use `||`, `|` and `^` from version 9.
Lowest	`?:`	Ternary operator

The order of precedence determines the order in which operators are applied within an expression. Without rules on the order of precedence, `4+2*3` could be interpreted as 18 or 10 depending on whether the add or the

multiply was done first. Our rules say that multiply has a higher precedence, so the result is 10. If in doubt, use round brackets, as in `4+(2*3)` to make your meaning clear. Extra brackets do not slow down the script.

The divide operator returns an integer result if both the divisor and the dividend are integers. If either is a real value, the result is a real. So `1/3` evaluates to `0`, while `1.0/3`, `1/3.0` and `1.0/3.0` all evaluate to `0.333333...`

The minus sign occurs twice in the list because minus is used in two distinct ways: to form the difference of two values (as in `fred:=23-jim`) and to negate a single value (`fred :=-jim`). Operators that work with two values are called *binary*, operators that work with a single value are called *unary*. There are six unary operators, `` ` ``, `[]`, `()`, `-`, `not` and `~`, one ternary operator `?:`, the remainder are binary.

There is no explicit `TRUE` or `FALSE` keyword in the language. The value zero is treated as false, and any non-zero value is treated as true. The result of a logical operator has the value 1 for true. So `not 0` has the value `1`, and the `not` of any other value is `0`. If you use a real number in a logical test, be very cautious about testing for it being equal to another value. For example, the following loop may never end:

```
var add:=1.0;
repeat
   add := add - 1.0/3.0; ' beware, 1/3 would have the value 0!
until add = 0.0;         ' beware, add may never be exactly 0
```

Even changing the final test to `add<=0.0` leads to a loop that could cycle 3 or 4 times depending on the implementation of floating point numbers.

Backwards compatibility

If you write scripts that must run in versions of Spike2 before 9.00 do not use any of the following symbols as operators:

```
~ ! % << >> >>> && & || | ^
```

Unary operators

Unary operators take a singe argument and transform it to generate a result. The type of the result depends on the argument and the operator.

Op	Example	Use
`	M[][]`	Transpose a matrix or convert a vector to a matrix. The operator comes after the argument.
trans	trans(M)	This is an alternative to the ` operator.
[]	a[2]	Array subscript operator.
()	a*(b+c)	Round brackets, used to force order of evaluation and for argument lists.
-	-a	Negate the following numeric item. The result is of the same numeric type (real or integer). Negating the full-scale negative integer value (0x8000000000000000) generates an overflow error.
not !	not a !a	Logical not. Has the value 1 if `a` is 0, otherwise 0. `a` can be real or integer.
~	~1	Bitwise not. The argument must be an integer and the result is the bitwise inversion of the argument. The result of ~1 is -2, in hexadecimal `~0x0000000000000001` is `0xfffffffffffffffe`.

Binary operators

Binary operators take two arguments, one before the operator and one after. They can be sub-divided into arithmetic, bitwise, comparison and logical operators.

Arithmetic

The result type of an arithmetic binary operator is integer if both arguments are integers, otherwise it is real.

Op	Example	Use

*	a*b	Forms the product of the argument. Multiplying two integers will cause an overflow error is the result exceeds 64-bit integer range.
/	a/b	Divides the first argument by the second. Division by 0 is an error. It is a common error to forget that the result of dividing an integer by an integer is an integer, so `1/3` is 0. `1/3.0` (or `1.0/3` or `1.0/3.0`) is `0.333333...`
mod %	a mod b a%b	The remainder after dividing the first argument by the second, equivalent to `a-(a/b)*b`
+	a+b	Form the sum of the two argument. There will be an overflow error if both arguments are integers and the result exceeds integer range.
-	a-b	Form the difference of the two arguments. There will be an overflow error if both arguments are integers and the result exceeds integer range.
<<	a%<<b%	Bitwise left shift the first integer argument by the second integer argument. The shift must be positive and bits shifted in at the bottom are 0. For example, `1<<3` is 8. In binary terms: 0001 becomes 1000.
>>	a%>>b%	Signed bitwise right shift of the first integer argument by the second integer argument. The shift must be positive and the bits shifted in at the top match the sign of the number. For example `3>>2` is 0 and `-3>>2` is -1. `8>>3` is 1.
>>>	a%>>>b%	Unsigned bitwise right shift of the first integer argument by the second integer argument. The shift must be positive and the bits shifted in at the top are always 0. If the first argument is positive, the result is the same as for >>. However, `-1>>>1` is 9223372036854775807. In hexadecimal `0xffffffffffffffff>>>1` is `0x7fffffffffffffff`.

Bitwise

The bitwise operators all take integer arguments and generate an integer result.

Op	Example	Use
<<	a%<<b%	Bitwise left shift the first argument by the second argument. The shift (second argument) must be positive and bits shifted in at the bottom are 0. For example, `1<<3` is 8. In binary terms: 0001 becomes 1000.
>>	a%>>b%	Signed bitwise right shift of the first argument by the second argument. The shift must be positive and the bits shifted in at the top match the sign of the number. For example `3>>2` is 0 and `-3>>2` is -1. `8>>3` is 1.
>>>	a%>>>b%	Unsigned bitwise right shift of the first argument by the second argument. The shift must be positive and the bits shifted in at the top are always 0. If the first argument is positive, the result is the same as for >>. However, `-1>>1` is 9223372036854775807. In hexadecimal `0xffffffffffffffff >>> 1` is `0x7fffffffffffffff`.
band &	a% band b% a% & b%	Bitwise AND. If bit `n` of both the arguments is 1, the output bit `n` is 1, else it is 0. For example, `5 band 1` is 1 (in binary `0101 & 0001` is `0001`). The bitwise and is often used to clear out bits from a number. For example to clear bits 0-3 from `a%`: `a% := a% & ~15;`
bor |	a% bor b% a% | b%	Bitwise OR. If bit `n` of either argument is 1, the output bit `n` is 1, else it is 0. For example, `4 bor 1` is 5 (in binary `0100 & 0001` is `0101`). Bitwise OR is often used to set bits after they have been cleared by a bitwise AND: `a% := a% | 7;`
bxor ^	a% bxor b% a% ^ b%	Bitwise exclusive OR. If bit `n` of both arguments differ, the output bit `n` is 1, else it is 0. For example, `5 bor 1` is 4 (in binary `0101 & 0001` is `0100`). Bitwise exclusive OR is often used to toggle bits: `a% := a% ^ 1;` changes the state of bit 0.

Comparison

Comparison operators take two arguments that can be real or integer values. If both arguments are integers, the integer version of the operator is used, otherwise the real value version. The operators return the integer value 1 if the result of the comparison is true, and the integer value 0 if it is false. There are six comparison operators:

Op	Example	Use
<	a < b	Less than.

<=	a <= b	Less than or equal.
>=	a >= b	Greater than or equal.
>	a > b	Greater than.
=	a = b	Equal. Be very cautious when testing real values for equality, especially if you are testing the result of an expression.
<>	a <> b	Not equal.

Logical

The logical binary operators take two numeric arguments and combine them to generate an integer result of 1 if the logical operation is true and 0 if the logical operation is false. The two arguments are treated as false if they are equal to zero and true if they are not equal to zero. There are three logical operators:

Op	Example	Use
and &	a and b a && b	Logical AND. If both the arguments are non-zero, the result is 1, otherwise it is 0.
or ||	a or b a || b	OR operation. If either argument is non-zero, the result is 1, otherwise it is 0.
xor	a xor b	Exclusive OR operation. This is equivalent to (a <> 0) <> (b<>0). The result is 1 (true) if one of the arguments is true (not zero) and the other is false (zero).

String operators

In string expressions, the following operators are allowed, listed in order of precedence:

	Operators	Names
Highest	*	Multiply. "-" * 10 is equivalent to "----------". This was added at version 9.
	+	Concatenate
	<, <=, >, >=	Less, less or equal, greater, greater or equal
	=, <>	Equal and not equal
Lowest	?:	Ternary operator

The order of precedence determines the order in which operators are applied within an expression. Without rules on the order of precedence, "a"+"B"*3 could be interpreted as "aBaBaB" or "aBBB" depending on whether the add or the multiply was done first. Our rules say that multiply has a higher precedence, so the result is "aBBB". If in doubt, use round brackets, as in "a"+("B"*3) to make your meaning clear. Extra brackets do not slow down the script.

Multiplication

As applied to a string, the multiplication operator expects a string on the left and an integer on the right. If both the string and the integer are known at compile time, the string is generated as a literal, but this will generate an error if the result is more than 1000 characters. Otherwise, the string is processed at run time, but it is an error if the resulting string exceeds the maximum allowed string length.

Concatenation

This operator expects a string on the left and the right and generates a string that is the first string followed by the second. If both strings are known at compile time, the operation is done then to generate a literal. Otherwise, the strings are processed at run time; it is an error if the resulting string exceeds the maximum allowed string length.

Comparisons

The comparison operators can be applied to strings. Strings are compared character by character, from left to right. The comparison is case sensitive. To be equal, two strings must be identical. The character order for comparisons (lowest to highest) of the standard ASCII characters is:

```
space !"#$%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
abcdefghijklmnopqrstuvwxyz{|}~
```

If two strings match up to the length of the shorter string, the longer string is considered greater. Script strings are stored as Unicode characters in UTF_16LE format. This uses 16-bit values to store the characters with the majority of common characters fitting in one 16-bit value. However, the Unicode standard allows for over 1,000,000 characters, so some characters require two values to encode them. The encoding system is carefully arranged so that lexical sorting systems will still generate sensible results by comparing at the 16-bit level. Our comparisons also work at the 16-bit level and give sensible results for the comparison operators even for uncommon characters.

Do not confuse assignment `:=` with the equality comparison operator, `=`. They are entirely different. The result of an assignment does not have a value, so you cannot write statements like `a$:=b$:=c$;`.

Backwards compatibility

If you write scripts that must run in versions of Spike2 before 9.00, you must not use the string `*` operator.

The ternary operator

The ternary operator `?:` was added to the script language at Spike2 version 6 and has the following format:

`numeric expression ? expression1 : expression2`

The result of the ternary operator is *`expression1`* if *`numeric expression`* evaluates to a non-zero result otherwise it is *`expression2`*. You can use this anywhere that an expression would be acceptable, including when passing arrays as arguments to functions. However, *`expression1`* and *`expression2`* must be type compatible in the context of their use. For example, if one is a string, then the other must also be a string. If they are arrays passed as arguments, they must have the same type and the same number of dimensions. If they are arguments passed by reference, they must have identical type. In an expression, one can be integer and the other real, in which case the combined type is treated as real.

If either *`expression1`* or *`expression2`* evaluates to a **`const`** object, then the result will be treated as **`const`**. For example, the following gives a type mismatch error at the `?` because **`colours$`** is **`const`**, so the result of the ternary operator is also **`const`**:

```
const colours$[] := {"red", "green", "blue"};
var cols$[3] := {"cyan","magenta","yellow"};
Test(cols$[0] = "red" ? colours$ : cols$);

proc Test(s$[])
Message(s$);
end;
```

You can make it compile by changing the declaration of **`Test`** to **`proc Test(const a$[])`** so that a **`const`** array is acceptable.

You are not allowed to use this operator to choose between function or procedure names passed as arguments into functions or procedures.

Examples of expressions

The following (meaningless) code gives examples of expressions.

```
var jim,fred,sam,sue%,pip%,alf$,jane$;
jim := Sin(0.25) + Cos(0.25);
fred := 2 + 3 * 4;      'Result is 14.0 as * has higher precedence
fred := (2 + 3)* 4;     'Result is 20.0
fred += 1;              'Add 1 to fred
sue% := 49.734;         'Result is 49
sue% := -49.734;        'Result is -49
pip% := 1 + fred > 9;   'Result is 1 as 21.0 is greater than 9
jane$ := pip% > 0 ? "Jane" : "John"; 'Result is "Jane"
alf$ := "alf";
```

```
sam := jane$ > alf$;    'Result is 0.0 (a is greater than J)
sam := UCase$(jane$)>UCase$(alf$); 'Result is 1.0 (A < J)
sam := "same" > "sam";  'Result is 1.0
pip% := 23 mod 7;       'Result is 2
jim := 23 mod 6.5;      'Result is 3.5
jim := -32 mod 6;       'Result is -2.0
sue% := jim and not sam;'Result is 0 (jim = -2.0 and sam = 1.0)
pip% := 1 and 0 or 2>1; 'Result is 1
sue% := 9 band 8;       'Result is 8 (9=1001 in binary, 8=1000)
sue% := 9 bxor 8;       'Result is 1
sue% := 9 bor 8;        'Result is 9
```

The following expressions are valid in version 9 onwards, but not in previous versions of Spike2:

```
sue% := jim && not sam; 'Result is 0 (jim = -2.0 and sam = 1.0)
pip% := 1 && 0 || 2>1;  'Result is 1
jim% := 1 << 4;         'Result is 16
sue% := 9 & 8;          'Result is 8 (9=1001 in binary, 8=1000)
sue% := 9 ^ 8;          'Result is 1
alf$ := "alf"*2;        'Result is "alfalf"
```

Mathematical constants

Before Spike2 version 8.03 and 7.17 we didn't provide maths constants e and π as built-in constants **_e** and **_pi**. If you need to write a script that will also work with older version of Spike2 *e* is `Exp(1.0)` and π (pi) is `4.0*ATan(1.0)`. Alternatively, here are their values to as much accuracy as you can store in a double:

π 3.141592653589793
e 2.718281828459045

Flow of control statements

If scripts were simply a list of commands to be executed in order, their usefulness would be severely limited. The flow of control statements let scripts loop and branch. It is considered good practice to keep flow of control statements on separate lines, but the script syntax does not require this. There are two branching statements, `if...endif` and `docase...endcase`, and three looping statements, `repeat...until`, `while...wend` and `for...next`. You can also use user-defined functions and procedures with the `Func` and `Proc` statements.

The keywords for these statements are also used to control code folding and automatic formatting in the script editor.

if...endif

The `if` statement can be used in two ways. When used without an `else`, a single section of code can be executed conditionally. When used with an `else`, one of two sections of code is executed. If you need more than two alternative branches, the `docase` statement is usually more compact than nesting many `if` statements.

```
if expression then            'The simple form of an if
   zero or more statements;
endif;
if expression then            'Using an else
   zero or more statements;
else
   zero or more statements;
endif;
```

If *expression* is non-zero, the statements after the `then` are executed. If *expression* is zero, only the statements after the `else` are executed. The following code adds 1 or 2 to a number, depending on it being odd or even:

```
if num% mod 2 then
   num%:=num%+2;            'note that the semicolons before...
```

```
else                          '...the else and endif are optional.
   num%:=num%+1;
endif;

'The following are equivalent
if num% mod 2 then num%:=num%+2 else num%:=num%+1 endif;
num% += num% mod 2 ? 2 : 1;'An alternative using ?:
```

docase...endcase

These keywords enclose a list of `case` statements forming a multi-way branch. Each `case` is scanned until one is found with a non-zero expression, or the `else` is found. If the `else` is omitted, control passes to the statement after the `endcase` if no case expression is non-zero. Only the first non-zero `case` is executed (or the `else` if no case is non-zero).

```
docase
 case exp1 then
   statement list;
 case exp2 then
   statement list;
 ...
 else
   statement list;
endcase;
```

This example displays the type of a file handle:

```
'case.s2s
var i%,m$;
i% := ViewKind();      'get type of the current view
docase
   case i% = 0 then m$ := "time";
   case i% = 1 then m$ := "text";
   case i% = 2 then m$ := "output sequence";
   case i% = 3 then m$ := "script";
   case i% = 4 then m$ := "result";
   case i% = 8 then m$ := "external text";
   case i% = 9 then m$ := "external binary";
   case i% = 10 then m$ := "Spike2";
   case i% = 12 then m$ := "XY view";
   else m$ := "something else...";
endcase;
message("Current window is of type "+m$);
```

The following example sets a string value depending on the value of a number:

```
var base%:=8,msg$;
docase
 case base%=2 then  msg$ := "Binary";
 case base%=8 then  msg$ := "Octal";
 case base%=10 then msg$ := "Decimal";
 case base%=16 then msg$ := "Hexadecimal";
 else msg$ := "Pardon?";
endcase;
```

repeat...until

The statements between `repeat` and `until` are repeated until the expression after the `until` keyword evaluates to non-zero. The body of a repeat loop is always executed at least once. If you need the possibility of zero executions, use a `while` loop. The syntax of the statement is:

```
repeat
   zero or more statements;
until expression;
```

For example, the following code prints the times of all data items on channel 3 (plus an extra -1 at the end):

```
var time := -1;                'start time of search
repeat
   time := NextTime(3, time);  'find next item time
   PrintLog("%f\n", time);     'display the time to the log
until time<0;                  'until no data found
```

You can break out of a `repeat` loop early using the `break` statement. You can skip to the `until` statement with the `continue` statement.

```
var t;
repeat
   t := NextTime(3, t);        'get a time
   if (t<0) then break endif;  'stop if time not found
   if (t<10) then continue endif; ' skip if not enough time yet
   PrintLog("At time %5.2f 10 second count = %d\n", t, Count(3, t-10, t));
until t >= MaxTime(3);
```

while...wend

The statements between the keywords are repeated while an expression is not zero. If the expression is zero the first time, the statements in between are not executed. The `while` loop can be executed zero times, unlike the `repeat` loop, which is always executed at least once.

```
while expression do
   zero or more statements;
wend;
```

The following code fragment, finds the first power of two that is greater than or equal to some number:

```
var test%:=437, try%:=1;
while try%<test% do         'if try% is too small...
   try% := try% * 2;        '...double it
wend;
```

You can break out of a `while` loop early using the `break` statement. You can skip back to the start of the loop with the `continue` statement.

```
var i% := 0;
while i% <= 999 do
    ...
    if StopEarly() then
        break
    else
        if SkipRestOfLoop() then continue endif;
    endif;
    ...
wend;
```

for...next

A `for` loop executes a group of statements a number of times with a variable changed by a fixed amount on each iteration. The loop can be executed zero times. The syntax is:

```
for v := exp1 to exp2 {step exp3} do
   zero or more statements;
next;
```

`v` This is the loop variable and may be real or integer. It must be a simple variable, not an array element.

`exp1` This expression sets the initial variable value just before the looping begins. Both `exp2` and `exp3` are calculated before the assignment.

`exp2` This expression is evaluated once, before the loop starts and before the assignment to `v`, and is used to test for the end of the loop. If `step` is positive or omitted (when it is 1), the loop stops when the variable is greater than `exp2`. If `step` is negative, the loop stops when the variable is less than `exp2`.

`exp3` This expression is evaluated once, before the loop starts, and sets the increment added to the variable when the `next` statement is reached. If there is no `step exp3` in the code, the increment is 1. The value of `exp3` can be positive or negative.

The following example prints the squares of all the integers between 10 and 1:

```
var num%;
for num% := 10 to 1 step -1 do
   PrintLog("%d squared is %d\n", num%, num% * num%);
next;
```

This is a more complicated example that prints prime numbers:

```
const MaxN% := 1000;      'the maximum number to search
var prime%[MaxN%],i%,t%;  'the "Sieve" and loop variables
for i%:=1 to MaxN%-1 do   'ignore 0 as not included in search
   prime%[i%] := 1        'mark all numbers as possible primes
next;
for t%:=2 to MaxN%-1 do   'start search at 2 as 1 is special case
   if prime%[t%] then     'if number remains, must be prime
      for i% := 2*t% to MaxN%-1 step t% do
         prime%[i%]:=0    'remove all multiples of prime number
      next;
   endif;
next;
t% := 0;                  'use this to put output in columns
for i% := 1 to MaxN%-1 do 'search the "sieve" for next prime
   if prime%[i%] then     'have we found one?
      if t% mod 10 = 0 then PrintLog("\n") endif;
      t% := t% + 1;       'count the number
      PrintLog("%5d", i%);'output the prime
   endif;
next;
```

If you want a `for` loop where the end value and/or the `step` size are evaluated each time round the loop you should use a `while…wend` or `repeat…until` construction. You can break out of a `for` loop early using the `break` statement. You can skip to the `next` statement with the `continue` statement.

break, continue

Within the three looping statements (`for...next`, `repeat...until` and `while...do`) you can use the `break` and `continue` statements. The `break` statement jumps out of the enclosing looping statement; the `continue` statement jumps to the evaluation of the expression that determines if a `repeat` or `while` will run again and to the `next` in a for loop. For example:

```
var i%;
for i% := 0 to 1000 do
   if StopEarly() then break endif;
   if SkipToNext() then continue endif;
   DoSomething(i%);
next;                'continue jumps here

DoSomethingElse();   'break jumps to this statement
```

It is an error to use either of these statements outside a loop. If you need to jump out of more than one level of looping statements, put the code in a `Proc` or `Func` and use the `return` statement to jump out.

The `break` and `continue` statements are useful because they give you a way to jump out of deeply nested `if` or `docase` statements inside a loop in a similar way that the return statement allows you to jump out of the middle of a `Proc` or `Func`.

The `break` and `continue` statements were added at Spike2 version 7; if you use them your script will not compile in older versions of Spike2.

halt

The `Halt` keyword ends a script. A script also terminates if the path of execution reaches the end of the script. When a script halts, any open external files associated with the `Read()` or `Print()` functions are closed and any windows with invalid regions are updated. Control then returns to the user unless `ScriptRun()` has been used to set the next script to run. The `Halt` statement can occur at the outer level of a script or in a function or procedure. For example:

```
Proc DoSomething(arg)
if arg < 0 then
   Message("Bad argument to DoSomething(%g)", arg);
   halt;
endif;
   ...
end;
```

User-defined functions

A user-defined function is a named block of code. It can access global variables and constants, the function arguments and create its own local variables and constants. Information is passed into user-defined functions through arguments. Information can be returned by giving the function a value, by altering the values of arguments passed by reference or by changing global variables.

User-defined functions that return a value are introduced by the **func** keyword, those that do not are introduced by the **proc** (procedure) keyword. Within this documentation we refer to both **func** and **proc** generically as functions.

The **end** keyword marks the end of a function. The **return** keyword returns from a function. If **return** is omitted, the function returns to the caller when it reaches the **end** statement. Arguments can be passed to functions by enclosing them in brackets after the function. Functions that return a value or a string have names that specify the type of the returned value unless the function returns a user-defined **object** type, see below. A function is defined as:

```
func {OType} fname({argument list}) or proc pname({argument list})
{var local-variable-list;}             {var local-variable-list;}
{const local-constant-list;}           {const local-constant-list;}
{statements including return x;}       {statements including return;}
end                                    end
```

The statements within the function can occur in any order as long as local variable and constants are declared before they are used. The **end** keyword terminates the statement. Functions may not be nested within each other. You use a function by 'calling' it and passing any required arguments. The arguments you pass must match the argument list (though we do allow real and integer values to be mixed where they are compatible).

```
func ReturnsString$(n%, greeting$)
return print$("User %d says %s", n%, greeting$);
end

var a$;
a$ := ReturnsString$(1, "hi");
```

User-defined object member functions

If you define a **func** or **proc** within an **Object...Objend** block, this creates a *member* function. In addition to referencing global and local variables and constants, a member function can also access the members of the object. The same name can be defined as a global, as a local and as a member, so there is a defined search order to resolve the meaning of an unqualified name:

1. in the local symbols
2. in the object members
3. in the global symbols

The first name found wins. Here is an example of an object with member functions.

```
Object OExample
    var x;
    proc SetX(newX) x := newX end
    proc Test() SetX(1) end  'refer to SetX from within a member
    proc Silly(x) self.x := x end; 'Legal, but less efficient than SetX().
ObjEnd

OExample obj;    'Make an instance of the OExample object
obj.SetX(2);     'refer to SetX from outside the object
obj.Test();
PrintLog("Object x value = %f\n", obj.x);
```

This example declares an object type; think of this as a template for making objects of this type. The object has a member variable called **x** and member procedures, **SetX()**, **Test()** and **Silly()**. **SetX()** can refer directly to the member variable **x**, and **Test()** can refer directly to **SetX()** because both procedures are members of the same object. **Silly()** is equivalent to **SetX()** and illustrates a use of **self**.

However, at the global level, the names **SetX**, **Test**, **Silly** and **x** have no meaning as there are no global items with these names. However, there is a global item **obj** with members **Test**, **SetX**, **Silly** and **x**, so **obj.SetX()**, **obj.Test()**, **obj.Silly()** and **obj.x** all can be referenced.

Member functions 'know' which instance of the object they are attached to as the script compiler passes a reference to the object as the first (invisible) function argument with the name **self**. Inside **SetX()** you can refer to **x** as **self.x**, if you wish, but it is less efficient (slower) and takes more typing than using **x**. You cannot declare any argument or local variable with the name **self** as it already exists. The **self** item allows you to pass the entire object to another function, or to refer to a member variable rather than a local variable, as in the **Silly()** function in the example.

return

The `return` keyword is used in a user-defined function to return control to the calling context. In a `proc`, the `return` must not be followed by a value. In a `func`, the return should be followed by a value of the correct type to match the function name. If no return value is specified, a `func` that returns a real or integer value returns 0, a `func` that returns a string value returns a string of zero length and a `func` that returns a user-defined object returns an newly initialised object.

You can use a `return` statement in a function or procedure to break out of a loop or an `if` or `case` statement. If you are using a return to break out of a loop early and do not need to be compatible with version 6, you could consider using the `break` statement.

Note that although `return` can only return a single simple value (not an array), you can return other values from a function call by using reference arguments or array arguments (which are always passed by reference).

If a function reaches the `end` statement without hitting a `return`, this is equivalent to hitting a `return` with no value.

Functions returning a user-defined object

A `func` can return a copy of a user-defined object that is visible within the function. This is not always the most efficient thing to do as the object and all its members must be copied. You may get better results by passing an object as a reference argument. The name of a function that returns a user-defined object cannot end in `$` or `%` as this contradicts the user-defined type of the function. The syntax is:

```
func OType name({argument list})
OType obj;
{statements that set obj to the desired state};
return obj;
end
```

In this example, **OType** is a previously-defined object type.

Recursive functions

A recursive function is one that calls itself, either directly or via some intermediary function. The canonical example of this is the integer factorial. However, if you really need a factorial, see `BinomialC()` or `LnGamma()` first:

```
Func Factorial(n%)
return (n% <= 1) ? 1 : n% * Factorial(n%-1) endif;
end
```

Each call to a function is allocated a new set of local variables; recursive function calls do not share local variables.

Forward references to functions

You are allowed to call functions before they are defined. This is called a *forward reference* and is required if two functions are mutually recursive (call each other) or you may prefer to declare functions at the end of your script. There is no problem with this when the function name ends with a $ or a % as the function return type is set, which is all the script compiler needs to know. However, if you refer to a function with a name that does not end in $ or %, this could be a `proc` or a `func` returning a real value or a `func` returning a user-defined object type.

The script compiler is smart enough to figure this all out once it finds the function definition, but you can forward declare a `func` or `proc` to make your intentions clear. The syntax for this is:

```
func {OType} fname(...) end
proc pname(...) end
```

The `(...)` stands for a list of zero or more arguments, which are not supplied. There can only be comments and white space between the `(...)` and the `end` keyword. The actual arguments are discovered when the function is defined later in the script.

Name resolution

The same name can be declared as a global, as an object member and as a local variable in a function. Unless you qualify the name in some way, the name selected is the *closest binding* name. Locally declared items or function arguments are the closest binding, followed by member names in a member function and global names are looked at last of all. Built-in command names have the lowest priority.

```
var x$ := "global";

Proc local()
    var x$ := "local";
    Message("%s\n", x$);
end;

object TObj
    var x$ := "member";

    Proc Mbr()
        var x$ := "member local";
        Message("x$ := %s\n", x$);
        Message("self.x$ := %s\n", self.x$);
    end

ObjEnd

Message("x$ := %s\n", x$);
local();
TObj obj;
obj.Mbr();
```

Example function

This example function finds the greatest common divisor of two integers:

```
var n1%, n2%;
n1% := Input("First number", 10000, 1, 2000000000);
n2% := Input("Second number", 30000, 1, 2000000000);
Message("Greatest common divisor of %d and %d is %d", n1%, n2%, gcd%(n1%, n2%));
halt;

'Greatest common divisor
'm%  The first number to obtain the greatest common divisor. This
'    should be greater than 0.
```

```
'n%  The second number. This must also be greater than 0.
'Return  The greatest common divisor of m% and n%
func gcd%(m%, n%)
var d%;
while n% do
   d% := Max(m%, n%) mod Min(m%, n%);
   m% := Min(m%, n%);  'get smaller value
   n% := d%
wend;
return m%;
end;
```

Call tips

You can define pop-up call tips that appear when you hover the mouse over a user-defined `Proc` or `Func` name. These are enabled in the script file settings dialog.

The **gcd%** *Example function*, above, illustrates how you could format the function information. The basic idea is that contiguous comments before the `func` or `proc` definition are used to generate the call tip. If you hover the mouse pointer over the name **gcd%**, a formatted version of the comments appears with a copy of the function declaration line at the end. The example above generates:

```
Greatest common divisor
m%      The first number to obtain the greatest common divisor. This
        should be greater than 0.
n%      The second number. This must also be greater than 0.
Return  The greatest common divisor of m% and n%
```

If a line starts with two or more spaces or a tab character, or starts with a single word and then two or more spaces or a tab, the following text is aligned at a tab. This very simple system allows you to generate tidy help call tips, should you wish. If you do not provide any comments, the call tip is the text on the line where the function is defined. If there are no comments on the line before the definition, the parser searches the lines directly after the definition.

Hiding Func and Proc

You can 'hide' the function from the call tip system by adding a comment on the same line as the function that starts '!, so:

```
Func Hidden() '! Do not help user with this function
...
end;
```

Will not appear in the Functions list and will not display a call-tip.

Code folding

The `proc`, `func` and `end` keywords are used to set fold points for code folding and automatic formatting.

Argument lists

The argument list is a list of variable names separated by commas. Older versions of Spike2 limited the number of arguments to 20. The type of each item passed is either implicit in the name (if it ends in $ or %) or is a real unless the name is preceded by a user-defined object type:

```
Object TObject
var x,y,z;
Objend

Proc example(a, b%, c$, TObject d)
...
end;
```

In this example, **a** is a real value, **b%** is an integer, **c$** is a string and **d** is the user-defined object type **TObject**.

Pass by Value or by Reference

There are two ways to pass arguments to a function: by value and by reference:

Value — Arguments passed by value are local variables in the function. Their initial values are passed from the calling context, so this is equivalent to making a copy. Changes made in the function to a variable passed by value do not affect the calling context. Passing by value is the standard method for everything except arrays.

Reference — Arguments passed by reference are the same variables (by a different name) as the variables passed from the calling context. For reasons of efficiency, arrays are always passed by reference. Non-array variables have `&` before the argument name declaration to pass by reference. Changes made to arguments passed by reference do affect the calling context. Because of this, reference arguments must be passed as variables (not expressions or constants) and the variable must match the type of the argument. For convenience, we do allow you to pass a literal of matching type to a **`const`** reference argument. To pass an object by reference place the `&` between the object type and the object name: `TObject& d`, for example.

By default, simple (non-array) variables are passed by value. Simple variables can be passed by reference by placing the `&` character before the name in the argument list declaration.

```
proc BothWays(a, &b)
a := 6;     'a is local to BothWays
b := 6;     'This changes the caller value
end;

var x,y;
BothWays(x, y);   'x is unchanged, y is set to 6
```

Arrays and sub-arrays are always passed by reference. Array arguments have empty square brackets in the function declaration, for example `one[]` for a vector and `two[][]` for a matrix. The number of dimensions of the passed array must match the declaration. The array passed in sets the size of each dimension. You can find the size of an array with the `Len()` function. You can declare an array argument as **`const`** and the compiler will generate an error if you attempt to modify the array or pass it as a non-**`const`** argument to another function. An individual array element is a treated as a simple variable.

You can even change the size of an array that was passed as an argument as long as you meet the rules for resizing, and the new size will be seen by the caller of your function.

If you use the `trans()` or `diag()` operators to pass the transpose or diagonal of an array to a function, or pass a sub-array, the array is still passed by reference and changes made in the function to array elements will change the corresponding elements in the original data. You cannot resize such an array. If the original array was declared **`const`**, the result of these operators is also **`const`** and you can only pass such objects into functions with a **`const`** argument..

const arguments

You can place the keyword **`const`** before any argument (including a reference argument). If you do this, it is an error to modify the variable within the **`func`** or **`proc`**. Declaring an argument **`const`** makes it clear to the caller that the function will not modify the argument.

Default arguments

Simple (non array, non user-defined object) arguments that are passed by value can also specify a default value. This is done by following the variable name by `:= value`, for example:

```
func fred(a := 1, b$ := "", c% := 0)
...
return a;
end;
```

When arguments are defined with default value you can omit them entirely when they are trailing arguments, or skip over them if they are not. For example, all the following are legitimate:

```
'call as         equivalent to
fred();          'fred(1,"",0)
fred(2);         'fred(2,"",0)
fred(,"x");      'fred(1,"x",0)
fred(,,6);       'fred(1,"",6)
```

Default arguments are just a convenience. The code is generated exactly as if you had supplied the default argument(s). A common use is to extend an established command to cover a special case. By adding an addition

argument with a default value you can leave all your existing code unchanged and then use the extra argument only when required.

References and deprecated behavior [8.05]
Before Spike2 version 8.05 we allowed you to pass a real variable where a reference to an integer variable was expected. This was to remain compatible with ancient versions of the compiler. If your script uses this, you can make it compile by checking the *Allow float script var to be passed as an integer reference matching 8.04* option in the Compatibility section of the Edit menu Preferences.

Examples of user-defined functions

```
proc PrintInfo()      'no return value, no arguments
PrintLog(ChanTitle$(1));   'Show the channel title
PrintLog(ChanComment$(1));   'and the comment
return;         'return is optional in this case as...
end;            '...end forces a return for a proc
```

```
func sumsq(a, b)      'sum the square of the arguments
return a*a + b*b;
end;
```

```
func removeExt$(name$)   'remove text after last . in a string
var n% := InStrRE(name$, "\.[^.]*$"); 'See if we match
if n% = 0 then return name$ endif;    'no extension
return Left$(name$, n%-1);            'string without the extension
end;
```

```
proc sumdiff(&arg1, &arg2)   'returns sum and difference of args
arg1 := arg1 + arg2;   'sum of arguments
arg2 := arg1 - 2*arg2;   'original arg1-arg2
return;         'results returned via arguments
end;
```

```
func sumArr(const data[])   'sum all elements of a vector
var sum:=0.0;         'initialise the total
var i%;         'index
for i%:=0 to Len(data[])-1 do
   sum := sum + data[i%];   'of course, ArrSum() is much faster!
   next;
return sum;
end;
```

```
Func SumArr2(const data[][]) 'Sum of all matrix elements
var rows%,cols%,i%,sum;    'sizes, index and sum, all set to 0
rows% := Len(data[0][])    'get sizes of dimensions...
cols% := Len(data[][0]);   '...so we can see which is bigger
if rows%>cols% then        'choose more efficient method
 for i%:=0 to cols%-1 do sum += ArrSum(data[i%][]) next;
else
 for i%:=0 to rows%-1 do sum += ArrSum(data[][i%]) next;
endif;
return sum;
end;
```

Variables declared within a function exist only within the body of the function. They cannot be used from elsewhere. You can use the same name for variables in different functions. Each variable is separate. In addition, if you call a function recursively (that is it calls itself), each time you enter the function, you have a fresh set of variables.

Scope of user-defined functions

Unlike global variables, which are only visible from the point in the script in which they are declared onwards, and local variables, which are visible within a user-defined function only, user-defined functions are visible from all points in the script.

Mutually recursive functions

Because functions are globally visible, you may define two functions that call each other (mutually recursive). If you do this it is your responsibility to ensure that there is a way out of the mutual recursion. If you do not, the script will compile, but will crash at run time as it will (eventually) exhaust system memory.

Functions as arguments

The script language allows a function or procedure to be passed as an argument. The function declaration includes the declaration of the function type to be passed. Functions and procedures can occur before or after the line in which they are used as an argument. You cannot pass a member function or procedure as an argument.

```
proc Sam(a,b$,c%)
...
end;
```

```
func Calc(va)
return 3*va*va-2.0*va;
end;
```

```
func PassFunc(x, func ff(realarg))
return ff(x);
end;
```

```
func PassProc(proc jim(realarg, strArg$:="", son%:=0))
jim(1.0,"hello",3);
jim(2.0);                    'using default arguments
end;
```

```
var value := PassFunc(1.0, Calc);  'pass function
PassProc( Sam );               'pass procedure
```

The declaration of the procedure or function argument is exactly the same as for declaring a user-defined function or procedure. This means you can also define default arguments. When passing the function or procedure as an argument, just give the name of the function or procedure; no brackets or arguments are required. The compiler checks that the argument types of a function passed as an argument match those declared in the function header. See the `ToolbarSet()` function for an example.

Although user-defined functions and built-in functions appear similar, they are not the same and you are not allowed to pass a built-in function as an argument to a built-in function. Further, you cannot pass a built-in function to a user-defined function if it has one or more arguments that can be of more than one type. For example, the built-in `Sin()` function can accept a real argument, or a real array argument, and so cannot be passed. If you need to pass such a function you must wrap it in a user-defined function:

```
func USin(x) return Sin(x) end; 'USin(x) has an unambiguous signature
```

Channel specifiers

Several built-in script commands use a channel specifier to define a list of 1 or more channels. This argument is always called `cSpc` in the documentation. This argument stands for a string, an integer array or an integer.

`cSpc$` This can either hold a list of channel numbers and ranges, separated by commas or can specify a search of channel titles, units and comments to select channels that match. Use `ChanList()` to convert a `cSpc$` to a list of channel numbers.

Channel numbers and ranges: A channel range is a start channel number followed by an end channel number separated by two dots. The end channel number can be less that the start channel number. For example `"1,3,5..7,12..9"` is a list of channels 1, 3, 5, 6, 7, 9, 10, 11 and 12. If a `cSpc$` contains an error in a single channel specifier the error is noted and reported and parsing continues. An error in a channel range terminates parsing of the string, but any channels found before the error are preserved.

Channel search: A channel search is a string of the form `"what=regexpr"`, where `what` specifies what we search and `regexpr` is an ECMAScript regular expression that specifies the

text that must be matched to include this channel. The text of `what` is scanned and each T, U and V (case is important) in `what` is replaced by the Title, Units and Comment of a channel. Then the resulting string is searched by the regular expression. If what is empty, it is treated as T (title search). For example, `"T|U="y|V"` would match channels with a title that ended with y and units that start with V. `"=G\\d+ x"` would match channel titles like `"G2 x"` or `"G29 x"`.

This format can also be used in dialogs that accept a channel specification.

`cSpc%[]` The first element of the vector is the number of channels. The remaining elements are channel numbers. It is an error for the vector passed in to be shorter than the number of channels + 1. This can be a `const` array.

`cSpc%` This is either a channel number, or –1 for all channels, -2 for visible channels, –3 for selected channels. From Spike2 version `[10.05]` you can also use -6 for unselected channels. From version [10.09] you can use -4 for `No channels`, where this is allowed. Value -5 is reserved.

In a result view, channel numbers start at 1, but we accept 0 (meaning 1) to avoid breaking scripts written for older versions of Spike2. Some commands expect channels of specific types; channels that do not meet the type requirements are removed from the list. It is usually an error for a channel specification to generate an empty list.

Include files

There are times when you will want to reuse definitions or user-defined procedures and functions in multiple projects. You can do this by pasting the text into your script, but it can be more convenient to use the `#include` command to include script files into a script. A file that is included can also include further files. We call these *nested* include files. Only the first `#include` of a file has any effect. Subsequent `#include` commands that refer to the same file are ignored. This prevents problems with script files that include each other and stops multiple definitions when two files include a common file. The `#include` command must be the first non-white space item on a line. There are two forms of the command:

```
#include "filename" 'optional comment
#include <filename> 'optional comment
```

where `filename` is either an absolute path name (starting with a `\` or `/` or containing a `:`), for example `C:\Scripts\MyInclude.s2s`, or is a relative path name, for example `include.s2s`. The difference between the two command forms lies in how relative path names are treated. The search order for the first form starts at item 1 in the following list. The search for the second form starts at item 3.

1. The folder where the file with the `#include` command lives. If this fails...
2. The folder of the file that included that file up to the top of the list of nested include files. If this fails...
3. Any `\include` folder in the `Spike11` folder inside your `My Documents` folder . If this fails...
4. Any `\include` folder in `\Users\Public\Public Documents\Spike11Shared`. If this fails...
5. Any `\include` folder in the folder in which Spike2 is installed. If this fails...
6. Search the current folder.

Included files are always read from disk, even if they are already open. If you have set the Edit menu Preferences option to save modified scripts before running, modified include files are automatically saved when you compile. If this option is not set, the script compiler will stop the compilation with an error if it finds a modified include file. You must save the included file to compile your script.

There are no restrictions on what can be in an included file. However, they will normally contain constant and variable definitions and user-defined procedures and functions. It is usually a good idea to have all your `#include` commands at the start of a script so that anyone reading the source is aware of the included files.

The `#include` command was added to Spike2 at version 6.03 and is not recognised by any version before this. A typical file using `#include` might start with:

```
'$Example|Example of use of include files
#include <sysinc.s2s>          'my system specific includes
#include "include\proginc.s2s" 'search script relative folder
var myvar;                     'start of my code...
```

Opening included files

If you right-click on a line that holds an include command, and Spike2 can locate the included file, the context menu will hold an entry to open the file. The search for the file follows that described above, except that it omits step 2.

You can also open an include file by right-clicking on the name of any user-defined `Proc` or `Func` that is defined in an included file and selecting the Go to command. This will open the include file with the text caret at the start of the `Proc` or `Func`.

Include files and debugging

You can debug a script that uses included files in exactly the same way as one that does not. If you step into a user-defined function or procedure that is in an included file, the included file will open and the stepping marker will show where you have reached. If you want to set a break point in an included file before you run the script, open the included file, set the break point and then run the script (leaving the included file open).

Included files that are open within Spike2 are hidden from the script `ViewList()` command in the same way that the running script is not visible. Prior to Spike2 9.04, unlike the running script, whose view handle can be obtained with the `App()` command, there is currently no mechanism built into the system to let you find the

handles of included files. From version 9.04 onwards, there is a new flag for the `ViewList()` command to include the running script and any included files.

If you want Auto complete to work with included files you must check the Included files check box in the Edit menu Auto Complete... command.

Program size limits

The use of `#include` makes it much easier to construct very large scripts. However, there are some limits on the size of a script that you should bear in mind before trying to amalgamate every script you have ever written into one "superscript":

- The maximum instructions in a compiled script is currently set at 1000000. The number of instructions your script uses is displayed when you compile a script. The largest user-script we have ever seen had fewer than 300000 instructions.
- A script file can contain up to 1000000 lines and you can have up to 4095 included files. However, you will almost certainly run out of instruction space before you hit these limits.
- Everything in the script language is an object, and there is a limit of around 65000 objects. The script language itself creates around 700 objects and every global variable, constant, literal constant, user-defined object, procedure and function generates another. Temporary objects are also created when functions run (and disappear when they exit) and when evaluating expressions. When you compile a script, the number of objects appears at the top of the script window.
- The maximum number of arguments of a built-in `proc` or `func` is 20. Since version [9.05], the maximum number of arguments of a user-defined function is limited only by the total number of objects in a script, however, the first 20 arguments generate more efficient code. For backwards compatibility, use a maximum of 20 arguments.
- The maximum size of an array (product of all the array dimensions) is 100,000,000 elements.
- The maximum bins in a result view is 100,000,000 to match the maximum size of an array.
- The maximum size of a string is 100,000,000 characters (was 1,000,000 before [10.06])

We can expand these limits if anyone manages to hit them.

Script functions by topic

This section lists script commands by function.

Active cursor

Active vertical and horizontal cursor-related commands:

`CursorActive`	Set or get the active cursor mode
`CursorActiveGet`	Read back active cursor parameters
`CursorSearch`	Trigger active cursor searches from the script
`CursorValid`	Test if an active cursor search succeeded
`HCursorActive`	Set or get the active horizontal cursor mode
`HCursorValid`	Test if an active horizontal cursor is in a valid position

Analysis

These functions create new result views and analyse data into the result views.

`BinError`	Get or set error bar information
`MeasureToChan()`	Create a new Event/Marker/RealMark channel for analysis results
`MeasureToXY`	Create an XY view and associated measurement process
`MeasureX`	Set the x part of a measurement
`MeasureY`	Set the y part of a measurement
`MeasureChan()`	Add or modify a `Measurement...()` in XY or Time view
`RasterAux`	Set additional data for sorted rasters (deprecated)
`RasterGet`	Read back raster information
`RasterSet`	Set raster information
`RasterSort`	Get and set the sorting values
`RasterSymbol`	Get and set the time values for raster symbols
`SetEvtCrl`	Create result view for event correlation
`SetEvtCrlShift`	Set shift for shuffled event correlation
`SetFHst()`	Create result view for frequency histogram
`SetINTH`	Create result view for interval histogram
`SetPhase`	Create result view for phase histogram
`SetPower`	Create result view for power spectrum
`SetPSTH`	Create result view for post/peristimulus histogram
`SetWaveCrl`	Create result view for waveform correlation
`SetWaveCrlDC`	Set DC use/ignore for `SetWaveCrl()`
`SetAverage`	Create result view for waveform copy/average/sum
`SetResult`	Create result view for user-defined data
`Sweeps`	Number of items added to result view
`Process`	Process data into result view
`ProcessAll`	Process all result views attached to a time view
`ProcessAuto`	Process data automatically during sampling
`ProcessGate`	Set gating mode for the `Process()` command.
`ProcessTriggered`	Process triggered by data during sampling

Arbitrary waveform output

These commands control the output of waveforms during data sampling and offline. Waveforms are identified by a key code and up to 10 can be defined.

`PlayOffline`	Play offline via 1401 or sound card
`PlayWaveAdd`	Add waveform to the on-line play list
`PlayWaveChans`	Get or change the DAC channels of an area before sampling
`PlayWaveCopy`	Update output waveforms in 1401 after sampling starts
`PlayWaveCycles`	Get or change the repeats of a waveform

PlayWaveDelete	Delete one or more waveforms from the play wave list
PlayWaveEnable	Controls which waveforms in the list are available for playing
PlayWaveInfo$	Get waveform information including list of all waveform keys
PlayWaveKey2$	Set a second key to play an arbitrary waveform
PlayWaveLabel$	Get or set the label associated with each waveform
PlayWaveLink$	Get or set the links between waveforms
PlayWavePoints	Change the size of an area online
PlayWaveRate	Get or set the desired base wave replay rate
PlayWaveSpeed	Scale factor for the wave replay rate - can be used on-line
PlayWaveStatus$	Get information about waveform output during sampling
PlayWaveStop	Stop the current output immediately or set one more cycle
PlayWaveTrigger	Get or set the trigger state for a waveform

Array and matrix arithmetic

These functions are used with arrays and result views to speed up data manipulation. You can also use the built-in mathematical functions directly on an array.

ArrAdd	Adds an array or constant to an array
ArrConst	Copies an array, or sets an array to a constant value
ArrConv	Perform a discrete convolution
ArrCWT	Continuous Wavelet Transform of an array
ArrDiff	Replaces an array with an array of simple differences
ArrDiv	Divides an array by another array or a constant
ArrDivR	Divides array into another array or constant
ArrDot	Forms the dot product (sum of products) of two arrays
ArrFFT	Fourier transforms and related operations
ArrFilt	Applies a FIR filter to an array
ArrHist	Calculate a histogram from a data array
ArrIntgl	Integrates array; inverse of ArrDiff()
ArrMapImage	Map a matrix into a bitmap.
ArrMedian()	Find the median of an array.
ArrMul	Multiples an array by another array or constant
ArrRange	Find the range of values in an array
ArrRev	Reverse the order of data in an array
ArrSort	Sort an array and optionally order others in the same way
ArrSpline	Interpolate one array to another using cubic splines
ArrStats	Calculate mean, variance, skewness and kurtosis of an array
ArrSub	Subtract constant from array, or difference of two arrays
ArrSubR	Subtract array from constant, or reversed difference of arrays
ArrSum	Sum, mean and standard deviation of an array
Clamp	Limit a numeric array to a low to high range
Len	Returns the length of a string or array
LinPred	Linear prediction and maximum entropy method power spectra
MATDet	Calculate the determinant of a matrix
MATInv	Invert a matrix
MATMul	Multiply matrices and vectors.
MATSolve	Solve a set of linear equations
MATTrans	Transpose a matrix (also see the trans() operator)
PCA	Principal component analysis (singular value decomposition)
Spline2D	Interpolate scattered x,y points

Implementation note

Many of these routines move data from a source to a destination array. When these two arrays are of the same type there is the possibility that they can overlap (when the source and destination are part of the same array). If this is the case, and the destination starts after the source, the result will not be as expected. This is not the case for the ArrConst() command, which detects this possibility and copies data backwards if this is a problem. All the other commands where this is a problem process data forwards.

In general, when working through two arrays element by element, the routines read both arrays, do the operation, then write the result to the destination. So adding an array to itself:

```
var arr[100];
...
ArrAdd(arr, arr);     ' OK, doubles the values in arr
ArrAdd(arr, arr[1:]); ' OK, destination before source
ArrAdd(arr[1:], arr); ' Not OK
```

Binary files

Spike2 can read and write binary files. Binary files provide links to other programs and are generally more efficient than text for transferring large quantities of data.

`FileClose`	Close a file opened in binary mode
`FileOpen`	Open an external file in binary mode
`FileTimeDate`	Get times and dates associated with a file
`BRead`	Extract 32-bit integer, 64-bit real and string data from a file
`BReadSize`	Extract 8 and 16-bit integer and 32-bit real data from a file
`BRWEndian`	Change byte order for read/write of numbers in binary files
`BSeek`	Change the current file position for next read or write
`BWrite`	Write 32-bit integer and 64-bit real data to a file
`BWriteSize`	Write 8 and 16-bit integer, 32-bit real and string data to a file

Channel commands

These commands manipulate data channels.

`BinError`	Get or set error bar information
`Binsize`	The sample interval for waveforms, otherwise time resolution
`BurstMake`	Extract bursts to a memory channel from an event channel
`BurstRevise`	Modify a list of bursts in a memory channel
`BurstStats`	Collect statistics on bursts in preparation for `BurstRevise()`
`Chan$`	Get the symbolic name for a channel
`ChanCalibrate`	Calibrate waveform or WaveMark channels from known values
`ChanColour`	Override drawing mode based channel colours by palette
`ChanColourGet`	Get channel colour as RGB
`ChanColourSet`	Set channel colour override as RGB
`ChanComment$`	Get or set the channel comment
`ChanData`	Fill an array with a waveform or event times
`ChanDecorate`	Set RealMark channel drawing decoration
`ChanDelete`	Delete a channel from a time view
`ChanDuplicate`	Duplicate channels in a time view
`ChanHide`	Make a channel invisible
`ChanImage`	Set a bitmap to appear behind a channel
`ChanIndex`	Select which RealMark item to display and measure
`ChanKey`	Control sonogram and XY view keys
`ChanKind`	Get the type of a channel
`ChanLinPred`	Replace short waveform sections with predicted data to fix artifacts
`ChanList`	Get a list of channels meeting a specification
`ChanMeasure`	Make the same measurements as the Cursor Regions dialog
`ChanNew`	Create a new channel for `ChanWriteWave()` to write to
`ChanOffset`	Set waveform and WaveMark value for input of zero
`ChanOrder`	Modify the channel order and y axis grouping
`ChanPenWidth`	Set the pen width for a time or result view channel
`ChanProcessAdd`	Add new processing option to a waveform or RealWave channel
`ChanProcessArg`	Read or modify an argument of a channel process
`ChanProcessClear`	Delete a process, all processes for a channel, or all processes
`ChanProcessCopy`	Copy channel processes (rectify, smooth) to other channels
`ChanProcessInfo`	Get information about the processes attached to a channel
`ChanPort`	Get the physical sampling port of a time view data channel
`ChanScale`	Set waveform and WaveMark channel scale factor
`ChanSave`	Copy time view channels between views with optional time shift

ChanSearch	Search a channel for a feature (peak, level crossing, slope…)
ChanSelect	Select and report on selected state of channels
ChanShow	Make a channel visible
ChanTitle$	Get or set the channel title string
ChanUndelete	Recover a deleted channel in a 64-bit .smrx file.
ChanUnits$	Get or set the channel units
ChanValue	Get channel data at a particular time or x axis position
ChanVisible	Get the visibility state of a channel
ChanWeight	Change the relative vertical space of a channel
ChanWriteWave	Write data to a waveform or RealWave channel
ChanZoom()	Zoom/unzoom channels in Time and Result views
Count	Count items in a time range, sum bins in result view
DrawMode	Get or set display mode for a channel
DupChan	Get information about duplicate channels
EventToWaveform	Convert an event channel into a waveform channel
FitLine	Fit a straight line to waveform or result channel
LastTime	Find the previous item in a channel (and return values)
MarkEdit	Edit marker codes and other information
MarkInfo	Get information on extended marker types
MarkMask	Set the marker filter for a channel
MarkTrace	Set the WaveMark trace to use by default as a waveform
MarkSet	Set the marker codes for data in a time range
Maxtime	Time of last item on the channel
MemChan	Create a channel in memory
MemDeleteItem	Delete one or more items from a memory channel
MemDeleteTime	Delete one or more items based on time in a memory channel
MemGetItem	Get information on item in memory channel
MemImport	Import items into a memory channel
MemSave	Write the contents of a memory channel to the data file
MemSetItem	Edit or add an item in a memory channel
Minmax	Find minimum and maximum values (and positions)
NextTime	Find the next item in a channel (and return values)
VirtualChan	Create data channels from existing channels and maths functions

Curve fitting

These functions do least squares fits of data to functions.

ChanFit	Fit function to time, result or XY channels and display fitted line
FitData	Fit a selected function to arrays of x,y data points
FitExp	Fits multiple exponentials
FitGauss	Fit multiple Gaussian curves (normal distributions)
FitLine	Fit straight line to time or result view data
FitLinear	Fit to linear combination of user-defined functions
FitNLUser	Fit to non-linear user function
FitPoly	Fit polynomial to the data
FitSigmoid	Fit sigmoid to the data
FitSin	Fit multiple sinusoids

Click here to learn more about curve fitting.

Debugging operations

These functions can be used when debugging a script.

App(-4)	Get the number of system handles held by Spike2
Debug	Set a permanent break point and disable/enable debugging
DebugHeap()	Get information about Spike2 memory usage
DebugList	List internal Spike2 script objects
DebugOpts	Gets and optionally sets system level debugging options
Eval	Convert the argument to text and display

Digital filtering

These functions create and apply digital filters and manipulate the filter bank.

`ArrFilt`	Array arithmetic routine to apply FIR coefficients to an array
`FiltApply`	Apply a set of coefficients or a filter bank filter to a waveform
`FiltAtten`	Set the desired attenuation of an FIR filter in the filter bank
`FiltCalc`	Force coefficient calculation of an FIR filter in the filter bank
`FiltComment$`	Get or set the comment for an FIR filter in the filter bank
`FiltCreate`	Create a new FIR filter definition in the filter bank
`FiltInfo`	Retrieve information about an FIR filter in the filter bank
`FiltName$`	Get or set the name of an FIR filter in the filter bank
`FiltRange`	Get the useful sampling rate range for an FIR filter bank filter
`FIRMake`	Generate FIR filter coefficients in an array
`FIRQuick`	Generate FIR filter coefficients with desired attenuation
`FIRResponse`	Calculate frequency response of array of FIR coefficients
`IIRApply`	Apply an IIR filter bank filter to a waveform channel
`IIRBp`	Create and/or apply an IIR bandpass filter
`IIRBs`	Create and/or apply an IIR bandstop filter
`IIRComment$`	Get or set the comment for an IIR filter in the filter bank
`IIRCreate`	Create a new IIR filter definition in the filter bank
`IIRHp`	Create and/or apply an IIR high-pass filter
`IIRLp`	Create and/or apply an IIR low-pass filter
`IIRName$`	Get or set the name of an IIR filter in the filter bank
`IIRNotch`	Create and/or apply an IIR notch filter
`IIRReson`	Create and/or apply an IIR resonator filter

Editing operations

These functions mimic the Edit menu commands and provide additional functionality.

`EditClear`	Delete text from a text window at the caret
`EditCopy`	Copy the current selection to the clipboard
`EditCut`	Delete the current selection to the clipboard
`EditFind`	Find text
`EditPaste`	Paste the clipboard into the current text field
`EditReplace`	Find and replace text
`EditSelectAll`	Select the entire text or cursor window contents
`MoveBy`	Move relative to current position
`MoveTo`	Move to a particular place
`Selection$`	This function returns the text that is currently selected

Environment

These functions don't fit well into any of the other categories!

`App`	Get the program serial number
`Date$`	Get system date in a string in a variety of formats
`Error$`	Convert a runtime error code to a message string
`Profile`	Access to the system registry and to Spike2 Preferences
`ProgKill`	Terminate a process started by `ProgRun()`
`ProgRun`	Run a program, optionally position the window
`ProgStatus`	Test if a program started by `ProgRun()` is still active
`ScriptRun`	Set a script to run when the current script ends
`Seconds`	Get or set current relative time in seconds
`Sound`	Play a tone or a `.wav` file
`Speak`	Convert text to speech on systems that support this
`System`	Get system revision as number
`System$`	Get system name as a string, access environment variables
`Time$`	Get system time in a string in a variety of formats

TimeDate	Get system date and time as numbers

File system

Spike2 can read file information, change the current directory, delete and copy files and convert foreign files into Spike2 format. You can also execute external files.

FileConvert$	Convert a foreign file to a Spike2 file and open it
FileCopy	Copy one file to a new location
FileDate$	Retrieve date of creation of Spike2 data file
FileDelete	Delete one or more files
FileList	Get a list of files or directories
FilePath$	Get the current directory or directory for new data files
FilePathSet	Change the current directory or directory for new data files
FileStatus	Get read-only, hidden, directory status of a file path
FileTime$	Retrieve time of creation of Spike2 data file
FileTimeDate	Retrieve time and date of creation of Spike2 data file as numbers
FileTimeDateSet	Set the time of creation (start of sampling) of a data file

Grid views

Grid views hold a rectangular array of cells.

Draw	Set the first column to display
EditClear	Clear selected cells
EditCopy	Copy selected cells to the clipboard
EditCut	Cut selected cells to the clipboard
EditPaste	Replace selection with clipboard contents
EditSelectAll	Select entire grid
FileName$	Get associated file name
FileNew	Create a new grid view
FileOpen	Open an existing grid view
FilePrint	Print current file
FileSave	Save as current name
FileSaveAs	Save with a different name
FontGet	Get the current font characteristics
FontSet	Set the current font characteristics
GrdAlign	Set the alignment of grid columns
GrdColourGet	Get the colours for a grid cell
GrdColourSet	Set colours of column, cell or rectangular range
GrdColWidth	Set/get the width of grid columns (as pixels or text)
GrdGet	Read back the contents of grid cells
GrdSet	Set the contents of grid cells
GrdShow	Show and hide column and row headers
GrdSize	Get and set the rows and columns in the grid
Modified	Get and set the modified and read-only state of views
MoveBy	Move the selection relative to the current position
MoveTo	Move the selection to an absolute position
Selection	Find the currently selected cells
ViewStandard	Reset the formatting to the standard state
WindowSize	Size view, also set size to fit the grid
XHigh	The right-most visible column
XLow	The left-most visible column
XRange	Set the first column to display
YHigh	The top visible row
YLow	The bottom visible row
YRange	Set the top row to display

Horizontal cursors

Horizontal cursors are controlled from a script with the following commands.

`HCursor`	Set or get the position of a horizontal cursor
`HCursorActive`	Set or get the active horizontal cursor mode
`HCursorChan`	Gets the channel that a horizontal cursor belongs to
`HCursorDelete`	Delete a designated horizontal cursor
`HCursorExists`	Test if a cursor exists
`HCursorLabel`	Gets or sets the horizontal cursor style
`HCursorLabelPos`	Gets or sets the horizontal cursor position
`HCursorNew`	Add a new horizontal cursor on a channel (at a given position)
`HCursorRenumber`	Renumbers the cursors from bottom to the top of the view
`HCursorValid`	Test if an active horizontal cursor is in a valid position

Idle functions and Idle time

There are several user-interaction commands that can set up Idle functions. These are functions that get run whenever Spike2 has spare time, or when a particular event (such as a mouse move) happens. The functions that can set these up are:

`DlgAllow`	Set allowed actions and change and idle functions for `DlgShow()`
`ToolbarSet`	Add a button (and associate a function with it)
`ToolbarMouse`	Link mouse actions to user functions

Idle functions are given time to run whenever Spike2 is waiting for user input, or when you use the `Yield()` command. It helps the system to have idle time as this is also used to release resources. There is some control over how Spike2 shares idle time with the rest of the system in the Scheduler settings in the Preferences.

Note that the `Interact()` command releases idle time to the system but does not run user-defined Idle functions.

Why idle time is important

Each time you move or click the mouse, or press a key, Windows sends messages to the application that has the input focus. Windows also sends messages to an application for many other reasons. The messages accumulate in an application message queue. Windows calls the application periodically and the application is expected to extract messages from the queue and acts on them. The application returns control to Windows when the queue is empty. The application also does housekeeping as part of this processing. This housekeeping includes releasing resources that are no longer in use. All these activities happen in Idle time.

If a script runs for a while without giving idle time to the system by user interaction or by calling `Yield()`, there can be problems. If an application does not process the message queue for 5 seconds or so, the application is marked as not responding, which can mess up slow screen updates (we work around this). A worse problem is that resources are not released, and a long period of non-idle time can exhaust resources, leading to non-recoverable errors.

Prior to version [10.02] idle time was especially important when you sampled data. When a script called `SampleStart()`, Spike2 relied on idle time to set up the separate thread that transfers data into the data file.

For all versions, Spike2 uses idle time to track the current time in the sampling device(s) and to handle display screen updates as new data arrives. It is also used to update Info windows and invalid screen areas and run Idle functions.

Image related

These are the script commands that can be used with images or bitmaps:

`ArrMapImage`	Map a matrix into a bitmap using a colour scale.
`ChanImage`	Set an image to display behind data.
`ColourGet`	Collect information about a colour scale.

ColourSet	Create a colour scale.
EditCopy	Create a bitmap on the clipboard from a matrix or the current view.
EditImageLoad	Load a file image to the clipboard.
EditImageSave	Save a clipboard image to a file.
EditPaste	Copy a clipboard image to an array.
MMImage	Move a multimedia image to an array.
MMVideo	Get information about a multimedia image.
MousePointer	Create or load mouse pointer
Spline2D	Create images by splining 2D and 3D data.
XY Views	These can be used for line and point drawing.

Info windows

Info windows generate high-visibility displays for various purposes.

ColourSet	Set the application colours including for information windows
EditCopy()	Get the currently displayed text
FontGet	Get the current font used in an information window
FontSet	Set the current font used in an information window
InfoCloseAll	Closes all Info windows associated with the current view
InfoImage	Set or read back the information window background image settings
InfoOpen	Create a new information window or get the handle of existing window
InfoRun	Start, stop and reset information window timers
InfoSettings	Set or read back the current settings for an information window
InfoSpeak	Set the speech output mode and settings for an information window
ViewColourSet	Set information window colours
ViewLink	Find owner of an information window.

Mathematical functions

The following mathematical functions are built into Spike2. You can apply many of these functions to real arrays by passing an array to the function. Also, see the Array and matrix arithmetic commands.

Abs	Absolute value of a number or array
ATan	Arc tangent of number or array
BinomialC	Number of ways of choosing k from n (binomial coefficient)
Ceil	Ceiling of a number or array (next highest integral value)
Clamp	Limit a value, or a numeric array, to a low to high range
Cos	Cosine of a number or array
Cosh	Hyperbolic cosine or a number or array
Exp	Exponential function of a number or array
Floor	Floor of a number or array (next lowest integral value)
Frac	Remove integral part of a number or array
GammaP	Incomplete Gamma function, used to calculate probabilities
LinPred	Linear prediction of equally spaced data
Ln	Natural logarithm of a number or array
LnGamma	Natural logarithm of the Gamma function (use for big factorials)
Log	Logarithm to base 10 of a number or array
Max	Finds maximum of array or variables
Min	Finds minimum of array or variables
PolyEval	Evaluate a polynomial of real or complex numbers
PolyRoot	Extract the roots of a real or complex polynomial
Pow	Raise a number or an array to a power
Rand	Generate pseudo-random numbers with uniform density
RandExp	Generate random numbers with exponential density
RandNorm	Generate random numbers with normal density
Round	Round a real number to the nearest integral value
Sin	Sine of a number or array
Sinh	Hyperbolic sine of a number or array
Sqrt	Square root of a number or an array
Tan	Tangent of a number or array

Tanh	Hyperbolic tangent of a number or array
Trunc	Remove fractional part of number or array
ZeroFind	Find the roots (f(x) = 0) of an equation

Multimedia files

These function control avi files associated with data files.

MMAudio	Get information on the audio content
MMImage	Get a video image in a variety of formats
MMOpen	Open and or get the handle of a multimedia file
MMPosition	Set the play position and play state of a multimedia file
MMRate	Set the desired frames per second during recording
MMVideo	Get information about the video format
FileSaveAs	Save a video frame as an image

Multiple monitor support

These functions give the script writer control over positioning the Spike2 application and Spike2 windows in a multiple monitor environment.

DlgCreate	Position a script-controllable dialog relative to monitors.
DlgMouse	Set mouse position relative to dialog (useful with multiple monitors)
System	Get monitor count, positions and identify the primary monitor.
Window	Position a script-controllable window relative to monitors.
WindowGetPos	Get position of a window, optionally including screen information.
WindowVisible	Maximise the application over the entire desktop.

Result views

These commands manipulate result views. Result views can also be treated as arrays; so all the array arithmetic commands are available.

BinError	Get or set error bar and bin count information
Binsize	Width of each bin in the x direction
BinToX	Convert bin number to x axis value
DrawMode	Get or set display mode for a channel
DrawModeCopy()	Copy draw mode from one channel to others.
Maxtime	Number of bins in the result view
Minmax	Find minimum and maximum values (and positions)
Optimise	Set reasonable y range for display
RasterAux	Deprecated, use RasterSort() and RasterSymbol()
RasterGet	Read back raster information
RasterSet	Set raster information
RasterSort	Get and set the sorting values
RasterSymbol	Get and set the time values for raster symbols
Sweeps	Gets and sets the number of items added to result view
XHigh	Bin number of the end of the displayed area
XLow	Bin number of the start of the displayed area
XRange	Set the x axis range for next draw
XScroller	Show or hide the x axis scroll bar and controls
XTitle$	Get and set x axis title
XToBin	Convert x axis value to bin number
YAxisLock	Get and set the locking state of grouped channels
YHigh	Get upper limit of y axis
YLow	Get lower limit of y axis
YRange	Set y axis range

Runtime sampling

These commands control data sampling. There is only one new data file at a time, and these commands refer to it, regardless of the current view.

`ViewExtraTime`	Set extra time at the end of the x axis during sampling
`SampleAbort`	Stop sampling and throw data away
`SampleHandle`	Gets the view handle of sampling windows and controls
`SampleInfo`	Get data about Talker/Listener time drifts
`SampleKey`	Adds to the keyboard marker channel, controls output sequencer
`SampleReset`	Clear the current data file and restart sampling
`SampleSeqClock`	Get and optionally change the sequencer clock rate
`SampleSeqStep`	Get the current sequencer step
`SampleSeqTable`	Get and set output sequencer table data
`SampleSeqVar`	Set an output sequencer variable
`SampleSeqTick0()`	Set/get the reference time for the `TICKS` instruction
`SampleStart`	Start sampling after creating a new time view
`SampleStatus`	Get the sampling state
`SampleStop`	Stop sampling and keep the data
`SampleText`	Adds a string to the text marker channel
`SampleWrite`	Control writing data to sampling file
`TalkerSendStr`	Send a text string to a running talker
`TalkerReadStr`	Read any pending strings from a running talker

Sampling configuration

These commands set the sampling configuration to use when you create a new data file.

`FileOpen`	File type 6 loads a sampling configuration
`FileSaveAs`	File type 6 saves a sampling configuration
`OutputReset`	Control reset values for DAC and digital outputs
`SampleAutoComment`	Control automatic prompt for file comment after sampling
`SampleAutoCommit`	Controls how frequently a data file is flushed to disk
`SampleAutoFile`	Set if file automatically written to disk at end of sampling
`SampleAutoName$`	Set file name template for automatic file saving
`SampleBigFile`	Set or get the big file setting in the sampling configuration
`SampleCalibrate`	Set calibration of waveform or WaveMark channel
`SampleChanInfo`	Get information about channels in the sampling configuration
`SampleChannels`	Set or get the maximum number of channels to sample
`SampleComment$`	Set or get channel comments or Script Bar comment and label
`SampleConfig$`	Get the name of the last loaded configuration file
`SampleClear`	Sets the Sampling configuration to a known state
`SampleDebounce`	Get and set the debounce time for Event and digital marker channels
`SampleDerived`	Adds a channel to generate data derived from an existing channel
`SampleEvent`	Adds a channel to sample event data
`SampleDigMark`	Adds a channel to sample digital marker data
`SampleKeyMark`	Restore the keyboard marker channel to a sampling configuration
`SampleLimitSize`	Set or clear the size limit on a file
`SampleLimitTime`	Set or clear the time limit for sampling
`SampleMode`	Set the sampling mode (Continuous, Timed, Triggered)
`SampleOptimise`	Set methods for optimising waveform and WaveMark rates
`SampleProcess`	Add, delete and report on real-time sampling processes
`SampleRepeats`	Set the number of files to sample in sequence
`SampleScript()`	Set a script to run during sampling
`SampleSeqClock`	Get and optionally change the sequencer clock rate
`SampleSeqCtrl`	Set or get the allowed sequencer jump methods
`SampleSequencer`	Set the name of the sequencer file
`SampleSequencer$`	Get the name of the sequencer file
`SampleStartTrigger`	Set triggered or non-triggered sample start
`SampleTalker`	Adds a talker channel to the configuration
`SampleTextMark`	Adds a channel for text notes

SampleTextMarkLink	Set serial line TextMark links to sequencer jumps, PlayWave
SampleTimePerAdc	Set the base ADC conversion interval
SampleTitle$	Set and get the title of a channel
SampleTrigger	Enable peri-triggered sampling of groups of channels
SampleUsPerTime	Set and get the basic timing interval
SampleWaveform	Adds a channel to sample waveform data
SampleWaveMark	Adds a channel to sample WaveMark data

Serial line control

These functions let the script writer read from and write to COM ports. This feature can used to control equipment during data capture. If you need to capture text during sampling, you could consider using a TextMark channel. To read arbitrary binary data you can use the `SerialRead()` command.

Originally, COM ports were implemented as RS232 serial-line communication ports. Nowadays they are usually plug in USB devices that implement serial lines, but can actually be any device that transfers streams of data. There are also purely software ports (virtual COM ports) that can be used to communicate between different programs running on the same machine; these require you to install device drivers.

SerialOpen	Open and configure a port (set Baud rate, parity, data bits...), list ports
SerialWrite	Write characters to the serial port
SerialRead	Read characters from the serial port
SerialCount	Count the number of data items available to read
SerialClose	Release a previously opened serial port
SampleTextMark	Create a TextMark channel in a sampling configuration to link to a serial port.

Signal conditioner control

These functions control serial line controlled signal conditioners.

CondFilter	Get or set the conditioner low-pass or high-pass filter
CondFilterList	Get a list of possible low-pass or high-pass filter settings
CondGain	Get or set the conditioner gain
CondGainList	Get a list of the possible gains for the conditioner
CondGet	Get all the settings for one channel of the conditioner
CondOffset	Get or set the conditioner offset for a channel
CondOffsetLimit	Get or set the conditioner offset range for a channel
CondRevision$	Get or set the conditioner offset for a channel
CondSet	Single call to set all channel parameters
CondSourceList	Get names of the signal sources available on the conditioner
CondType	Get the type of signal conditioner

Spike Monitor

These function control the Spike monitor window.

SMControl	Get and set the states of the control bar items
SMOpen	Open and or get the handle of a spike monitor window
ViewLink	Get the associated time view handle
Window	Set the position of the spike shape window
WindowGetPos	Get the window position
WindowVisible	Show and hide the spike shape window

Spike shape window

These functions can be used with a spike shape window. As the window is not a view in the sense of the time, result, XY and text views, to access the view you must use code of the form: `View(ss%).XLow();` where `ss%` is the handle of the target spike shape window. To see examples, look in the scripts folder at `verifyss.s2s`, which tests script control of spike shape windows.

Cursor	Moving cursor 0 sets the time to read spikes from
FileClose	Close the spike shape window
HCursor	Set the horizontal cursor positions
Optimise	Optimise the y range and horizontal cursors
SSButton	Set and read button states in the window
SSChan	Set and get channel information, save channel configuration
SSClassify	Classify and create WaveMark data
SSOpen	Open spike shape windows and get handle of open window
SSParam	Equivalent to the spike shape parameters dialog
SSRun	Get and set the run state and speed
SSTempDelete	Delete templates based on index and code
SSTempGet	Read back template shape
SSTempInfo	Get and set template information (width, locking, counts)
SSTempSet	Create and modify templates
SSTempSizeGet	Get template size and offset in display and displayed points
SSTempSizeSet	Set template size and position and display size in points
ViewLink	Get the associated time view handle
Window	Set the position of the spike shape window
WindowGetPos	Get the window position
WindowVisible	Show and hide the spike shape window
XHigh	Get the start of the time view range to process
XLow	Get the end of the time view range to process
XRange	Set time range in associated time view to process
YHigh	Read back the y axis range
YLow	Read back the y axis range
YRange	Set displayed y range

String functions

These functions manipulate strings and convert between strings and other variable types.

Asc	ASCII code of first character of a string
Chr$	Converts a code to a one character string
DelStr$	Returns a string minus a sub-string
InStr	Searches for a string in another string
InStrRE	Searches a string using regular expressions
LCase$	Returns lower case version of a string
Left$	Returns the leftmost characters of a string
Len	Returns the length of a string or array
Mid$	Returns a sub-string of a string
Print$	Produce formatted string from variables
ReadStr	Extract variables from a string
ReadSetup	Set separators and delimiters for ReadStr() and Read()
Replace$()	Generate a string replacing occurrences of a sub-string
Reverse$()	Generate a backwards version of a string
Right$	Returns the rightmost characters of a string
Str$	Converts a number to a string
Trim	Remove leading and trailing white space or user-defined characters
TrimLeft	Remove leading white space or user-defined characters
TrimRight	Remove trailing white space or user-defined characters
UCase$	Returns upper case version of a string
Val	Converts a string to number

Talkers

These functions can be used with Talkers:

SampleTalker	Adds a talker channel to the configuration
TalkerChanInfo	Get information about a channel of a talker
TalkerInfo	Get information about talkers or a talker and load a talker
TalkerSendStr	Send a text string to a running talker

TalkerReadStr	Read any pending strings from a running talker

Text files

Spike2 can create, read and write text files. You can also open a text file into a window.

FileNew	Open a new text file in a window
FileOpen	Open a text file in a window or for reading and writing
FileSaveAs	Save a view in variety of formats, including text
Print	Write formatted output to a file or log window
Read	Extract data from a text file
ReadSetup	Set separators and delimiters for Read() and ReadStr()
TabSettings	Get or set the tab and indent settings for a text view

Text views

Text views (which includes Script views and output sequencer views) hold text organised in lines.

EditClear	Clear selected text
EditCopy	Copy selected text to the clipboard
EditCut	Cut selected text to the clipboard
EditFind	Search for text
EditPaste	Replace selection with clipboard contents
EditReplace	Replace matching selection and search for next
EditSelectAll	Select all text
FileName$	Get associated file name
FilePrint	Print current file
FilePrintVisible	Print selection or line holding text caret
FileSave	Save as current name
FileSaveAs	Save with a different name
FontGet	Get the current font characteristics
FontSet	Set the current font characteristics
Gutter	Control display of the gutter
LogHandle	Get handle of the log view
Modified	Get and set the modified and read-only state of views
MoveBy	Move the text caret relative to the current position
MoveTo	Move the text caret to an absolute position
Print	Print text at the text caret position
PrintLog	Print text to the end of the log view
Read	Read the text line containing the caret
ReadSetup	Setup how read interprets item separators
TabSettings	Get and set the Tab settings
ViewLineNumbers	Control display of line numbers
ViewMaxLines	Get and set the maximum lines in a view
ViewZoom	Get and set the zoom factor
XHigh	Get line of first visible line plus visible line count
XLow	get the first visible line in the view

Time views

These commands manipulate time views.

BinToX	Convert time units to x axis units
ChanDecorate	Set RealMark channel drawing decoration
DrawMode	Get or set display mode for a channel
DrawModeCopy()	Copy draw mode from one channel to others.
Dup	Get the view handle of duplicates of the current view
EventToWaveform	Create a waveform channel from events
ExportChanFormat	Set channel format for export

ExportChanList	Set list of channels for export
ExportRectFormat	Set rectangular time view export
ExportTextFormat	Set format for text output of channels
FileApplyResource	Apply a resource file to the current time view
FileGlobalResource	Set a global resource file
FileInfo	Get the file format version
FileSaveResource	Create a resource file matching the current time view
FileTimeBase	Get and optionally set the underlying time units
Maxtime	Maximum x axis value for any channel in the view
Optimise	Set reasonable y range for channels with axes
ReRun	Get or set the rerunning state of a channel
VerticalMark	Vertical marker display control
ViewExtraTime	Set extra time at the end of the x axis during sampling
ViewOverdraw	Add a list of overdraw trigger times to a triggered display
ViewOverdraw3D	Set the drawing parameters for overdrawn data in 3D mode
ViewTrigger	Control on-line triggered displays and off-line display stepping
VirtualChan	Create data channels from existing channels and maths functions
WindowDuplicate	Duplicate a time view
XLow	Time of the start of the displayed area in seconds
XHigh	Time of the end of the displayed area in seconds
XRange	Set the x axis range for next draw
XScroller	Show or hide the x axis scroll bar and controls
XTitle$	Get x axis title
XToBin	Convert x axis units to time units
YAxisLock	Get and set the locking state of grouped channels
YRange	Set y axis range for a channel
YHigh	Get upper limit of y axis for a channel
YLow	Get lower limit of y axis for a channel

User interaction commands

These commands exchange information with the user or let the user interact with the data.

InfoOpen	Open the Info windows.
Inkey	Return key code of last key user pressed
Input	Prompt user for a number in a defined range
Input$	Prompt user for a string with a list of acceptable characters
Interact	Allow user to interact with data
Keypress	Detect if the `Inkey()` function would read a character
MenuCommand	Simulate the user selecting a menu command
Message	Display a message in a box, wait for OK
Print	Formatted text output to a file or window
PrintLog	Formatted text output to the Log window
Print$	Formatted text output to a string
Query	Ask a user a question, wait for response
Sound	Play a tone or a `.wav` file
Speak	Convert text to speech on systems that support this
Yield	Give idle time to the system; delay for a time
YieldSystem	Surrender current time slice and sleep Spike2

You can build simple dialogs, with a set of fields stacked vertically or you can build free-format dialogs (but with more work to define the positions of all the dialog fields):

DlgAllow	Set allowed actions and the change and idle call-back functions
DlgButton	Add buttons to the dialog and modify existing buttons
DlgChan	Define a dialog entry as prompt and channel selection
DlgCheck	Define a dialog item as a check box
DlgCreate	Start a dialog definition
DlgEnable	Enable and disable dialog items in change or idle functions
DlgGetPos	Get the dialog position in change and idle functions
DlgGroup	Position a group box in the dialog
DlgImage	Add an image to a dialog
DlgInteger	Define a dialog entry as prompt and integer number input

DlgLabel	Define a dialog entry as prompt only
DlgList	Define a dialog entry as prompt and selection from a list
DlgMouse	Set the mouse pointer position when the dialog opens
DlgProgress	Set a progress bar controlled by DlgValue()
DlgReal	Define a dialog entry as prompt and real number input
DlgShow	Display the dialog, get values of fields
DlgSlider	Define a dialog entry as a slider for integer number input
DlgString	Define a dialog entry as prompt and string input
DlgText	Define a fixed text string for the dialog
DlgValue	Gives access to dialog fields in change and idle functions
DlgVisible	Show and hide dialog items in change or idle functions
DlgXValue	Define a dialog entry to collect an x axis value

These commands control the various toolbars and link script functions to the toolbar:

App	Get the view handle of the toolbars
MousePointer	Add extra mouse pointers for use with the toolbar
Toolbar	Let the user interact with the toolbar
ToolbarClear	Remove all defined buttons from the toolbar
ToolbarEnable	Get or set the enables state of toolbar buttons
ToolbarMouse	Link mouse actions to user functions
ToolbarSet	Add a button (and associate a function with it)
ToolbarText	Display a message using the toolbar
ToolbarVisible	Get or set the visibility of the toolbar
SampleBar	Controls the sample toolbar buttons
ScriptBar	Controls the script toolbar buttons

These commands can be used to select files interactively:

FileApplyResource	Select a resource file to apply to the current view.
FileList	Select one or more files for processing by a script
FileOpen	Open a file of a defined type.
FilePathSet	Select the folder for file operations.
FileSave	Save a view-based document as a file in its native format.
FileSaveAs	Save a view-based document in a variety of file types.
FileSaveResource	Save resources associated with a view.

Vertical cursors

The following commands control the vertical cursors.

Cursor	Set or get the position of a cursor
CursorActive	Set or get the active cursor mode
CursorActiveGet	Read back active cursor parameters
CursorDelete	Delete a designated cursor
CursorExists	Test if a cursor exists
CursorFlags	Control interactive cursor use
CursorLabel	Set or get the cursor label style
CursorLabelPos	Set or get the cursor label position
CursorNew	Add a new cursor (at a given position)
CursorOpen	Open the value and regions dialogs, test if open
CursorRenumber	Renumber the cursors in ascending position order
CursorSearch	Trigger active cursor searches from the script
CursorSet	Set the number (and position) of vertical cursors
CursorValid	Test if an active cursor search succeeded
CursorVisible	Get or set cursor visibility

Windows and views

These commands open, close, manipulate, position, display, size, colour and create windows (views). These commands apply to windows in general. See the sections on Time, Result, XY and Text windows for more specific commands.

App	Get the application view handle and version and special views
ChanNumbers	Hide and show channel numbers
Colour	Get or set the palette entry associated with a screen item
Draw	Draw invalid regions of the view (and set x axis range)
DrawAll	Update all invalid regions in all views
EditCopy	Copy window to clipboard
FileClose	Closes a window or windows
FileComment$	Gets and sets the file comment for time views
FileConvert$	Convert a foreign file to a Spike2 file and open it
FileName$	Gets the file name associated with a window
FileNew	Opens an output file or a new text or data window
FileOpen	Opens an existing file (in a window)
FilePrint	Prints a range of data from the current view
FilePrintVisible	Prints the current view
FilePrintScreen	Prints all text-based, time and result views
FileQuit	Closes the application
FileSave	Save a view
FileSaveAs	Save a view in variety of formats
FontGet	Read back information about the font
FontSet	Set the font for the current window
FocusHandle	Get handle of script-controllable window with the focus
FrontView	Get or set the front window on screen
Grid	Get or set the visibility of the axis grid
LogHandle	Gets the view handle of the log window
Modified	Get and set the modified and read-only state of views
PaletteGet	Get the RGB colour of a palette entry
PaletteSet	Set the RGB colour of a palette entry
View	Change or override current view and get view handle
ViewColour	Override application colours for a view using the palette
ViewColourGet	Get view item colours as RGB
ViewColourSet	Override application colours for a view as RGB
ViewExtraTime()	Sets extra time to display at X axis end during sampling
ViewFind	Get a view handle from a view title
ViewKind	Get the type of the current view
ViewLink	Get the owner of the current view
ViewList	Form a list of handles to views that meet a specification
ViewStandard	Returns a window to a standard state
ViewUseColour	Get and set monochrome/colour use for view
Window	Sets the window size and position
WindowGetPos	Get window position
WindowSize	Changes the window size
WindowTitle$	Gets or changes the window title
WindowVisible	Sets or gets the visibility of the window (hide/show)
XAxis	Get or set visibility of the x axis
XAxisMode	Control how the x axis is drawn
XAxisStyle	Control x axis tick spacing and time view `hh:mm:ss` mode
XScroller	Get or set visibility of x axis scroll bar
XUnits$	Get units of the x axis
YAxis	Get or set the visibility of the y axis
YAxisMode	Control how the y axis is drawn
YAxisStyle	Control y axis tick spacing

XY views

XY views hold from 1 to 2000 channels of (x, y) co-ordinate pairs that can be displayed in a variety of styles. Most functions that work for views in general will work for an XY view. The maximum number of channels was 256 before Spike2 version 9. The following are the XY view specific commands.

ChanKey	Control keys in XY views and sonograms
MeasureToXY	Create an XY view and associated measurement process
XYAddData	Add data points to a channel of an XY view
XYColour	Set the colour of a channel

XYCount	Return the number of XY data points in a channel
XYDelete	Delete one or more data points from a channel
XYDrawMode	Control how a channel is drawn
XYGetData	Read data back from an XY channel
XYInChan	Count and get points in a shape defined by a channel
XYInCircle	Count the number of XY points within a circle
XYInRect	Count the number of XY points inside a rectangle
XYJoin	Get or set the point joining method
XYKey	Control the display of the XY view channel key
XYRange	Get rectangle containing one or more channels
XYSetChan	Create or modify an XY channel
XYSize	Get or set maximum size of an XY channel
XYSort	Change the sort (and draw) order of data in a channel
XYZOrder	Change the drawing order of channels

1401 control

These functions communicate with a 1401 when Spike2 is not sampling. These functions will communicate with any 1401 that is supported by the 1401 device driver; it is not limited to the 1401 types that are supported for sampling in Spike2.

U1401Close	Release the 1401 for use by Spike2 sampling.
U1401Ld	Load 1401 commands into the 1401 from disk files.
U1401Open	Open a 1401 for use by the other U1401xxx commands.
U1401Read	Read a text response from a 1401, optionally convert to numbers.
U1401To1401	Transfer an integer array to the 1401 from the host.
U1401ToHost	Transfer an integer array to the host from the 1401
U1401Write	Write a text string to the 1401

Alphabetical script function index

A

Abs()

This evaluates the absolute value of an expression as a real number. This can also form the absolute value of a real or integer array (replacing each array element by the absolute value of the element). If an integer array holds the maximum possible negative integer (-2^{63}) it is replaced by the maximum positive integer ($2^{63} - 1$) and the return value is negative.

```
Func Abs(x|x[]{[]...});
```

x A real number, or a real or integer array

Returns If `x` is an array, this returns 0 if all was well, or a negative error code if integer overflow was detected (the minimum integer value has no positive equivalent). Otherwise it returns `x` if `x` is positive, otherwise `-x`.

See also:
`ATan(), Cos(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan(), Trunc()`

App()

This returns view handles for system specific windows and program information.

```
Func App({type%});
```

`type%` Negative values return application specific information:

- -1 The program revision multiplied by 100.
- -2 The highest `type%` that returns a value.
- -3 The program serial number.
- -4 The number of system handles held by the application (for debugging) or 0 if this is not supported. Added at 6.08.
- -5 The number of GDI (graphic) objects used by the application (for debugging) or 0 if this is not supported. Added at 6.08.
- -6 Indicates if this is a 64-bit build of Spike2 (1) or a 32-bit build (0).
- -7 The number of User graphic objects used by the application (for debugging) or 0 if this is not supported. Added at version 8.05.

The remaining values return view handles. If `type%` is omitted, 0 is used.

0	Application	4	Edit toolbar	8	Sample control panel
1	System toolbar	5	Play wave bar	9	Sequence control panel
2	Status bar	6	Script bar	10	Sample Status bar
3	Running script	7	Sample bar		

Returns The requested information or a handle for the selected window. If the requested window does not exist, the return value is 0. It is possible for `App(3)` to return 0 if the running script has been unloaded from memory while it is executed. This can happen if it is started with `ScriptRun()`.

Do not confuse the system toolbar with the script toolbar controlled with the `Toolbar()` command.

Example

The following can be used to hide all the floating windows at the start of a script and restore them at the end.

```
var gFloat%[20];              'global for floating window states
proc HideAll()
var i%;
gFloat%[0] := App(-2);        'number of windows
for i% := 1 to gFloat%[0] do  'hide all windows and save state
   gFloat%[i%] := View(App(i%)).WindowVisible(0);
   next;
end

proc RestoreAll()
var i%;
for i% := 1 to gFloat%[0] do  'restore hidden windows
   View(App(i%)).WindowVisible(gFloat%[i%]);
   next;
end
```

See also:

`View(), Dup(), LogHandle(), System(), Window()`

ArrAdd()

This function adds a constant or an array to a real or integer array.

```
Func ArrAdd(dest[]{[]...}, const source[]{[]...}|value);
```

`dest` The destination array (1 to 5 dimensions). Assumed not to overlap destructively with `source` (if used).

source A real or integer array with the same number of dimensions as dest. If corresponding dimensions differ in size, the smaller size is used for each dimension. This array is not modified.

value A value to be added to all elements of the destination array.

Returns The function returns 0 if all was well, or a negative error code for integer overflow. Overflow is detected when adding a real array to an integer array and the result is set to the nearest valid integer.

In the following examples we assume that the current view is a result view:

```
var fred[100], jim%[200], two[3][30], add[3][30];
ArrAdd(fred[],1.0);         'Add 1.0 to all elements of fred
ArrAdd(fred[],jim%[]);      'Add elements 0-99 of jim% to fred
ArrAdd([],fred[10]);        'Add fred[10] to result channel 1
ArrAdd(two[][], add[][]);   'add corresponding elements
```

See also:
ArrConst(), ArrDiff(), ArrDiv(), ArrDivR(), ArrDot(), ArrFFT(), ArrFilt(), ArrIntgl(), ArrMul(), ArrSub(), ArrSubR(), ArrSum(), Len()

ArrConst()

This function sets an array or result view to a constant value, or copies the elements of an array or result view to another array or result view. You can copy number or strings. It is an error to attempt to copy numbers to a string array, or strings to a numeric array.

```
Func ArrConst(dest[]{[]...}, const source[]{[]...}|value);
```

dest The destination array of 1 to 5 dimensions of any type (real, integer, string).

source An array with same number of dimensions as dest. If the arrays have different sizes, the smaller size is used for each dimension. This array is not modified.

value A value to be copied to all elements of the destination array.

Returns The function returns 0, or a negative error code. If an integer overflows, the element is set to the nearest integer value to the result.

Implementation note: When used with two arrays of the same type there is the possibility that the arrays will overlap in a way that makes a forward element by element copy overwrite data that is about to be copied. If this is the case, the data is copied backwards. This is the *only* array arithmetic routine that allows for this possibility.

In the examples below the indices j and i mean repeat the operation for all values of the indices. a1d and b1d are vectors, a2d and b2d are matrices. The arrays and value must either both be numeric, or both be strings.

Function	Operation
ArrConst(a1d[], value);	a1d[*i*] := value
ArrConst(a1d[], b1d[]);	a1d[*i*] := b1d[*i*]
ArrConst(a2d[][], value);	a2d[*j*][*i*] := value
ArrConst(a2d[][], b2d[][]);	a2d[*j*][*i*] := b2d[*j*][*i*]

See also:
ArrAdd(), ArrDiff(), ArrDiv(), ArrDivR(), ArrDot(), ArrFFT(), ArrFilt(), ArrIntgl(), ArrMul(), ArrSub(), ArrSubR(), ArrSum(), Len()

ArrConv()

This function performs the discrete convolution of a signal f with a pulse g. Convolutions arise when you want to know the effect of passing a signal through a process. If this process is a filter, the g pulse is the impulse response of the filter. There is a nice description of the discrete convolution here. Convolutions can be calculated using the Fast Fourier Transform (FFT) or by summing array products.

```
Func ArrConv(conv[], const f[], const g[]{, flags% {,nOff%}});
```

conv A vector to hold the result. If this vector follows the rules for being resizeable with the resize command, it can be any length and the command will resize it to the correct length for the result.

Otherwise, it must be the correct size for the result, which is `Len(f)` or `Len(f)+Len(g)`, depending on `flags%`. This can be the same vector as `f`.

`f` A vector that we think of as the signal. This is not changed by the operation.

`g` A vector we think of as the pulse. This is not changed by the operation.

`flags%` This controls how we perform the convolution and the size of the result. If omitted it takes the value 0, which performs the calculation using FFTs and returns a result which is `Len(f)+Len(g)` long. It is the sum of:

1 This causes the calculation to be done by summing all the array products instead of using FFTs. This can be faster than using FFTs when `g` is short compared to `f`.

2 The result array is the same length as `f` instead of `Len(f)+Len(g)`. You may also wish to set `nOff%` to determine which points of the convolution to copy to the result.

`nOff%` An optional argument used when 2 is added to `flags%`. If present it must be in the range 0 to `Len(g)-1`. If omitted it takes the value `Len(g)/2`. It is used to determine how many points of the convolution result of length `Len(f)+Len(g)` to skip before copying the result to `conv[]`.

Returns If all went well, the return value is the length of the result array, `conv[]`. Otherwise it returns a negative error code.

Details

The discrete convolution can be expressed generally as:

$$result[n] = Sum_m(f[m]*g[n-m]) = Sum_m(f[n-m]*g[m])$$

where `m` ranges from minus to plus infinity and elements of `f` and `g` that do not exist have the value 0. From this definition, you can see that `f` and `g` are interchangeable. This is also the case in our implementation, however the code that does the calculation without using the FFT is optimised on the assumption that `Len(g) <= Len(f)`.

There are two ways commonly used to calculate the convolution:

1. By computing the sums as described above. This is a relatively slow operation, of order (`n*m`) operations (written as O(`nm`) where `n` and `m` are the sizes of the arrays).
2. By taking advantage of the Convolution theorem which takes O(`N` Log `N`) operations where `N` is the next power of two that is greater than or equal in size to `Len(f)+Len(g)`. This takes the forward FFT (Fast Fourier Transform) of the two source arrays (after zero-padding them to `N` points), takes the product of the transforms, then takes the inverse FFT of the result and adjusts it for the zero padding. The zero padding requirement is because the FFT we use is not implemented for sizes other than powers of 2.

ArrFilt command

The `ArrFilt()` script command is optimised to apply an FIR digital filter to a vector, and behaves in a very similar way to using this command with `flags%` set to 3 and the `g` vector set to the `coef[]` array backwards. There are also differences caused by how we treat the first and last points to allow for end effects. `ArrFilt()` is optimised for the case of `Len(f)` much greater than `Len(g)`.

ArrCWT()

This command computes the Continuous Wavelet Transform (CWT) of a sequence of real value for several commonly-used Wavelet functions. The CWT transforms an input into the amplitudes and positions of a Wavelet at a variety of scales. The output of this command can be used as the input to the `ArrMapImage()` command to create a bitmap that displays the result of the transform. You can add the bitmap to an XY view to display the result using the `ChanImage()` command. See this example.

The input to the command is a vector of `n%` real values `src[n%]`. The output is a matrix of real or complex values `dest[n%][s%]` where `s%` is the number of scales. The scales are arranged to increase in an exponential manner; they are not linearly increasing. Each output `dest[][s%]` holds results for one scale.

The command has several variants: a setup variant to set the type of transform, a variant to return information and variants to apply the transform.

Setup transform

This variant is used first to set the type of output requires, the wavelet family to use and to determine the range of scales that are to be generated. The default values for omitted arguments, or for all arguments if you never call this setup routine, the default are chosen so that the output is likely to be useful in all cases.

```
Func ArrCWT(wav%{, dScl{, minScl{, m%|w0}}}});
```

`wav%` This determines the wavelet family to use for the transform. You have a choice of:

Value	Family	Output	Description
0	Morlet	Complex	A sinusoid windowed by a Gaussian. The `w0` argument sets the angular frequency (the ω_0 parameter), the default is 6. This is the default wavelet family if you do not call this setup function.
1	Paul	Complex	The `m%` argument sets the order, the default is 4.
2	DOG	Real	Derivative of a Gaussian. The `m%` argument sets the order of the derivative. The default is 2, which is the Mexican Hat wavelet.

`dScl` This determines the ratio of successive scales. The scale of the j^{th} scale is given by $s_j = s_0 2^{j*\texttt{dScl}}$. where s_j is the wavelet size in points. The s_0 value is set by the next argument (`minSc%`). Put another way, `1/dScl` is the number of scales to double the scale length. The default value of `dScl` is 0.25 (4 scales per octave).

`minScl` This determines the minimum scale value for the result and is width of the smallest wavelet, in points. The default value is 2, which is probably the minimum value that can reasonably be used. Values are limited to a minimum of 1.

`w0` This sets the angular frequency of the Morlet wavelet. We limit the values to the range 2 to 20. The default value (6.0) is reasonable.

`m%` This sets the order of the Paul and DOG wavelets. We limit the values to the range 1 to 10. The default values are 4 (Paul) and 2 (DOG).

Returns 0 for success or -1 if the combination of wavelet family and output format do not make sense or one of the arguments is out of range.

Read back information

```
Func ArrCWT(get%{, maxScl|scale|freq|index{, sInt}});
```

`get%` A negative value that selects information to be returned:

Value	Returned information
-1	The wavelet family as set by the `wav%` argument.
-2	The ratio of successive scales, as set by `dScl`.
-3	The minimum scale as set by `minScl`.
-4	The `wPar` argument value for the current settings.
-5	You must provide the `maxScl` argument. This returns the number of scales required to span the range `minScl` up to and including `maxScl`. This is provided as a convenience and is calculated as: `Ceil(Ln(maxScl/minScl)/(Ln(2) * dScl))`. The maximum sensible scale is the number of points in the input waveform, although setting a scale of this size is unlikely to be useful. The maximum scale size in most cases is likely to be some fraction of this. The result is -1 if the scale value is
-6	Peak frequency that corresponds to a given scale. You must supply the `scale` and optionally `sInt`. The power spectrum of each wavelet spans a range of frequencies; the calculated value is the peak of the power spectrum. In the case of a Morlet wavelet, this is the frequency of the sinusoid. The result is 0 if `scale` or `sInt` are zero or negative.
-7	Scale that corresponds to a given peak frequency. You must supply the `freq` and optionally `sInt`. See the -6 option for the meaning of frequency. The result is 0 if `freq` or `sInt` are zero or negative.

-8 Cone of influence value for a given scale. You must provide the `scale`. Due to edge effects, the first and last points of the results at each scale are unreliable. If `sInt` is omitted the returned value is the number of unreliable points at the start (and end) of the results for this scale value, otherwise it is the duration of the unreliable region in x axis units.

-9 The index of the last dimension of the output matrix (`dest[][index]` or `dest[][][index]`) that corresponds to the input `scale` value. The result is -1 if the scale is a stupid value (zero or negative). It will also be negative if `scale` is less than the `minScl` value.

-10 The scale value that corresponds to the last output matrix dimension (`dest[][index]` or `dest[][][index]`) . The result is 0 if the index is negative or stupidly large.

`maxScl` Used when `get%` is -5 to set a scale that is greater than `minScl`. This is the width of the largest wavelet, in points, that you wish to have values calculated for.

`scale` The wavelet scale to use for the -6, -8 and -9 `get%` values.

`freq` A frequency in Hz to use for the -7 `get%` value.

`index` The index into the last dimension of the output array to convert to a scale when `get%` is -10.

`sInt` The interval between the source item values (and defaults to 1 if omitted). If the source data comes from a waveform channel in a time view, this will be in seconds and should be set to `BinSize(chan%)`. This is normally supplied for `get%` values -6 and -7 when converting between a scale and a frequency. If you omit it, frequencies become per point, not per second, and the cone of influence value is returned in points, not x axis units.

Returns The value requested.

Apply the transform

This variant applies the Continuous Wavelet Transform to a source vector to generate a matrix of results. The time to process this data is approximately proportional to `nData * Log2(nData) * nScale`, where `nData` is the number of source data points, and `nScale` is the number of output scales to calculate. To give you an idea of the time, on my Core i-7 4770K 3.5 GHz CPU, to process 60000 points to 110 scales took 518 ms. A Core i-5 laptop I also use takes maybe 50% longer.

```
Func ArrCWT(const src[], dest[][]{, out%});
Func ArrCWT(const src[], dest[2][][]{, out%});
```

`src` The data to be transformed. The length of this array, henceforth `n%`, sets the size of the first dimension of the `dest` matrix. This data is typically read from a data channel using the `ChanData()` script command. The length must be at least 8 points and is currently limited to a maximum of 262144 points (which is 2 to the power 18).

`dest` The destination matrix to hold the result of the transform. If the output of the transform is purely real, this array has two dimensions, `dest[n%][s%]` , where `n%` is the length of the `src` array and `s%` is the number of scales. If the transform output is complex, or is set to the amplitude and phase of the result, `dest` has three dimensions, and the first dimensions is of size at least 2.

`out%` If present, this sets the desired output format (which must be compatible with `dest`). You have a choice of:

Value	Result	Description
0	`[n%][s%]`	The absolute value of a complex transform (Morlet or Paul) or the real result of a Real transform (DOG). This is the default value if `dest` is a matrix (has 2 dimensions).
1	`[n%][s%]`	The power of the result.
2	`[2][n%][s%]`	The result of a complex transform as pairs of Real and Imaginary components at each output point. `[0][i%][j%]` is the Real component, `[1][i%][j%]` is the imaginary component. This is the default value if `dest` has three dimensions.
3	`[2][n%][s%]`	The result of a complex transform as amplitude and phase. `[0][i%][j%]` is the amplitude, `[1][i%][j%]` is the phase.

Return 0 for success or -1 if `out%` is not compatible with `dest`.

See also:
`ArrMapImage(), ChanData(), ChanImage()`

The Continuous Wavelet Transform

The Continuous Wavelet Transform (CWT) of a sequence of N data points x_0, x_1... x_N is:

$$W_n(s) = \sum_{n'=0}^{N-1} x_{n'} \overline{\psi} \left[\frac{(n'-n)\partial t}{s} \right]$$

where *n* is the index of the resulting entry in the CWT vector at scale *s*, *n'* is the translation along the x axis, ∂t is the sampling interval along the x axis, and ψ is an admissible wavelet function and a bar means complex conjugate. Applying the CWT in this form is computationally expensive so we use the Convolution theorem to decrease the order of operations from $O(n^2)$ to $O(n\log(n))$ per scale.

The term admissible means that the wavelet function must have a mean value of zero and be localised in both the x axis space and in frequency space. A smoothly changing, non-infinite waveform that tapers to zero at each end and has a mean of zero will satisfy these conditions.

The scale of a wavelet can be thought of as its x axis width. The larger the scale, the more x axis space it occupies. To make wavelets at different scales comparable, they are normalised so that the power of the wavelet at each scale is unity.

You can find a detailed explanation by following this link. This article, "A Practical Guide to Wavelet Analysis" by C. Torrence and G. P. Compo, 1998, is well worth reading and contains a detailed description of the methodology.

Wavelet families

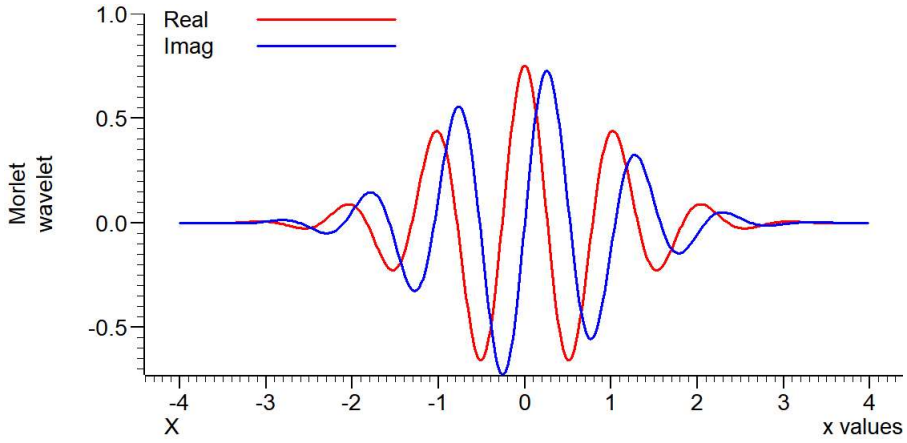
We support three wavelet families: Morlet, Paul and DOG (Derivative of Gaussian). You can find a detailed explanation of these and the methodology used in this paper. In the equations, below, the η is a dimensionless parameter (most often interpreted as time) and is the x axis value in the images.

Morlet

This is a complex wavelet given by the equation:

$$\pi^{-1/4} e^{i\omega_0 \eta} e^{-\eta^2/2}$$

The ω_0 parameter is the angular frequency (the `w0` argument to the set up call). The image below shows the Morlet wavelet with the default value of ω_0, which is 6. The wavelet is composed of a sinusoid multiplied by a Gaussian. If you are interested in the frequency content of a signal, for example if you are looking for the equivalent of a sonogram, then this is the wavelet family to use.

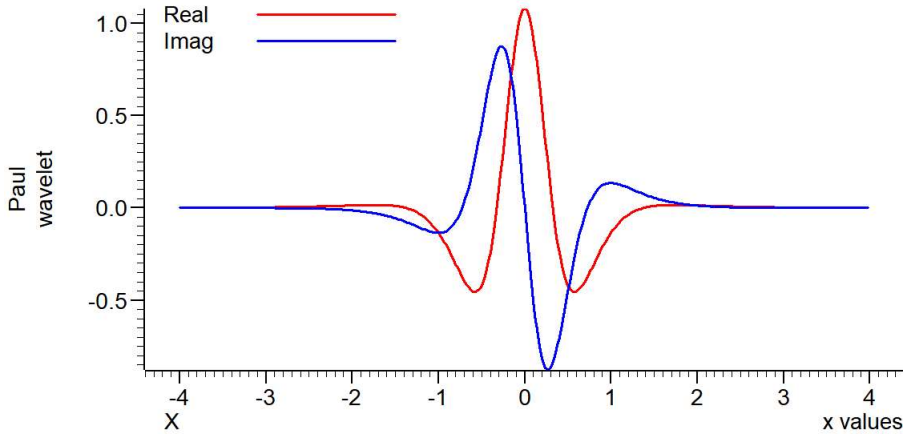


Paul

This is a complex wavelet given by the equation:

$$\frac{2^m i^m m!}{\sqrt{\pi(2m)!}}(1-i\eta)^{-(m+1)}$$

The *m* parameter is an integer and is set by the `m%` argument of the setup call. The image shows the Paul wavelet for the default value of *m* (4).

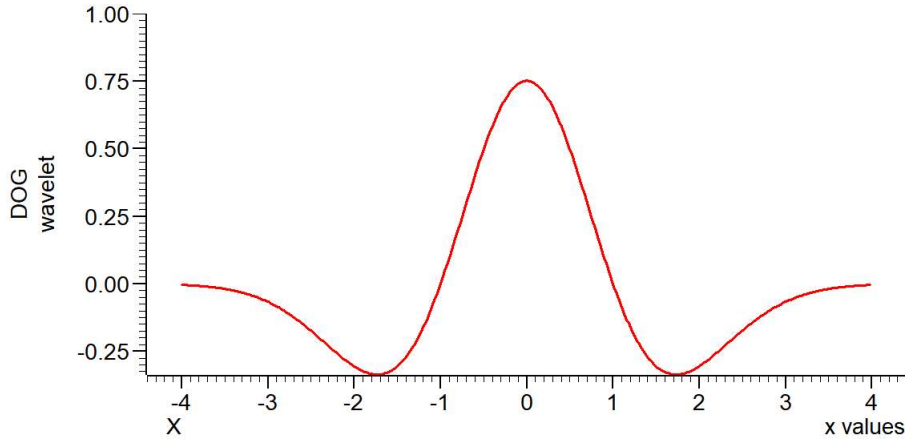


Derivative of Gaussian (DOG)

This is a wholly real wavelet given by the equation:

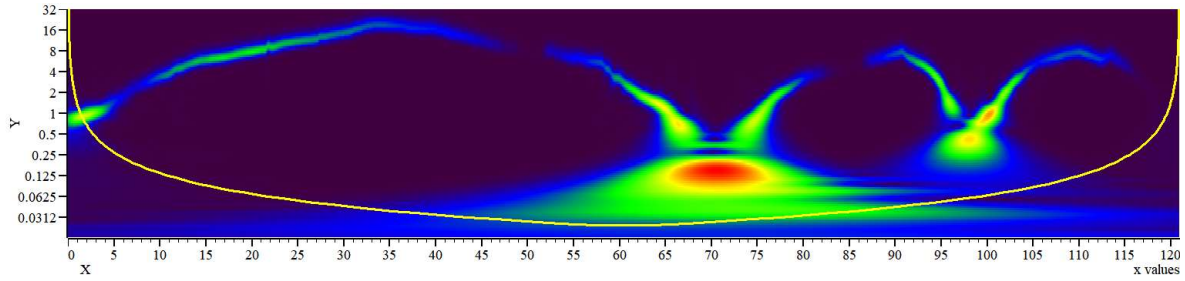
$$\frac{(-1)^{m+1}}{\Gamma\left(m+\frac{1}{2}\right)}\frac{d^m}{d\eta^m}\left(e^{-\eta^2/2}\right)$$

The *m* parameter is an integer and is set by the `m%` argument of the setup call. The image shows the DOG wavelet for the default value of m (2), which generates the so-called Mexican Hat wavelet.



Example

This example script demonstrates how to use the `ArrCWT()` command to generate the following image from the `demo.smr` data file supplied with the Spike2 distribution.



This example displays the Continuous Wavelet Transform of the waveform channel from the file using the Morlet wavelet. The Y axis is showing frequency in Hz and the x axis the time in seconds. The yellow line shows the *Cone of Influence*, which is discussed below. We have broken the scrip up into several parts. Paste the parts (in order) into an empty script view to generate the full script. The image shows significant energy at very low frequencies. This is because the data has a large DC offset. If you add a DC Remove channel process to the waveform channel, then run the analysis again, the very low frequency energy is removed.

Get the waveform data and setup values

The first part of the script opens the data file and fills an array with the waveform data to transform:

```
'Open data file and extract data array
var dataPath$:=FilePath$(-4)+"\\data\\demo.smr";
var vh% := FileOpen(dataPath$, 0, 1); if (vh% < 0) | ViewKind() then Message("No view"); halt endif

const ch% := 1;                         'Data channel in the view
var wavFamily% := 0;                    'Choice of wavelet family, 0=Morlet
var minScl := 3;                        'Sets minimum scale to start the analysis at
var dScl := 0.1;                        'Sets 10 scales per frequency doubling

var sInt:= BinSize(ch%);                'Data channel sampling interval
var time := MaxTime();                  'How far we need the analysis to go
var nX% := time / sInt;                 'Maximum size of array assuming contiguous data
var maxScl := nX%/2;                    'Maximum scale for the analysis
var a[nX%];                             'Array to analyse
ChanData(ch%, a, 0, time);              'Fill array from chosen channel
```

This first couple of lines open the `demo.smr` data file and stop the script with a message if it is not found.

The next lines set the channel to analyse, set the wavelet family to use, the minimum wavelet scale to process and the number of scales to generate per octave (doubling of frequency). The minimum scale is expressed in terms of data points, that is a wavelet with 3 data points. You might choose to start the analysis at a higher scale to save calculation time (it costs the same time to analyse each line, so analysis time is proportional to the number of scales analysed).

The last group of lines extract information from the file that we use for analysis parameters, sets the maximum scale to generate (in this case being half the data length, in points, but it could be quite a bit less), and allocates a data array large enough to hold the waveform data in the file, and then reads data into the file. A more rigorous script would check that the `ChanData()` command really did read `nX%` data points.

Perform the Continuous Wavelet Transform

This part of the script sets up the transform, allocates memory for the result, and runs the analysis:

```
'Do CWT analysis
ArrCWT(wavFamily%, dScl, minScl, 6);    'Setup Morlet with w0 of 6
var nScl% := ArrCWT(-5, maxScl);        'Number of scales to span minScl-maxScl
var m[nX%][nScl%];                      'Matrix to store the transform
ArrCWT(a, m, 0);                        'Transform the array
```

The first line prepares the `ArrCWT()` command by setting the wavelet family to use, the scale growth between successive output lines, the scale of the first output line and the parameter to define which wavelet of the selected family to use. This example is set to use the Morlet family of wavelets, which is particularly well-suited when you want to extract frequency information from the raw data as the wavelet is the product of a sinusoid and a Gaussian.

The second line gets the number of lines of output that are required to span the range of scales from `minScl` to `maxScl`. In this case we have stopped the analysis at a scale of half the number of input data points. You should note that the larger the scale, the more the result is smeared in time, so if you were not interested in the results at low frequencies (large scales), you might choose to stop at a smaller scale.

The third line generates the matrix to hold the result. For each point in the source data array we get `nScl%` results, one for each scale.

The last line generates the results as the absolute value of the result (which is complex for a Morlet wavelet).

Draw the matrix as a bitmap and place it in an XY view

The results are a little difficult to interpret as numbers; it is much easier to see what is going on in a picture. The next section of the script generates a bitmap from the results and then displays it as the background image of an XY view.

```
'Draw the matrix using ArrMapImage()
ArrMapImage(m, "<CB>");                 'Make a bitmap in the clipboard
FileNew(12, 1);                         'Create new XY view
XRange(0, nX%*sInt);                    'Set display x range to match source data
var fMax := ArrCWT(-6, minScl, sInt);   'Convert min scale to max freq
var fMin := ArrCWT(-6, maxScl, sInt);   'and max scale to min freq
YRange(-1, fMin, fMax);                 'Set display range (all channels)
YAxisAttrib(-1, 1);                     'Set the Y axis as logarithmic
YAxisStyle(-1, 0, Log(2));              'Set Y axis major ticks at powers of 2
ChanImage(1, "<CB>");                   'Set the clipboard image in the view
var ret% := ChanImage(1, 2, 1.0, 0, fMin, nX%*sInt, fMax);  'Place the image
```

The first line converts the matrix holding the results into a bitmap and copies it to the clipboard. We have chosen to use the default parameters, so this has found the largest and smallest values in the result matrix and mapped these onto the Rainbow colour scale. By setting different arguments you can set other colour scales and define you own range of values to map onto the colour scale.

The second line creates an XY view and makes it visible. The view has one, empty data channel that we use in the final code section (below) to draw the Cone of Influence. A more responsible script would save the view handle of the XY view and check that there was no error when creating it; we omit this here to keep the code short.

The third line sets the x axis range to display, which is set to match the time range of the original data.

Lines 4-8 of this section generate a logarithmic y axis, labelled in powers of two. To do this the code asks the `ArrCWT` command for the frequencies that match the minimum and maximum scales. We then set the y axis to display this range. However, if we stop at this point, the display will show a linear axis. However, we know that the bitmap shows scales that increase by a constant factor for each line. We can express this as a logarithmic axis. The `YAxisAttrib()` command switches to a logarithmic axis, which will be in powers of 10, by default. For this data file this generates major ticks at 0.1, 1 and 10 Hz. To illustrate how to choose other tick marks, the `YAxisStyle()` command sets the major ticks at powers of two, and the result for this file shows ticks at 32, 16, 8, 4, 2 and so on.

The last two lines set a channel image as the image on the clipboard generated by `ArrMapImage()` and the final line ties this image to specific co-ordinates in the view, rather than being a general background.

Draw the *Cone of Influence*

The final section draws the so-called *Cone of Influence*. There are edge effects introduced into the CWT due to the mathematical short-cut of using the Fourier Transform to do the calculation. The problem is that the Fourier transform assumes that the raw data is cyclic, but this is not the case. Further, the Fourier Transform we use works on a power of 2 data points, so we have to extend the raw data to the next power of two points by zero-padding. This will introduce discontinuities into the data which affects the results. Of course, if your data naturally reaches 0 at the start and end, these effects are greatly mitigated. The *Cone of Influence* marks the region of the result where the results are unreliable due to this effect. The final part of the code draws a line (in yellow) to illustrate this. Areas of the image below this line should be considered unreliable.

```
'Draw Cone of Influence to XY view channel 1
var i%, coi, freq;
var dLimit := (nX%-1) * sInt;           'x axis at right hand side of image
for i% := 0 to nScl%-1 do               'Loop over scales for left side of the COI
    coi := CalcCOI(i%, freq);           'Calculate COI and associated frequency
    if (2*coi >= dLimit) then break endif;
    XYAddData(1, coi, freq);            'Add this COI point
next;

while (i% > 0) do                       'Draw the right side of the COI
    i% := i% -1;
    coi := CalcCOI(i%, freq);           'Calculate COI and associated frequency
    XYAddData(1, dLimit-coi, freq);     'Add this COI point
wend;

XYDrawMode(1,2,0);                      'Set data points to size 0
XYDrawMode(1,4,3);                      'Set lines to size 3
XYJoin(1, 1);                           'Join consecutive points
ChanColourSet(1,1,1,1,0);               'Make lines yellow

func CalcCOI(i%, &freq)                 'Calc COI value and associated frequency
var scale := minScl*pow(2, dScl*i%);    'Generate scale
freq := ArrCWT(-6, scale, sInt);        'Generate frequency for this scale
return ArrCWT(-8, scale, sInt);         'Generate COI value
end;
```

The `CalcCOI(i%, &freq)` user-defined function works out the centre frequency and returns the COI value for a particular line of the output. The rest of the code draw the COI values forwards until it reaches a value that would be more than half-way through the data (to draw the left-hand side), then works backwards to draw the right-hand side.

ArrDiff()

This function replaces an array or result view with an array of differences. You can use this as a crude form of differentiation, however `ArrFilt()` provides a better method.

```
Proc ArrDiff(dest[]);
```

`dest[]` A real or integer vector that is replaced its differences. The first element of `dest` is not changed.

The effect of the `ArrDiff()` function can be undone by `ArrIntgl()`. The following block of code performs the same function as `ArrDiff(work[])`:

```
var work[100],i%;
for i%:=99 to 1 step -1 do work[i%] -= work[i%-1]; next;
```

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`, `ArrIntgl()`, `ArrMul()`, `ArrSub()`, `ArrSubR()`, `ArrSum()`, `Len()`

ArrDiv()

This function divides a real or integer array by an array or a constant. Use `ArrDivR()` to form the reciprocal of an array. Division by zero and integer overflow are detected.

```
Func ArrDiv(dest[]{[]...}, const source[]{[]...}|value);
```

`dest` An array of real or integer values. Assumed not to overlap destructively with `source` (if used).

`source` A real or integer array with the same number of dimensions as `dest`, used as the denominator of the division. If the arrays have different sizes, the smaller size is used for each dimension. This array is not modified.

`value` A value used as the denominator of the division.

Returns 0 or a negative error code if integer overflow or division by zero occurs.

If there was integer overflow when assigning the result to an integer array the result is set to the nearest allowed integer value. If division by zero occurs, the associated destination element is not changed.

The function performs the operations listed below. The indices `j` and `i` mean repeat the operation for all values of the indices. Both `a1d` and `b1d` are vectors, `a2d` and `b2d` are matrices. The arrays and value may be integer or real.

Function	Operation
`ArrDiv(a1d[], value);`	`a1d[`*i*`] := a1d[`*i*`] / value`
`ArrDiv(a1d[], b1d[]);`	`a1d[`*i*`] := a1d[`*i*`] / b1d[`*i*`]`
`ArrDiv(a2d[][], value);`	`a2d[`*j*`][`*i*`] := a2d[`*j*`][`*i*`] / value`
`ArrDiv(a2d[][], b2d[][]);`	`a2d[`*j*`][`*i*`] := a2d[`*j*`][`*i*`] / b2d[`*j*`][`*i*`]`

See also:
`ArrAdd(), ArrConst(), ArrDiff(), ArrDivR(), ArrDot(), ArrFFT(), ArrFilt(), ArrIntgl(), ArrMul(), ArrSub(), ArrSubR(), ArrSum(), Len()`

ArrDivR()

This function divides a real or integer array into an array or a constant.

```
Func ArrDivR(dest[]{[]...}, const source[]{[]...}|value);
```

`dest` An array of real or integer values used as the denominator of the division and for storage of the result. Assumed not to overlap destructively with `source` (if used).

`source` A real or integer array with the same number of dimensions as `dest` used as the numerator of the division. If the arrays have different sizes, the smaller size is used for each dimension. This array is not modified.

`value` A value used as the numerator of the division.

Returns 0 or a negative error code if integer overflow or division by zero occurs.

If there was integer overflow when assigning the result to an integer array the result is set to the nearest allowed integer value. If division by zero occurs, the associated destination element is not changed.

The function performs the operations listed below. The indices `j` and `i` mean repeat the operation for all values of the indices. Both `a1d` and `b1d` are vectors, `a2d` and `b2d` are matrices. The arrays and value may be integer or real.

Function	Operation
`ArrDivR(a1d[], value);`	`a1d[`*i*`] := value / a1d[`*i*`]`
`ArrDivR(a1d[], b1d[]);`	`a1d[`*i*`] := b1d[`*i*`] / a1d[`*i*`]`
`ArrDivR(a2d[][], value);`	`a2d[`*j*`][`*i*`] := value / a2d[`*j*`][`*i*`]`
`ArrDivR(a2d[][], b2d[][]);`	`a2d[`*j*`][`*i*`] := a2d[`*j*`][`*i*`] / b2d[`*j*`][`*i*`]`

See also:
ArrAdd(), ArrConst(), ArrDiff(), ArrDiv(), ArrDot(), ArrFFT(), ArrFilt(), ArrIntgl(), ArrMul(), ArrSub(), ArrSubR(), ArrSum(), Len()

ArrDot()

This function multiplies a vector by another and returns the sum of the products (sometimes called the dot product). The vectors are not changed.

```
Func ArrDot(const arr1[], const arr2[]);
```

arr1 A real or integer vector.

arr2 A real or integer vector.

Returns The function returns the sum of the products of the corresponding elements of the two vectors.

See also:
ArrAdd(), ArrConst(), ArrDiff(), ArrDiv(), ArrDivR(), ArrFFT(), ArrFilt(), ArrIntgl(), ArrMul(), ArrSub(), ArrSubR(), ArrSum(), Len()

ArrFFT()

This command performs spectral analysis on a result view, or on an array of data. The FFT uses transforms between a wholly real data array and a complex spectrum (see the Technical notes). Variants of this command produce log amplitude, linear amplitude, power and relative phase as well as an option to window the original data. The command has the syntax:

```
Func ArrFFT(dest[], mode%{, wnd%|mMag[]));
```

dest A real vector to process. It should be a power of two points long, from 8 points upwards; the upper size is limited by available memory. If the number of points is not a power of two, the size is reduced to the next lower power of two points.

mode% The mode of the command, in the range 0 to 5 or 10-11 for experimental ACSR algorithm. Modes are defined below.

wnd% Used only in mode 0 to set the window to use. 0 = none, 1 = Hanning, 2 = Hamming. Values 3 to 9 set Kaiser windows with -30 dB to -90 dB side band ripple in steps of 10 dB. If this is omitted a Hanning window is applied. This argument does not exist before version 8.03.

Returns The function returns 0 or a negative error code. From version 8.03 onwards it returns the window Power factor in mode 0.

Modes 1 and 3-5 take an array of data that is a set of equally spaced samples in some unit (usually time). If this unit is xin, the output is equally spaced in units of 1/xin. In the normal case of input equally spaced in seconds, the output is equally spaced in 1/seconds, or Hz. If there are n input points, and the interval between the input points is t, the spacing between the output points is 1/(n*t). The transform assumes that the sampled waveform is composed of sine and cosine waves of frequencies: 0, 1/(n*t), 2/(n*t), 3/(n*t) up to (n/2)/(n*t) or 1/(2*t).

Display of phase in result views

The phase information sits rather uncomfortably in a result view. When it is drawn, the x axis has the correct increment per bin, but starts at the wrong frequency. If you need to draw it, the simplest solution is to copy the phase information to bin 1 from bin n/2+1 and set bins 0 and n/2 to 0 (this destroys any amplitude information):

```
ArrFFT([], 5);           'Generate Power and Phase (or modes 3, 4)
ArrConst([1:],[n/2+1:]); 'Copy phase to the start of the view
[0]:=0; [n/2]:=0;        'Set phase of the DC and Nyquist points
Draw(0,n/2+1);           'Display the phase
```

Mode 0: Window data

```
Func ArrFFT(dest[], mode%{, wnd%});
```

This mode is used to apply a window to the data array. See `SetPower()` for an explanation of windows. The selected data is multiplied by a window of maximum amplitude 1.0, minimum amplitude 0.0. The return value is the power factor that is caused by using the window and then computing a power spectrum. If you do not supply `wnd%`, a raised cosine (Hanning) window is used and the return value is 0.375 (3/8). Prior to Spike2 version 8.03, mode 0 always applied a raised cosine window and the return value was 0.

Mode 1: Forward FFT

```
Func ArrFFT(dest[], 1);
```

This mode replaces the data with its forward Fast Fourier Transform. You would most likely use this to allow you to remove frequency components, then perform the inverse transform. The output of this mode is in two parts, representing the real and imaginary result of the transform (or the cosine and sine components). The first n/2+1 points of the result hold the amplitudes of the cosine components of the result. The remaining n/2-1 points hold the amplitudes of the sine components. In the case of an 8 point transform, the output has the format:

point	frequency	contents	point	frequency	contents
0	DC(0)	DC amplitude	4	4/(n*t)	Nyquist amplitude
1	1/(n*t)	cosine amplitude	5	1/(n*t)	sine amplitude
2	2/(n*t)	cosine amplitude	6	2/(n*t)	sine amplitude
3	3/(n*t)	cosine amplitude	7	3/(n*t)	sine amplitude

There is no sine amplitude at a frequency of `4/(n*t)`, the Nyquist frequency, as this sine wave would have amplitude 0 at all sampled points. Note that this FFT is optimised for use with real data. It is not a general purpose transform that takes a series of complex numbers as input and generates results with positive and negative frequencies. There are Technical notes that you can choose to install with Spike2 that cover how to generate a FFT from complex data values, should you require it.

Mode 2: Inverse FFT

```
Func ArrFFT(dest[], 2);
```

This mode takes data in the format produced by the forward transform and converts it back into a time series. In theory, the result of mode 1 followed by mode 2, or mode 2 followed by mode 1, would be the original data. However, each transform adds some noise due to rounding effects in the arithmetic, so the transforms do not invert exactly. One use of modes 1 and 2 is to filter data. For example, to remove high frequency noise use mode 1, set unwanted frequency bins to 0, and use mode 2 to reconstruct the data.

Mode 3: dB and phase

```
Func ArrFFT(dest[], 3);
```

This mode produces an output with the first `n/2+1` points holding the log amplitude of the power spectrum in dB, and the second `n/2-1` points holding the phase (in radians) of the data. In the case of our 8 point transform the output format would be:

point	frequency	contents	point	frequency	contents
0	DC(0)	log amplitude in dB	4	4/(n*t)	log amplitude in dB
1	1/(n*t)	log amplitude	5	1/(n*t)	phase in radians
2	2/(n*t)	log amplitude	6	2/(n*t)	phase in radians
3	3/(n*t)	log amplitude	7	3/(n*t)	phase in radians

There is no phase information for DC or for the point at `4/(n*t)`. This is because the phase for both of these points is zero. If you want the phase in degrees, multiply by 57.3968 (180º/pi). The log amplitude is calculated by taking the result of a forward FFT (same as mode 1 above) and forming:

dB = 10.0 Log(*power*) The *power* is calculated as for Mode 5

Mode 4: Amplitude and phase

```
Func ArrFFT(dest[], 4);
```

This mode produces the same output format as mode 3, but with amplitude in place of log amplitude. The amplitude is calculated by taking the result of a forward FFT (same as mode 1 above), and forming:

amplitude = `Sqrt`($\cos^2+\sin^2$) There is no sin component for the DC and Nyquist

Mode 5: Power and phase

```
Func ArrFFT(dest[], 5);
```

This mode produces the same output format as modes 2 and 3, but with the result as power. The sum of the power components is equal to the sum of the squares of the original data divided by the number of data points. The power is calculated by taking the result of a forward FFT (same as mode 1 above), and forming:

power = ($\cos^2$ + $\sin^2$) * 0.5 for all components except the DC and Nyquist
power = DC^2 or $Nyquist^2$ for the DC and Nyquist components

You can compare the output of this mode with the result of `SetPower()`. If you have a waveform channel on channel 1 in view 1, and do the following:

```
var spView%, afView%, pwr;         'assume in a time view
spView% := SetPower(1,1024);       'select power spectrum
Process(0,1024*View(-1).Binsize(1)); 'process first 1024 points
WindowVisible(1);                  'make window visible
afView% := SetAverage(1,1024,0);   'To copy 1024 points
Process(0,0);WindowVisible(1);     'show the data
View(afView%);                     'Move to the view holding data
pwr:=ArrFFT([], 0);ArrFFT([], 5); 'Apply window, take power spectrum
ArrDiv([:513], pwr);               'compensate for power loss due to the window
DrawMode(1,1);                     'display as a histogram
Draw(0,500);Optimise(0);           'Show 500 bins of Power
View(spView%);                     'Look at SetPower() result
Draw(0,500);Optimise(0);           'Show same bins of power spectrum
```

The results should be identical.

Mode 10: Get maximum amplitude

```
Func ArrFFT(arr[], 10, mMag[]);
```

The input array `arr` of `n` points (a power of 2) is transformed in the same manner as for `ArrFFT(dest[], 1)`. The `mMag` array, which must be of size at least `n/2+1` points, is used to store the maximum magnitude of any component. You must zero the `mMag` array before the first use (when it will be filled with the magnitudes of all components). Subsequent calls compare the new components in `arr` and update `mMag` if a larger component is located. This can be used for the training phase of the ACSR algorithm.

Mode 11: Filter data using ACSR algorithm

```
Func ArrFFT(arr[], 11, mMag[]);
```

Artifact Component Specific Rejection is an algorithm designed to reject signals with a known power spectrum and can be used in a situation where a desired signal is polluted by a constant (statistically stationary) background interference and the background can be established. See, *A Novel Technique to Reject Artifact Components for Surface EMG Signals Recorded During Walking with Transcutaneous Spinal Cord Stimulation: A Pilot Study*, Frontiers in Human Neuroscience, June 2021, Volume 15, Article 660583.

This filters the input array (`arr`) of n points (a power of 2) by applying a forwards FFT, then subtracting the `mMag` array of components (assuming they have the same phase as each transformed component in `arr`, setting negative amplitudes to zero and inverse transforming. This is equivalent to the following adjustment to each frequency component before the inverse transform:

```
func ACSR(&c, &s, mag)
var theta := ATan(s, c);
var m := Sqrt((c*c) + (s*s)) - magnitude;
if m < 0 then m := 0 endif;
c := m * cos(theta);
s := m * sin(theta);
end
```

where **c** and **s** are the cosine (real) and sine (imaginary) components and **mag** is the maximum magnitude to subtract.

The subtraction removes the maximum possible contribution of the background interference signal; you cannot claim that the result truly represents the signal with the constant interference removed. The result is suitable for qualitative analysis, for instance to assess EMG activity.

See also:
`ArrAdd(), ArrConst(), ArrDiff(), ArrDiv(), ArrDivR(), ArrDot(), ArrFFT(), ArrFilt(), ArrIntgl(), ArrMul(), ArrSub(), ArrSubR(), ArrSum(), Len(), SetPower()`

Technical Notes

Several users have been confused by our use of the FFT in the `ArrFFT()` command in both Spike2 and Signal. The confusion arises because of two factors:

1. We transform between a purely real time series and a complex frequency series
2. We show results only for positive frequencies in the forward and inverse transforms so that the result of a transform occupies the same space as the original data.

The reason we do this is that for many practical cases this is all that is required. Further, it makes the FFT run in half the time.

This section explains what we actually do, and how to obtain results for positive and negative frequencies if you require it. Results are quoted, not proved. If you need more information about the mathematics of this subject, see the references at the end.

Discrete Fourier transforms

The inverse discrete complex Fourier transform (inverse DFT) is a one to one mapping of any series of complex numbers $A(n)$, n=0,1,2,…N-1 onto another series defined by:

$$X(j) = \sum_{n=0}^{N-1} A(n) W_N^{nj}, j = 1,2,...N-1$$

where $W_N = e^{2\pi i/N}$ and i is the square root of -1. The inverse relationship that gives the $A(n)$ is terms of the $X(j)$ is the discrete Fourier transform (DFT):

$$A(n) = \frac{1}{N} \sum_{j=0}^{N-1} X(j) W_N^{-nj}, n = 1,2,...N-1$$

It is a matter of convention where the factor of $1/N$ goes. As shown above, the $A(n)$ are the amplitudes of the constituent sine and cosine waves. Some implementations place the $1/N$ in front of the inverse transform.

Circular nature of the transform

An important property of this pair of transforms is that the indices n and j are to be interpreted modulo N. That is

$X(j + kN) = X(j)$

for all integral values of k (positive or negative). W_n^{nj} expands to $e^{\frac{i2\pi nj}{N}}$ which is equivalent to:

$$\cos(\frac{2\pi nj}{N}) + i \sin(\frac{2\pi nj}{N})$$

Adding any multiple of N to j or k clearly makes no difference to the value.

Fast Fourier transforms

You will notice that the above discussion makes no statements about the value of N. In fact, it is true for any positive N. However, the calculation of the transform as written is a process of order $N2$ operations, which reduces its practical usefulness for large transforms. However, if the number N is highly composite (has many divisors), the process can be greatly speeded up. For the case of N a power of two, the process becomes of order $N\ log2(N)$ operations. This means that the time to process an array of size N increases at a rate of $log2(N)$ per point, not at N per point. For example, doubling the transform size from 1024 to 2048 points takes 1.1 times as long per point, not 4 times as long.

The Fast Fourier transform, or FFT, is the calculation of a discrete Fourier transform using a highly composite N. Many implementations of it (including ours) make N a power of two, but it need not be as long as N is the product of reasonably small numbers compared to the original N. All the following assumes that N is a power of 2.

Negative frequencies

The following discussion assumes that we are transforming data between the time domain *X(j)* and the frequency domain *A(n)*. The circular nature of the time and frequency indices *n* and *j* discussed above allow us to consider the frequency domain to run from either *0* to *N-1* or from *-N/2* to *N/2*, the latter being the more usual. You should notice that both the *0* and *N/2* frequency components are shared between the positive and negative frequencies (which is why there are only *N* components in a frequency range from *-N/2* to *N/2*, and not *N+1*).

In terms of sampled data, the frequency domain runs from minus half the sampling rate of the data to plus half the sampling rate. It seems strange that the DFT of a sinusoid produces the surprising result that it consists of one sinusoid at a positive frequency and one at a negative, each of half the amplitude you might expect. In fact, all the maths is saying is that both answers fit the data and it cannot choose between them.

Symmetry relationships

If the time series *X(j)* is wholly real, the *A(n)* must be conjugate even. Likewise, if *X(j)* is wholly imaginary, the *A(n)* must be conjugate odd. That is, if we use * to represent the complex conjugate, if *X(j)* is wholly real, *A(n) = A(-n)** and if *X(j)* is wholly imaginary, then *A(n) = -A(-n)**.

Application of the DFT to real world data

There are several problems to be resolved when you attempt to apply the FFT algorithm to real world data to convert a time series to the frequency domain. The first step is to sample the data. It is easy to show that if a sinusoid of frequency *N/2+f* is sampled, the sampled data is indistinguishable from a sinusoid of frequency *N/2-f*. This effect is called aliasing. To get unambiguous results, the input data must be band limited to the range *0* to *N/2* (or in fact to any band of this width anywhere in the frequency spectrum). The frequency *N/2* is usually referred to as the Nyquist critical frequency, or just the Nyquist frequency.

The next problem is that the maths assumes that the time series data is “circular”, that is that the frequency content is not changed in amplitude if the data is arbitrarily rotated. This assumption is, in general, not true, so when the discrete Fourier transform is used on real data the data is usually massaged in some way to make it circular. The usual method is to multiply the time series by some smooth function that is small at the ends and unity in the middle. The function is called a “window”, and the process is called “windowing”.

The good effect of a window is to reduce the spurious components caused by the discontinuity between the first and last sample. The bad effect is that the window causes each data point to be “smeared” in the result, and it significantly reduces the contribution of points near the ends of the sampled data to the result. The topic of windowing is complex and is beyond the scope of this discussion.

Finally, the results are for a discrete set of equally spaced frequencies. However, the input data will, in general not consist of these discrete frequencies. A sinusoid of an intermediate frequency will be represented in both adjacent discrete frequencies.

The transform used in Spike2

Notice that the FFT transforms complex time series to complex frequency series. However, in the real world, sampled time series data does not normally include imaginary components. We usually want to transform wholly real time series to complex frequency series. There are three ways round this:

1. Set the imaginary components to 0 and calculate the FFT. This has the advantage of simplicity, but does twice as much work as is actually required.
2. If you have two series to transform, treat one as real and the other as imaginary, do the transform, then use the symmetry properties of the DFT to separate out the data. This is fine if you have two arrays, but this is not always the case. This method also suffers from cross-spectral leakage between the two signals due to truncation and rounding inaccuracies in the calculations.
3. Split the data into two arrays and transform them together, then amalgamate the results and take advantage of the symmetry so that the results can be packed into the same space as the original.

We implement the third option, which seems complicated, but is actually no harder to program than the FFT itself. However, if you need to generate results including negative frequencies you have a small amount of post-processing work to do. As we do not have a complex data type available we store the real and imaginary components separately.

Forward FFT
ArrFFT(x[], 1)

The forward transform takes *N* real data points and transforms them into *N* positive frequency complex components. The first component *A(0)* is the mean amplitude of the original data points. The next *N/2-1* components are the amplitudes of the real (cosine) components of data, the next component *A(N/2)* is the Nyquist frequency amplitude, and the remaining *N/2-1* components are the imaginary (sine) components.

If you want the results expressed in the same format as used by the DFT with negative frequencies, you will need more space to store the result. The example below assumes that we will transform 8 real data points:

DFT for 8 real data points

To transform 8 real data points, the DFT would require 8 real data points and 8 zeros for the imaginary components. In the transform, the real data (Rn) starts with the DC component, then three frequency components for positive frequencies, then the Nyquist component, followed by three components for negative frequencies. However, the negative frequency components are the same as the positive frequency components.

The imaginary result corresponding to DC is 0 (as all the imaginary data is zero). It is not quite so obvious why the imaginary Nyquist component is 0, but if you imagine a sine wave at the Nyquist frequency it would be zero at each sample. The imaginary frequency components (In) show odd symmetry around the DC and Nyquist frequency.

Real								Imaginary (all zeros)							
X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	0	0	0	0	0	0	0	0

Transforms to

R_0	R_1	R_2	R_3	R_N	R_3	R_2	R_1	0	I_1	I_2	I_3	0	$-I_3$	$-I_2$	$-I_1$

ArrFFT(x[],1) equivalent

To do the same job with `ArrFFT()` you need supply only the 8 real data points. The real and imaginary results can also be fitted into 8 bins. Notice that the amplitudes of the components other than the DC and Nyquist are doubled compared to the DFT result above. You could argue that this is incorrect, but for a real problem, most users consider a sine wave at a frequency to be one positive frequency, not two sine waves of half the amplitude each, one at the positive frequency and one at the negative frequency.

X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7

Transforms to

R_0	$2R_1$	$2R_2$	$2R_3$	R_N	$2I_1$	$2I_2$	$2I_3$

If you really need the full result, it is a simple matter to expand the data into the full format. The Rn and In are the same as for the DFT above.

Inverse FFT ArrFFT(x[],2)

The inverse transform makes the assumption that the result will be purely real. To achieve this, the negative frequencies needed for the DFT are derived from the positive frequencies. The operation is the exact inverse of that described above.

A very common use of the inverse transform is to filter data by forward transforming, removing unwanted components, then inverse transforming. Another common use is to produce random signals with a known power spectrum by assigning known amplitudes and randomised phases and inverse transforming. In both these cases, the assumption that the output is wholly real is warranted.

Power and phase ArrFFT(x[],5)

The phase is calculated from the real and imaginary components with the phase of the DC and Nyquist components always 0, so they are omitted. The power is calculated so that (in terms of the DFT):

$$\frac{1}{N}\sum_{j=0}^{N-1}|X(j)|^2 = \sum_{n=0}^{N-1}|A(n)|^2$$

The `ArrFFT()` implementation starts with an array of real data and replaces it with an array of power components. The power components combine the negative and positive frequencies. In the diagram, P is power and θ is phase. The sum of the five powers in the result is equal to the sum of the squares of the Xj divided by 8.

X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7

Transforms to

P_0	P_1	P_2	P_3	P_N	θ_1	θ_2	θ_3

In terms of the result of the forward FFT described above:

P0 = R02, P1 = R12+ I12, P2 = R22+ I22, P3 = R32+ I32, PN = RN2

If you need to recover power for the positive and negative frequencies separately, the power at frequencies -3, -2 and -1 is $P_3/2$, $P_2/2$ and $P_1/2$ and the power for the positive frequencies is also halved. The phase given is correct for positive frequencies, however it must be changed to $2\pi-\theta n$ for negative frequencies.

Summary

The `ArrFFT()` command presents results in a format that is convenient and easy to use for most practical uses of the FFT with real world data. However, if you need results that show both positive and negative frequencies, these are also obtainable by manipulation of the results. Further, although not discussed here, you can generate a full complex to complex transform by using the `ArrFFT()` command twice.

References

You can find further information on the mathematics of the DFT and FFT in these references. The first is the famous paper that rediscovered the FFT. The second contains the algorithms for transforms of real only data. The third is a well-known book that covers the whole topic (and many others) in great depth.

1. Cooley, J. W., and Tukey, J. W., "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297-301, April 1965.
2. Cooley, J. W., Lewis, P. A. W., and Welch, P.D., "Programming considerations in the calculation of sine, cosine and Laplace transforms," *J. Sound Vib.*, vol 12, pp. 315-337, July 1970.
3. Rabiner, L. R., and Gold, B., 1975, *Theory and Application of Digital Signal Processing*, Pub. Prentice-Hall.

ArrFilt()

This function applies a FIR (Finite Impulse Response) filter to a real array. You can use `FiltCalc()` and `FIRMake()` to generate filter coefficients for a wide range of filters.

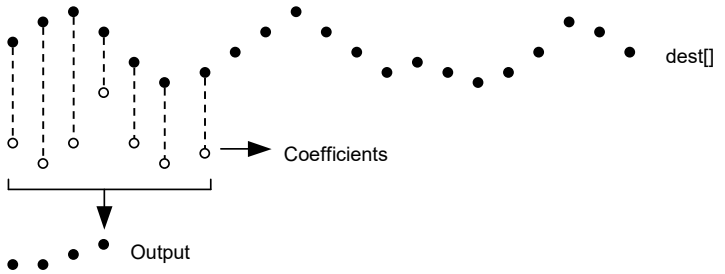
```
Func ArrFilt(dest[], const coef[]);
```

`dest[]` A real or integer vector holding the data to filter. It is replaced by the filtered data. There must be at least as many data points a coefficients.

`coef[]` A real vector of filter coefficients. This is usually an odd number of data points long so that the result is not phase shifted. If you use an even number, the result is delayed by 1/2 a sample. Prior to [7.08],

the result was advanced by 1/2 a sample. An even number of coefficients is used with a full differentiator. There must be at least 2 coefficients.

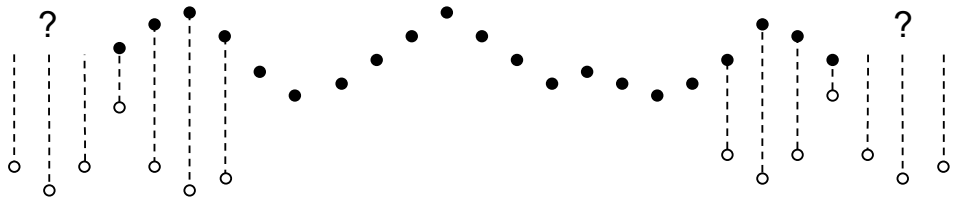
Returns The function returns 0 if there was no error, or a negative error code.



This diagram shows how the FIR filter works. Open circles represent filter coefficients, solid circles are the input and output waveforms. Each output point is generated by multiplying the waveform by the coefficients and summing the result. The coefficients are then moved one step to the right and the process repeats.

From this description, you can see that the filter coefficients (from right to left) are the *impulse response* of the filter. The impulse response is the output of a filter when the input signal is all zero except for one sample of unit amplitude. In the example above with seven coefficients, there is no time shift in the output. If the filter has an even number of coefficients, there is an output time shift of half a sample.

The filter operation is applied to every vector element. There is a problem at the start and end of the vector where some coefficients have no corresponding data element.



The simple solution is to take these missing points as copies of the first and last points. This is usually better that taking these points as 0. You should remember that the first and last (*nc*+1)/2 points are unreliable, where *nc* is the number of coefficients.

A simple use of this command is to produce three point smoothing of data, replacing each point by the mean of itself and the two points on either side:

```
var data[1000],coef[3];      'Arrays of data and the coefficients
...                          'Fill data[] with values
coef[0]:=0.33333; coef[1]:=0.33333; coef[2]:=0.33333;
ArrFilt(data[],coef[]);      'smooth the data.
```

If you use this command to apply a causal filter, that is, one with all coefficients that use data points ahead of the current point set to zero, you must still provide these coefficients. If you omit the trailing zero coefficients, the output will be time shifted backwards by half the number of coefficients you do supply.

See also:

`ArrAdd()`, `ArrConst()`, `ArrConv()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrIntgl()`, `ArrMul()`, `ArrSub()`, `ArrSubR()`, `ArrSum()`, `FiltCalc()`, `FIRMake()`, `Len()`

ArrHist()

This function generates a histogram from a data array. This function is designed for repeated use with the same destination histogram and different source data; it is up to the caller to zero the histogram array before the first use. You can use `Min()`, `Max()`, `ArrSum()` and `ArrStats()` to find suitable values for the `start` and `binSz` parameters.

```
Func ArrHist(hist%[], const data[], start, binSz{, &left%{, &right%}});
```

`hist%` An integer array to hold the generated histogram. The elements of this array are incremented by 1 for each item in the data array that falls in one of the histogram bins. The array is not set to zero before

incrementing the bins. The bin to increment for data element `i%` is given by: `(data[i%]-start)/binSz`.

`data` An array of real or integer (from version [10.00]) data values to add into the histogram.

`start` The value at the start of the first bin,

`binSz` The width of each bin. This can be negative but must not be 0. With an integer data array this value is usually integral.

`left%` Optional. This integer variable is incremented by the number of data items with a negative bin number (falling to the left of the histogram).

`right%` Optional. This integer variable is incremented by the number of data items with a bin number greater than or equal to `Len(hist%[])` (falling to the right of the histogram).

Returns The number of data items that fell inside the histogram.

See also:
`Min(), Max(), ArrSum(), ArrStats()`

ArrIntgl()

This function is the inverse of `ArrDiff()`, replacing each point by the sum of the points from the start of the array up to the element. The first element is unchanged.

```
Proc ArrIntgl(dest[]);
```

`dest` A vector of real or integer data.

The function is equivalent to the following:

```
for i%:=1 to Len(dest[])-1 do dest[i%] += dest[i%-1]; next;
```

See also:
`ArrAdd(), ArrConst(), ArrDiff(), ArrDiv(), ArrDivR(), ArrDot(), ArrFFT(), ArrFilt(), ArrMul(), ArrSub(), ArrSubR(), ArrSum(), Len()`

ArrMapImage()

This command maps a matrix into a bitmap of the same size as the matrix and writes the bitmap to the clipboard or to an image file on disk. This command is commonly used to convert the output of the `ArrCWT()` command into an image. It can also be used to convert the result of any process that generates a 2D map into an image. The bitmap will be created as 32-bits per pixel unless this causes a memory problem in which case an 8 bit per pixel map with a maximum of 256 colours is generated. This script command was added at Spike2 version [9.02].

You can create bitmaps of any width with this command (within the memory constraints of the system). However, Windows does not expect bitmaps wider or taller than 65535 pixels and many internal functions will fail if the bitmap is larger than this (some fail with widths more than 32767 pixels). As it is not unusual for the output of the `ArrCWT()` command to exceed this width, we provide mechanisms to restrict the width and height of the output. As an option, we allow you to generate larger maps, but these may fail with other programs. See the `ArrCWT()` example for code adding an image to an XY view.

```
Func ArrMapImage(arr[][], path$){, flags%{, map%{, vMin{, vMax}}}});
```

`arr` A matrix holding data to be mapped into a bitmap. If this array is `arr[nx%][ny%]`, the bitmap will have `nx%` pixels in the horizontal direction and `ny%` pixels in the vertical direction. Unless the `flags%` argument is used to invert the vertical axis, `arr[][0]` sets the top row of pixels of the bitmap and `arr[][ny%-1]` sets the bottom row.

`path$` This is either the file name on disk to write to or `"<CB>"` to place the resulting image on the clipboard as a bitmap. If this is a file name, the file extension determines the type of the image saved to disk if it is one of: `.bmp`, `.jpg`, `.jpeg`, `.gif`, `.png`, `.tif` or `.tiff`. If it is none of these, the file is saved in bitmap format. If the disk file already exists, it will be replaced.

`flags%` Additional controls to define how we perform the colour mapping. If omitted, this argument is 0. Bits 0-3 (values 0-7) are used to determine how we manage the situation when the bitmap would be wider than 65535 pixels. In this case, we limit the width to the maximum possible size and then map the source values to the pixels. When you use large bitmaps as a channel background (see the `ChanImage()` command), Spike2 takes special steps to render the image; other programs may not be so forgiving.

Value	Meaning
0	Sub-sample the source array in the x direction to get the value to map to a pixel.
1	Take the mean value of the range of source values that map to each pixel.
2	Take the maximum value of the range of source values that map to each pixel.
3	Take the minimum value of the range of source values that map to each pixel.
4-6	Reserved, treated as 0.
7	Do not limit the width. This can generate an output image that may not be readable by some programs.

You can also add additional flag values. Currently we define only one:

Value	Meaning
8	Invert the bitmap so that `arr[][0]` generates the bottom row of pixels and not the top row.

`map%` This is the colour scale number as defined in the `ColourGet()` command. If this is not supplied, the default value is 1 (the Rainbow map).

`vMin` The values `vMin` to `vMax` define the range of values to map onto the range of the colour map. `vMin` sets the data value in `arr[][]` to map onto the low end of the colour map. If you do not provide these values, the range of data values found in `arr[][]` is used; `vMin` will then be the lowest value found. You can use `ArrRange()` to find the data range.

`vMax` This sets the value to map to the high end of the colour map. If omitted, the highest value in `arr[][]` is used. There is no requirement for `vMax` to be greater than `vMin`. If `vMax` and `vMin` are the same, the display will be entirely set to the lowest colour in the map. Data values outside the range `vMin` to `vMax` are limited to the nearer end of the range.

Return 0 for success or a negative error code.

See also:
`ArrRange(),ArrCWT(),ColourGet(), EditPaste(), EditCopy(), EditImageSave()`

ArrMedian()

This command (added at [10.13]) finds the median of an array. If an array has an odd number of elements, the median is the element in the middle of the list if the array were sorted into order. If the array has an even number of elements, the median is the mean of the centre two elements if the array were sorted into order. The median can be useful as a robust estimator of the mean for use in situations where occasional outliers (or infinities) make the use of the mean inappropriate. The command is:

```
Func ArrMedian(const data[]);
```

data An integer or real array.

Returns The median value of the array. Note that if the array is of integers, the median value is returned as a double, so the accuracy is limited to the resolution of a double. If the median has a magnitude greater than around `4.5e15` it is in danger of losing accuracy (an integer can have a magnitude of `1.8e19`).

See also:
`ArrSum(), ArrStats()`

ArrMul()

This command is used to form the product of a pair of arrays, or to scale an array by a constant. A less obvious use is to negate an array by multiplying by -1.

```
Func ArrMul(dest[]{[]...}, const source[]{[]...}|value);
```

`dest` An array of real or integer numbers. If `dest` is integer and source is `real`, a real multiplication is done and truncated to integer. If both are integer an integer multiplication is done. If a result exceeds integer range, the result is limited to the maximum or minimum value possible. Assumed not to overlap destructively with `source` (if used).

`source` A real or integer array with the same number of dimensions as `dest`. If the arrays have different sizes, the smaller size is used for each dimension.

`value` A real value to multiply the data in `dest`.

Returns The function returns 0 if all was well, or a negative error code. If the destination array is integer and the result of any multiplication overflows integer range the result is truncated to integer range and an overflow error is reported.

The function performs the operations listed below. The indices *i* through m mean repeat the operation for all values of the indices. Both `a1d` and `b1d` are vectors, `a2d` and `b2d` are matrices and so on. The arrays and value may be integer or real.

Function	Operation
`ArrMul(a1d[], value);`	`a1d[i] := a1d[i] * value`
`ArrMul(a1d[], b1d[]);`	`a1d[i] := a1d[i] * b1d[i]`
`ArrMul(a2d[][], value);`	`a2d[j][i] := a2d[j][i] * value`
`...`	`...`
`ArrMul(a5d[][][][][], b5d[][][][][]);`	`a5d[m][l][k][j][i] := a5d[m][l][k][j][i] * b5d[m][l][k][j][i]`

See also:
`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`, `ArrIntgl()`, `ArrSub()`, `ArrSubR()`, `ArrSum()`, `Len()`, `MATMul()`

ArrRange()

This command will scan an array of any number of dimensions and return the range of the data. This command was added at Spike2 version [9.02]. There are two variants, one for Real values and one for integer:

Real data

```
Proc ArrRange(const arr[]..., &vMin, &vMax);
```

`arr` A real array with any number of dimensions to be searched for the minimum and maximum value.

`vMin` A real variable that is returned holding the minimum value found in the array.

`vMax` A real variable that is returned holding the maximum value found in the array.

Integer data

```
Proc ArrRange(const arr%[]..., &vMin%, &vMax%);
```

`arr%` An integer array with any number of dimensions to be searched for the minimum and maximum value.

`vMin%` An integer variable that is returned holding the minimum value found in the array.

`vMax%` An integer variable that is returned holding the maximum value found in the array.

See also:
`Max()`, `Min()`, `MinMax()`

ArrRev()

This command reverses the order of data in the first dimension of an array. If you want to reverse the characters in a string, use `Reverse$()`.

```
Proc ArrRev(arr[]...)
```

`arr` An array of any type (real, integer or string) with any number of dimensions.

This is easy to understand when the array has a single dimension, when the data in that dimension is reversed. Arrays with `n` dimensions (`n` > 1) are treated as an array of `n`-1 dimensions of a 1 dimensional array, which is reversed. Put another way, the first dimension is reversed. Two identical `ArrRev()` commands restore the array to its original order. For example a matrix defined as:

```
var m[3][2] := {{0, 1, 2}, {3, 4, 5}};
```

after `ArrRev(m)` would become:

```
{{2, 1, 0}, {5, 4, 3}}
```

You can, of course, reverse part of an array:

```
var days$[5] :={"Monday", "Thursday", "Wednesday", "Tuesday", "Friday");
ArrRev(days$[1:3]);
```

Would put your working week back in order.

See also:
`Reverse$()`

ArrSort()

This function sorts (or from [10.17], shuffles) an array of any data type and optionally orders additional arrays to match the sorted array. When multiple arrays exist, all arrays are treated as having the same length, which is the shortest length of any array in the list.

```
Func ArrSort(sort[]{, opt%{, arr1[]{, arr2[]{, ...}}});
```

`sort[]` An array of integer, real or string data to sort. From [10.12] this can be a zero length array. From [11.00], if `opt%` is -1 to randomize the order this can be an array of Objects.

`opt%` This optional argument holds the sorting options. If omitted, the value 0 is used meaning ascending order and case insensitive. If positive, it is the sum of the following flag values:

- 1 Sort in descending order. If omitted, the data is sorted in ascending order.
- 2 Case sensitive sort when sort[] is an array of strings. String sorts are usually case insensitive. If omitted, the sort is case insensitive.

If set to -1, the array is not sorted but shuffled (re-arranged into a random order).

`arrn[]` If present, these arrays are sorted in the same order as the `sort[]` array. The arrays can be of any type. You can sort up to 18 additional arrays. These arrays should not overlap with themselves or with the `sort` array.

Returns The number of sorted elements (the least number of elements in `sort` and `arr1`...`arrn`).

See also:
`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrMul()`, `ArrSubR()`, `ArrSum()`, `Len()`

ArrSpline()

This function interpolates an array of real or integer data sampled at one rate into another array sampled at a different rate using cubic splines. It can also interpolate a matrix of (x,y) value pairs to an array sampled at a constant rate. This assumes that the source data has continuous first and second derivatives and that the second derivatives vary linearly from point to point. The second derivatives at the first and last point of the source data

are set to zero. We interpolate from up to one input sampling interval before the first source point to up to one sampling interval after the last source point.

```
Func ArrSpline(dest[], const source[]{, xInc{, xStart}});
Func ArrSpline(dest[], const srcXY[][]{, xInc{, xStart}});
```

`dest` A real or integer vector that holds the interpolated result. If the source array is a matrix, this is a real array.

`source` A vector of y values. The associated x values are assumed to be the same as the index of each data point (0, 1, 2...). It is integer array if `dest` is integer or real if `dest` is real.

`srcXY` A matrix of (x,y) source points. `srcXY[][0]` holds x values and `srcXY[][1]` holds y values. The x values must increase with the index, otherwise the return value is -1.

`xInc` If present, this is the x value change between `dest` values. If omitted, we assume that the points in `source` or `scrXY` and `dest` span the same time range. There are no restrictions on `xInc`, it can even be negative (in which case the output is backwards relative to the input). If the `source` variant is in use, `xInc` is the ratio of the `dest` sample interval to the `source` sample interval.

`xStart` If present, this is the x position of the first point of `dest`. If omitted, the first points of both arrays are at the same x position.

Returns 0 if all was OK or -1 if the `scrXY` variant is used and the x values do not increase monotonically.

You will get the best results if you can supply source data before and after the output time range. The effect of a source point on the interpolation of an interval n points away falls by a factor of approximately 4 each time n increases by 1. There is rarely the need to supply more than 15 data points before and after the interpolation range.

An example

Suppose we have a source vector of 100 points sampled at 100 Hz, with the first point sampled at 5.02 seconds, and we want to generate an array sampled at 1000 Hz that starts at 5.335 and lasts 0.5 seconds. In this case, the value of `xInc` is 0.001/0.01, which is 0.1. The value of start is 31.5, which is the time difference (5.335 - 5.02) divided by 0.01, the sample interval of the source channel.

See also:

`ChanProcessAdd(), DrawMode()`

ArrStats()

This calculates the mean, variance, skewness and (excess) kurtosis of an array of real data. It can also be used with a matrix to calculate vectors of the mean, variance, skewness and kurtosis of each column of the input matrix. Use the `ArrSum()` command if you only require the mean and standard deviation. Note that outliers or infinities can make the results less useful; the `ArrMedian()` command can give a more robust mean estimate.

```
Func ArrStats(const data[], &mean, &v, &skew, &kur{, mode%});
```

`data` A vector of source data.

`mean` A real variable returned holding the mean of the data.

`v` A real variable returned holding the variance of the data.

`skew` A real variable returned holding the skewness of the data.

`kur` A real variable returned holding the kurtosis of the data.

`mode%` Optional, default 0. Determines the algorithm used to sum the values to form the mean. 0 for naive sum, 1 for block sum, 2 for Kahan sum. This was added at [10.12].

Returns 0 if all was well or 1 if the variance of the data was 0.0 (in which case the skewness and kurtosis are also set to 0.0).

The second form of the command is provided as a convenience when the data is in a matrix with the data in the columns. When used in the second form, the vector must be at least the size of the first dimension of the matrix.

```
Func ArrStats(const data[][], mean[], v[], skew[], kur[]{,mode%});
```

`data` A matrix with the data in the columns.

`mean` A real vector returned holding the means of each column of data.

`v` A vector returned holding the variances of each column of data.

`skew` A vector returned holding the skewness of each column of data.

`kur` A vector returned holding the kurtosis of each column of data.

`mode%` Optional, default 0. Determines the algorithm used to sum the values to form the mean. 0 for naive sum, 1 for block sum, 2 for Kahan sum. This was added at [10.12].

Returns The number of columns for which the variance was 0 (in which case the skewness and kurtosis are also 0).

Summing algorithm

From Spike2 version [10.12] there are three algorithms available for calculation of the mean. In normal use when working on arrays of sampled data the default method (`mode%` set to 0), is fine. However, if your data has a huge dynamic range, you may find that a simple sum of the values does not give you the result you expect. This is due to the fact that the resolution of floating point numbers is proportional to their magnitude. Put another way, a 64-bit IEEE Real number has around 16 decimal digits of precision. If you add two floating point numbers, the precision of the result is, at best, the precision of the number with the larger magnitude. When you add a lot of numbers together you can end up in the situation where some numbers get lost in the noise. If this becomes an issue you can use the `mode%` argument to get a better result.

mode% 0: Naive sum

This is the standard method and is the fastest. It offers no protection against data with a huge range between the smallest and largest values. However, it is fine for data that is of a similar magnitude (for example sampled waveforms).

mode% 1: Block sum (pairwise summation)

This method sums blocks of data recursively, the aim being that if the data points are similar, you are always adding numbers of similar size, so every number makes a similar contribution to the sum. This runs a little slower than `mode%` 0. If offers some protection against data with a large dynamic range. For those interested in such things, we use a block size of 128 points.

mode% 2: Kahan sum

This method tracks the error in the sum due to floating point precision issues and in most cases it gives the best result you can expect. There are still pathological cases with huge numbers that can cause bad results, but for realistic data this gives a very accurate result. It is several times slower than the other methods, and for most data sets the result is little different from mode 1.

How the variance, skew and kurtosis values are calculated

The first form of the command is equivalent to the following script (but runs many times faster):

```
Func ArrStats(arr[], &mean, &dVar, &skew, &kur, mode%)
var i%, n%;
mean := 0; dVar := 0; skew := 0; kur := 0;
ArrSum(arr, mean, mode%);  'Calculate the mean
n% := Len(arr);
var d, p;
for i% := 0 to n%-1 do
    d := arr[i%]-mean; 'difference from mean
    p := d*d;          'Square of difference
    dVar += p;         'Sum of squares
    p *= d;            'Cube of difference
    skew += p;         'Sum of cubes
    kur += p*d;        'Sum of fourth powers
next;
if (dVar > 0.0) then   'If not a degenerate case...
    dVar /= (n%-1);    'calculate the variance
    skew /= n%*dVar*sqrt(dVar); 'Skew
```

```
    kur /= n%*dVar*dVar;
    kur -= 3.0;         'and kurtosis
    return 0;
else
    return 1;
endif;
end;
```

The second form of the command is equivalent to:

```
var i%, return% := 0;
for i% := 0 to Len(data[0][])-1 do
    return% += ArrStats(data[][i%], mean[i%], v[i%], skew[i%], kur[i%], mode%);
next;
```

See also:
`ArrSum(), ArrMedian()`

ArrSub()

This function forms the difference of two arrays or subtracts a constant from an array. If the destination is an integer array, overflow is detected when subtracting real values.

```
Func ArrSub(dest[]{[]...}, const source[]{[]...}|value);
```

`dest` A real or integer array that holds the result. Assumed not to overlap destructively with `source` (if used).

`source` A real or integer array with the same number of dimensions as `dest`. If the arrays have different sizes, the smaller size is used for each dimension. This array is not changed.

`value` A real or integer value.

Returns 0 if all is well or a negative error code if integer overflow is detected.

The function performs the operations listed below. The indices *j* and *i* mean repeat the operation for all values of the indices. Both `a1d` and `b1d` are vectors; `a2d` and `b2d` are matrices. The arrays and value may be integer or real.

Function	Operation
`ArrSub(a1d[], value);`	`a1d[`*i*`] := a1d[`*i*`] - value`
`ArrSub(a1d[], b1d[]);`	`a1d[`*i*`] := a1d[`*i*`] - b1d[`*i*`]`
`ArrSub(a2d[][], value);`	`a2d[`*j*`][`*i*`] := a2d[`*j*`][`*i*`] - value`
`ArrSub(a2d[][], b2d[][]);`	`a2d[`*j*`][`*i*`] := a2d[`*j*`][`*i*`] - b2d[`*j*`][`*i*`]`

See also:
`ArrAdd(), ArrConst(), ArrDiff(), ArrDiv(), ArrDivR(), ArrDot(), ArrFFT(), ArrFilt(), ArrIntgl(), ArrMul(), ArrSubR(), ArrSum(), Len()`

ArrSubR()

This function forms the difference of two arrays or subtracts an array from a constant. If the destination is an integer array, overflow is detected when subtracting real values.

```
Func ArrSubR(dest[]{[]...}, const source[]{[]...}|value);
```

`dest` A real or integer array. Assumed not to overlap destructively with `source` (if used).

`source` A real or integer array with the same number of dimensions as `dest`. If the arrays have different sizes, the smaller size is used for each dimension. This array is not changed.

`value` A real or integer value.

Returns 0 if all is well or a negative error code if integer overflow is detected.

The function performs the operations listed below. The indices *j* and *i* mean repeat the operation for all values of the indices. Both `a1d` and `b1d` are vectors; `a2d` and `b2d` are matrices. The arrays and value may be integer or real.

Function	Operation
`ArrSubR(a1d[], value);`	`a1d[i] := value - a1d[i]`
`ArrSubR(a1d[], b1d[]);`	`a1d[i] := b1d[i] - a1d[i]`
`ArrSubR(a2d[][], value);`	`a2d[j][i] := value - a2d[j][i]`
`ArrSubR(a2d[][], b2d[][]);`	`a2d[j][i] := b2d[j][i] - a2d[j][i]`

See also:
`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`, `ArrIntgl()`, `ArrMul()`, `ArrSub()`, `ArrSum()`, `Len()`

ArrSum()

This function forms the sum of the values in a vector or matrix, and optionally forms the mean and standard deviation of the vector or of each matrix column. The standard deviation of `m` data points with a sum of squared difference from the mean of `errSq` is `sqrt(errSq/(m-1))`. There are three command variants:

Process data

```
Func ArrSum(const arr[]|arr%[]{, &mean{, &stDev}});
Func ArrSum(const arr[][]|arr%[][]{, mean[], stDev[]});
```

`arr` A real or integer vector or matrix to process.

`mean` If present it is returned holding the mean of the values in the array. The mean is the sum of the values divided by the number of array elements. If `arr` is an `m` by `n` matrix, `mean` must be a vector of at least `n` elements and is returned holding the mean of each column of `arr`.

`stDev` If present, this returns the standard deviation of the array elements around the mean. If the array has only one element the result is 0. If `arr` is a `m` by `n` matrix, `stDev` must be a vector with at least `n` elements and is returned holding the standard deviation of each column of `arr`.

Returns The function returns the sum of all the array elements as a real number.

Set summation mode (new at [10.12])
There are more accurate (and slower) ways to sum data values when accuracy is important. By default, this routine uses a simple-minded and fast method. Any change made here persists until the script stops running.

```
Func ArrSum(mode%);
```

`mode%` Set -1 for no change or 0 for Naive sum (the default mode), 1 for Block sum or 2 for Kahan sum of the values. See `ArrStats()` for more information.

Returns The summation method in effect before any change made by this call.

See also:
`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`, `ArrIntgl()`, `ArrMul()`, `ArrStats()`, `ArrSub()`, `ArrSubR()`, `Len()`

Asc()

This function returns the code point (integer value) of the first character in the string as an integer. If the string holds only ASCII text, the returned value is the ASCII code of the first character.

```
Func Asc(text$);
```

`text$` The string to process.

Returns The integer code of the first character of the string. This can be any Unicode character in the range `1` to `0xd7ff` or `0xe000` to `0x10ffff`. An empty string will return 0. In non-Unicode versions of Spike2 before version 8.03, the returned value was limited to the range 0 to 255. Strings are held in a vector of

16-bit values. Most common characters are encoded by a single 16-bit value, but some require two characters. If the value we return is in the range 1 to 0xd777 or 0xe000 to 0xffff, the character is encoded in a single 16-bit value. All other values are encoded in two 16-bit values. If you want to print the code points of all the characters of a string, use code%:=Asc(text$);DelStr$(text$, 1, 1); to iterate through the string until the string is empty.

See also:
Chr$(), DelStr$(), LCase$(), Left$(), Len(), Mid$(), Print$(), Right$(), Replace$(), Reverse$(), Str$(), UCase$(), Val()

ATan()

This function returns the arc tangent of an expression, or the arc tangent of an array:

```
Func ATan(s|s[]{[]...} {,c|const c[]{[]...}});
```

s A value or an array of real values with any number of dimensions.

c Optional. If present, both s and c must be values, or both must be real arrays with the same number of dimensions. If c is present, s/c is a tangent and the result is in the range -π to π using the signs of s and c to determine the quadrant. If c is omitted, the result is in the range -π/2 to π/2, equivalent to c being unity.

Returns If s is an array, each element of s (that is matched by an element of c if c is present) is replaced by the arc tangent. The function returns 0 if all was well or a negative error code. When s is not an array, the return value is the arc tangent of s or of s/c if c is present.

You can use ATan() to calculate other mathematical values. Arc sine of a single value: s can be calculated as: ATan(s/Sqrt(1-s*s)). Arc cosine can be calculated as: ATan(Sqrt(1-s*s)/s).

Examples

```
var s[100], a[20][200], b, c[100], d[20][20];
ATan(s, c);            'both arguments are vectors
ATan(a);               'single argument is multi-dimensioned array
ATan(b);               'single argument is a variable
ATan(Sin(1), Cos(1));  'result should be 1
ATan(a, d);            'Changes a[0:20][0:20] only (to match d)

Func ASin(s)           'Arc Sine function
return ATan(s, Sqrt(1-s*s));
end

Func ACos(s)            'Arc Cosine function
return ATan(Sqrt(1-s*s), s);
end
```

See also:
Abs(), Cos(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan(), Trunc()

B

BetaI()
Binomial Distribution
Student's T Distribution
F-Distribution
BinError()
BinomialC
BinSize()
BinToX()
BRead()
BReadSize()
BRWEndian()
BSeek()

BurstMake()
BurstRevise()
BurstStats()
BWrite()
BWriteSize()
Using BWriteSize() to save a bitmap

Betal()

This function computes both the Beta function and the Incomplete Beta function. The Beta function is defined as:

$$Beta(a,b) = \int_0^1 t^{a-1}(1-t)^{b-1}dt$$

The incomplete beta function is defined as:

$$BetaI_x(a,b) = \frac{\int_0^x t^{a-1}(1-t)^{b-1}dt}{Beta(a,b)}$$

These functions are not usually directly of interest, but they are used in generating distribution functions, for example Student's distribution, the F-Distribution and the Binomial distribution.

```
Func BetaI(a, b{, x});
```

a, b These values must be greater than zero.

x Optional. If present, the result is the incomplete beta function. If omitted, the result is the beta function. x must be in the range 0 to 1.

Returns If x is present, the result is the incomplete beta function in the range 0 to 1. Otherwise, the result is the beta function.

See also:
GammaP(), GammaQ(), Binomial Distribution, Student's T Distribution, F-Distribution

Binomial Distribution

The Binomial distribution is used when you have repeated trials and for each trial an event has a known probability *p* of occurrence. The examples below show you how to calculate the probabilities that you will get k % or fewer events in n% trials and k% or more in n% trials.

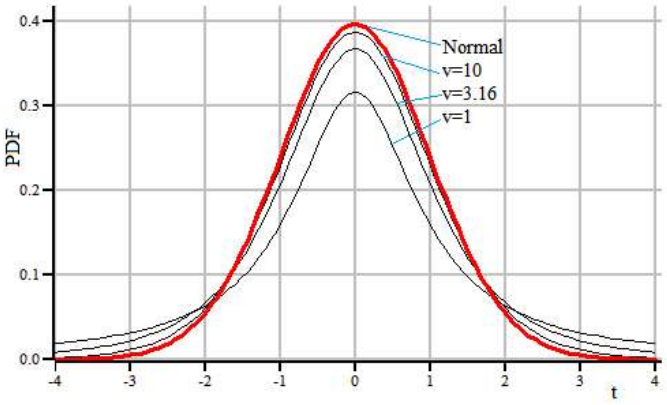
```
'p      The probability per trial of the event
'k%     A number of events in the range 0 to n%
'n%     The number of trials (greater than 0)
'Return the probability that an event occurs k% or fewer times in n% trials.
Func BinomialLE(p, k%, n%)
if k% >= n% then return 1.0 endif;
if k% < 0 then return 0.0 endif;
return BetaI(n%-k%, 1+k%, 1-p);
end;

'p      The probability per trial of the event
'k%     A number of events in the range 0 to n%
'n%     The number of trials (greater than 0)
'Return the probability that an event occurs k% or more times in n% trials.
Func BinomialGE(p, k%, n%)
if (k% <= 0) then return 1.0 endif;
if (k% > n%) then return 0.0 endif;
return BetaI(k%, n%-k%+1, p);
end;
```

The BinomialC() function will evaluate binomial coefficients for you, avoiding arithmetic overflow as much as is possible.

See also:
BetaI(), BinomialC(), Student's T Distribution, F-Distribution

Student's T Distribution

The Student's T distribution is a symmetric probability density function (PDF) used when asking questions about the mean of a sample taken from a normally distributed population when the number of samples is small. It is characterised by the number of degrees of freedom (which is usually the number of samples used to estimate the mean minus 1), and becomes indistinguishable from the Normal distribution when the degrees of freedom reaches a hundred or so. If you want to graph the function, you can do so with this script expression:




```
const pi := 3.141592653589793;
func StudentsTDist(t, v)
return pow(1+t*t/v, -(v+1)/2)*Exp(LnGamma((v+1)/2) - LnGamma(v/2))/Sqrt(v*pi);
end;
```

The t value is a statistic that relates to how different two means are and can be positive or negative. Exactly how it is calculated depends on the situation; we give some examples later in this section, but the simplest case is where you have a normally distributed population and you take single sample of `n%` items with a sample mean `Sm` and sample standard deviation `Ssd`. If the true mean of the population is `mean`, the `t` statistic is:

```
t := (Sm-mean)*Sqrt(n%)/Ssd;
```

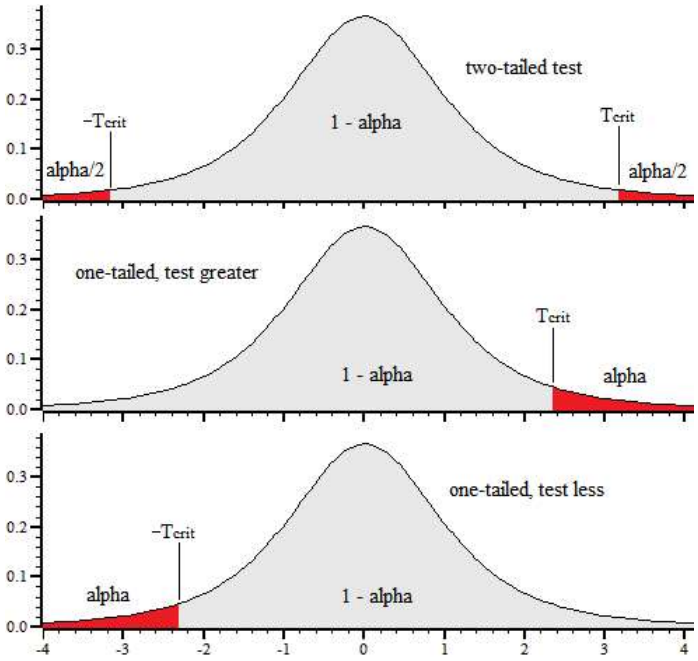
When asking questions about how likely one would be to observe a mean value that is different from or greater than or less than some value, the more useful functions are the cumulative probability distribution functions (in the range 0 to 1 as `t` varies from minus infinity to plus infinity), and the complement of this (1 minus it). These are usefully calculated from the incomplete beta function, as follows:

```
't   The t statistic
'v   The number of degrees of freedom (>0)
Func StudentCDF(t, v)
var tt := t*t;
var z := BetaI(0.5*v, 0.5, v/(v+tt))*0.5;
return t < 0 ? z : 1-z;
end;
```

```
'This is 1.0-StudentCDF(t, v);
Func OneMinusStudentCDF(t, v)
var tt := t*t;
var z := BetaI(0.5*v, 0.5, v/(v+tt))*0.5;
return t < 0 ? 1-z : z;
end;
```

The `StudentCDF()` function returns the probability that one can reject the Null hypothesis that the means are not different (or not greater than or not less than some value). The value we usually want is 1-`StudentCDF()`, which we calculate separately to preserve the accuracy for small values. In statistical tests, the probability level we are interested in, for example 0.05 (5%) or 0.01 (1%) is often referred to as *alpha*. `StudentCDF()` can be compared with 1-alpha, which is called the confidence level (95% or 99%), `OneMinusStudentCDF()` returns the probability that the hypothesis being tested would occur by chance and so can be rejected. That is, you would reject the hypothesis if `OneMinusStudentCDF() >` alpha.

One-tail and two-tail tests



If you are asking the question, "are these two mean values different", you want to know how likely is a sampled mean value to occur by chance. If you want to be 99% certain that your sample differs, your t value needs to differ from 0 by a sufficient value that the cumulative probability density has reached a value such that 99% of the PDF lies at lesser values. As the PDF is symmetric about 0, this happens when 1% of the PDF is excluded, which happens when 0.5% is above some critical *t* value *Tcrit* and 0.5% is below -*Tcrit*. In this case your alpha is 0.01 (or 1%) and the value you would test for is alpha/2. This is a two-tail test as you care what happens at both ends of the probability distribution.

If you are asking the question, "is this mean greater than that mean", you are only looking at one end of the distribution and the value you would test against is alpha. If you want to be 99% certain, *Tcrit* is now the level at which 99% of the distribution lies below, your alpha value is 0.01 and this is the value to test for. Due to the symmetry of the distribution, to test that "this mean is less than that mean" you need a negative t value; the probability to test for is the same as for a positive t value. A one-tail test should be used with caution.

Inverse of the distribution

The functions above calculate alpha given a t statistic and the number of degrees of freedom. If you want to specify alpha and calculate the t statistic value that corresponds to this, you need the inverse function. The following script code will do this for you:

```
'Function used by ZeroRoot() to locate the t value with the required probability
var gProb, gV;      'probability and degrees of freedom
func TZeroRoot(t)
return gProb - OneMinusStudentCDF(t, gV);
end;

'TValueFor: This calculates the t-value that you need to reject the NULL hypothesis
'given an alpha, the number of degrees of freedom and if you want a one sided test.
'alpha The desired probability level (typically 0.05)
'v     Degrees of freedom (> 0.0)
'os%   0=two sided, non-zero for 1 sided
func TValueFor(alpha, v, os%)
var t;
gProb := os% ? alpha : alpha*0.5;
gV := v;
ZeroFind(t, TZeroRoot, -20, 20);
return t;
end;
```

Examples

The following example code follows examples given for the Boost C++ libraries, which in turn follow NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/ (NIST is an agency of the US Commerce Department).

Confidence range of the mean

You have `Sn%` sample values taken from a normally distributed population with mean `Sm` and standard deviation `Ssd` and you want to know in what range of values can you be confident that you have bracketed the true population mean. This is plainly a two-tail test as we care about values in both directions. If you want to be

certain at the alpha level, the Tcrit value is given by: TValueFor(alpha/2, Sn%-1, 0) and the range is from Sm-Tcrit*Ssd/Sqrt(Sn%) to Sm+Tcrit*Ssd/Sqrt(Sn%). See here for details.

Test a sample mean for difference from a known mean

This test comes up if you already know what the mean should be (for example in quality control) and you want to know if there is a change. It can also come up in a paired test where you measure a value before and after a treatment on a set of items and your data set is the difference between before and after for each item, so you are asking is there a difference from zero. The following routine does this calculation:

```
'testing sample mean for difference from a "true" mean.
'n%     Number of items in the sample
'Sm     Sample mean
'Ssd    Sample standard deviation
'mean   The "true" mean
'os%    -1 means one-sided test for below mean, +1 means one-sided test for
'       above the mean, 0 means two-sided test for different
'Return the probability that the Null hypothesis (that the sample mean is
'       not different, greater or less than the true mean).
Func TSingleSample(n%, Sm, Ssd, mean, os%)
var t := (Sm - mean)* Sqrt(n%)/Ssd;
PrintLog("\nStudent's t test for a single sample\n");
PrintLog("Number of observations     = %D\n", n%);
PrintLog("Sample mean                = %g\n", Sm);
PrintLog("Sample Standard Deviation  = %g\n", Ssd);
PrintLog("Expected True mean         = %g\n", mean);
PrintLog("T Statistic                = %g\n", t);
PrintLog("Degrees of freedom         = %d\n", n%-1);
if (os% = 0) then t := abs(t) endif;
if (os% < 0) then t := -t endif;
var prob := OneMinusStudentCDF(t, n%-1);  'Single tailed value
if (os% = 0) then prob *= 2.0 endif;
PrintLog("Probability due to chance  = %g\n", prob);
return prob;
end;
```

In this test we have a single sample of n% items with mean Sm (Sample mean) and standard deviation Ssd (Sample standard deviation). The question is: is the mean different (two-tail) or greater or less (one-tail) that the known mean. The return value is the probability that the Null hypothesis can be rejected and is the value to compare against the alpha value for your test. You may choose to omit the PrintLog() lines. To use this function, for example using data from the NIST site: 3 observations with mean of 37.8, standard deviation of 0.96437 and expected true mean of 38.9:

```
TSingleSample(3, 37.8, 0.96437, 38.9, -1); 'Test for mean < 38.9
TSingleSample(3, 37.8, 0.96437, 38.9, 0);  'Test for mean <> 38.9
TSingleSample(3, 37.8, 0.96437, 38.9, 1);  'Test for mean > 38.9
```

The output is (omitting repeated lines for second and third cases):

```
Student's t test for a single sample
Number of observations     = 3
Sample mean                = 37.8
Sample Standard Deviation  = 0.96437
Expected True mean         = 38.9
T Statistic                = -1.97565
Degrees of freedom         = 2
Probability due to chance  = 0.093429 (Hypothesis: mean < 38.9)
...
Probability due to chance  = 0.186858 (Hypothesis: mean <> 38.9)
...
Probability due to chance  = 0.906571 (Hypothesis: mean > 38.9)
```

With an alpha of 0.05 (5%), all the hypotheses are rejected. However, if the alpha level was set at 0.1 (10%), then the hypothesis that the mean is less than 38.9 is not rejected.

Test two samples with equal variance for difference of means

In this case we have a first sample with n1% items with mean S1m and standard deviation S2sd and a second sample with n2% items and mean S2m and standard deviation S2sd. Again, we want to ask if the means are

different, or if the first mean is less than or greater than the second. The code to do this is (you can delete the `PrintLog()` lines if they are not wanted):

```
'ni%    The number in each set of samples
'Sim    The mean of sample i
'Sisd   The standard deviation of sample i
'os%    0 for two sided. 1 for sm1 > sm2, -1 for sm1 < sm2
'Return the probability that the Null hypothesis (that the sample means
'       are not different, Sm1 greater or less than Sm2).
Func TTwoSamplesSameVar(n1%, S1m, S1sd, n2%, S2m, S2sd, os%)
'Calculate the combined standard devaition of both samples
var S12sd := Sqrt(((n1%-1)*S1sd*S1sd + (n2%-1)*S2sd*S2sd)/(n1%+n2%-2));
var t := (S1m - S2m)/(S12sd * Sqrt(1.0/n1% + 1.0/n2%));
PrintLog("\nStudent's t test for two samples, same variance\n");
PrintLog("Number of observations     = %8d %8d\n", n1%, n2%);
PrintLog("Sample means               = %8g %8g\n", S1m, S2m);
PrintLog("Sample Standard Deviation  = %8g %8g\n", S1sd, S2sd);
PrintLog("T Statistic                = %g\n", t);
PrintLog("Degrees of freedom         = %d\n", n1%+n2%-2);
if (os% = 0) then t := abs(t) endif;
if (os% < 0) then t := -t endif;
var prob := OneMinusStudentCDF(t, n1%+n2%-2);  'One-tail value
if (os% = 0) then prob *= 2.0 endif;
PrintLog("Probability due to chance  = %8e\n", prob);
return prob;
end;
```

As the two samples have the same variance we can form a pooled standard deviation by combining the two standard deviations, which is what the calculation for `S12sd` does. The number of degrees of freedom is just the sum of the degrees of freedom of the two data sets The calculation of the t statistic is then very similar to the previous single sample test, just weighting by the two sample sizes. As an example, we consider the following two data sets:

Data set	Observations	Mean	Sd
1	249	20.1446	6.4147
2	79	30.481	6.10771

The code to test each possible hypothesis is:

```
TTwoSamplesSameVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771, -1); 'mean 1 < mean 2
TTwoSamplesSameVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771,  0); 'mean 1 <> mean 2
TTwoSamplesSameVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771,  1); 'mean 1 > mean 2
```

The output (omitting duplicated lines is):

```
Student's t test for two samples, same variance
Number of observations     =      249       79
Sample means               =  20.1446   30.481
Sample Standard Deviation  =   6.4147  6.10771
T Statistic                = -12.6206
Degrees of freedom         = 326
Probability due to chance  = 2.636609e-030 (Hypothesis: mean1 < mean 2)
...
Probability due to chance  = 5.273218e-030 (Hypothesis: mean1 <> mean 2)
...
Probability due to chance  = 1.000000e+000 (Hypothesis: mean 1 > mean 2)
```

This says that the hypotheses that mean1 is less than mean2 or that mean1 is different from mean2 cannot be rejected. The hypothesis that mean1 is greater than mean2 is rejected.

Test two samples with non-equal variance

Finally we have the code for two samples with non-equal variance. This time we cannot pool the standard deviations, and we also have a problem with the degrees of freedom. To calculate this, the Welch-Scatterthwaite approximation is used. Note that the approximation improves as the number of degrees of freedom increases. As ever, you can delete all the `PrintLog()` lines if they are not required:

```
Func TTwoSamplesDiffVar(n1%, S1m, S1sd, n2%, S2m, S2sd, os%)
var s1 := S1sd*S1sd/n1%;
var s2 := S2sd*S2sd/n2%;
```

```
'Calculate the t value
var t := (S1m - S2m)/Sqrt(s1 + s2);
'Calculate the combined degrees of freedom using the Welch-Scatterthwaite approximation.
var v := (s1+s2)*(S1+s2)/(s1*s1/(n1%-1) + s2*s2/(n2%-1));
PrintLog("\nStudent's t test for two samples, different variance\n");
PrintLog("Number of observations     = %8d %8d\n", n1%, n2%);
PrintLog("Sample means               = %8g %8g\n", S1m, S2m);
PrintLog("Sample Standard Deviation  = %8g %8g\n", S1sd, S2sd);
PrintLog("T Statistic                = %g\n", t);
PrintLog("Degrees of freedom         = %g\n", v);
if (os% = 0) then t := abs(t) endif;
if (os% < 0) then t := -t endif;
var prob := OneMinusStudentCDF(t, v);   'One-tail value
if (os% = 0) then prob *= 2.0 endif;    'Convert to two-tail value
PrintLog("Probability due to chance  = %8e\n", prob);
return prob;
end;
```

If we use the same input data as for the previous example, we have:

```
TTwoSamplesDiffVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771, 0); 'mean 1 < mean 2
TTwoSamplesDiffVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771, 0); 'mean 1 <> mean 2
TTwoSamplesDiffVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771, 0); 'mean 1 > mean 2
```

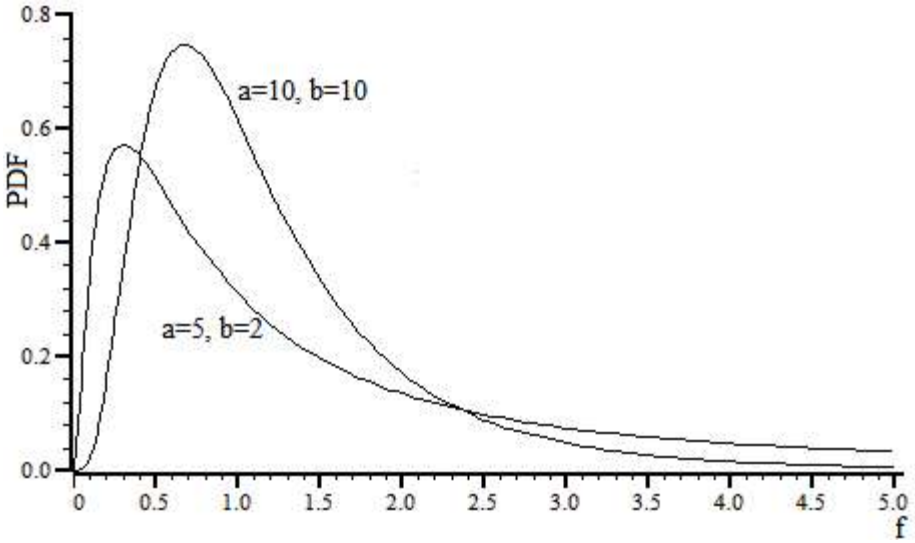
The output (omitting repeated lines) is:

```
Student's t test for two samples, different variance
Number of observations     =      249       79
Sample means               =  20.1446   30.481
Sample Standard Deviation  =   6.4147  6.10771
T Statistic                = -12.9463
Degrees of freedom         = 136.875
Probability due to chance  = 7.854523e-026 (Hypothesis: mean1 < mean 2)
...
Probability due to chance  = 1.570905e-025 (Hypothesis: mean1 <> mean 2)
...
Probability due to chance  = 1.000000e+000 (Hypothesis: mean1 > mean 2)
```

In this case, although the probability that this occurred by chance is increased by several orders of magnitude compared to the equal variance situation, the conclusions are the same.

See also:
`BetaI()`, Binomial Distribution, F-Distribution

F-Distribution



F Distribution Probability Density Function

The F Distribution probability density functions shows the likelihood of particular values of the F statistic occurring. The F statistic calculation depends on the situation; in a simple case it is the ratio of two variances. The F distribution can be calculated and displayed by the script code:

```
'The derivative of the BetaI function, used to calculate the Prob density of
'the F-Distribution. Maths as suggested by the Boost library.
Func BetaIDerivative(a, b, x)
if (a<=0.0) or (b<=0.0) or (x<0) or (x > 1.0) then return -1 endif;
if (x=0.0) or (x=1.0) then return 0 endif;
return  exp((b-1)*ln(1-x)+(a-1)*ln(x))/BetaI(a,b);
end;

'The Probability Density function for the F-Distribution. The use of two
'methods keeps the third argument to the derivative function away from 1.
Func FDistribution(a, b, x)
var z := b + a* x;
var y := a * b / (z * z);
if (a*x > b) then
    return y * BetaIDerivative(b/2, a/2, b/z);
else
    return y * BetaIDerivative(a/2, b/2, a*x/z);
endif;
end;

'Draw the F distribution from x=0 to range with a and b degrees of freedom
Func ShowFDist(a, b, range)
const n% := 100;        'points after 0
const np% := n%+1;      'total points to plot
var rv% := SetResult(np%, range/n%, 0, "F distibution", "f", "PDF");
DrawMode(-1,13);        'cubic spline mode
var i%, x;
for i% := 0 to n% do
    x := i%*(range/n%);
    [i%] := FDistribution(a, b, x);
next;
WindowVisible(1);
return rv%;
end;

ShowFDist(5, 2, 5);  'Draw an example
```

Unlike the Student's T distribution, the F distribution is asymmetric. For us, a more interesting and useful function is the Cumulative Distribution Function (CDF) of the F distribution. For statistical use, the complement of the CDF (1-CDF) is usually the most useful. The CDF and its complement can be calculated with:

```
'The CDF for the F-Distribution.
func FDistCDF(a, b, x)
var ax := a*x;
var z := b+ax;
return BetaI(a/2, b/2, ax/z);
end;

'We usually want the complement of the CDF
func OneMinusFDistCDF(a, b, x)
var ax := a*x;
var z := b+ax;
return BetaI(b/2, a/2, b/z);
end;
```

The value of the `FDistCDF()` function increases from 0 to 1 as x (the F statistic) varies between 0 and infinity. The value of `OneMinusFDistCDF()` decreases from 1 to 0 over the same range.

Inverse of the F distribution CDF complement

When asking questions such as, "What F value does 5% of the F-Distribution lie above?", we find the inverse of the complement of the F distribution CDF at the 5% level. This can be calculated by the `InvFCDFComp()` function:

```
var gA, gB, gAlpha; 'variables used by InvFCDFComp()

'Helper function for InvFCDFComp(), used by ZeroFind()
Func ZeroFRoot(x)
return OneMinusFDistCDF(gA, gB, x)-gAlpha;
end;

'Find the inverse of the complement of the cumulative F Distribution
'alpha The probability level
'a,b,  The degrees of freedom of the two distributions
Func InvFCDFComp(alpha, a, b)
var x;
gA := a; gB := b; gAlpha := alpha;
ZeroFind(x, ZeroFRoot, 0, 20);
return x;
end;
```

F-Test for equality of two standard deviations

The simplest use of the F-test is to determine if the standard deviation of two populations are the same. See this NIST handbook section for an explanation. The `FTestSameSd()` function calculates the one-tailed probability value and also returns the critical F statistic values for both the one-tailed test (alpha) and also for a two-tailed test (alpha/2). For a two-tailed test, both the critical levels at `alpha/2` and `1-alpha/2` apply. For the one-tailed test, only one of the levels at `alpha` or `1-alpha` apply. In this case, the F statistic is just the ratio of the squares of the two standard deviations and the number of degrees of freedom is the number of samples of each distribution minus 1:

```
'F-test for equal standard deviations
'n1%   Number of items in sample 1
'sd1   Standard deviation of sample 1
'n2%   Number of items in sample 2
'sd2   Standard deviation of sample 2
'alpha The probability level to calculate FCrit
Func FTestSameSd(n1%, sd1, n2%, sd2, alpha)
var f :=  sd1/sd2;      'The test statistic is the ratio...
f *= f;                 '...of the two sd's squared
var prob := OneMinusFDistCDF(n1%-1, n2%-1, f);
PrintLog("\nF-Test for same standard deviations\n");
PrintLog("Number of samples   = %8d %8d\n", n1%, n2%);
PrintLog("Standard deviations = %8g %8g\n", sd1, sd2);
PrintLog("Test statistic (F)  = %8g\n", f);
PrintLog("Prob (one-tailed)   = %8g\n", prob);
PrintLog("Upper critical level at alpha/2 = %8g\n", InvFCDFComp(alpha/2, n1%-1, n2%-1));
PrintLog("Upper critical level at alpha   = %8g\n", InvFCDFComp(alpha, n1%-1, n2%-1));
PrintLog("Lower critical level at alpha   = %8g\n", InvFCDFComp(1-alpha, n1%-1, n2%-1));
PrintLog("Lower critical level at alpha/2 = %8g\n", InvFCDFComp(1-alpha/2, n1%-1, n2%-1));
return prob;
end;
```

With the following data:

Sample	Items	sd
1	11	4.9082
2	9	2.5874

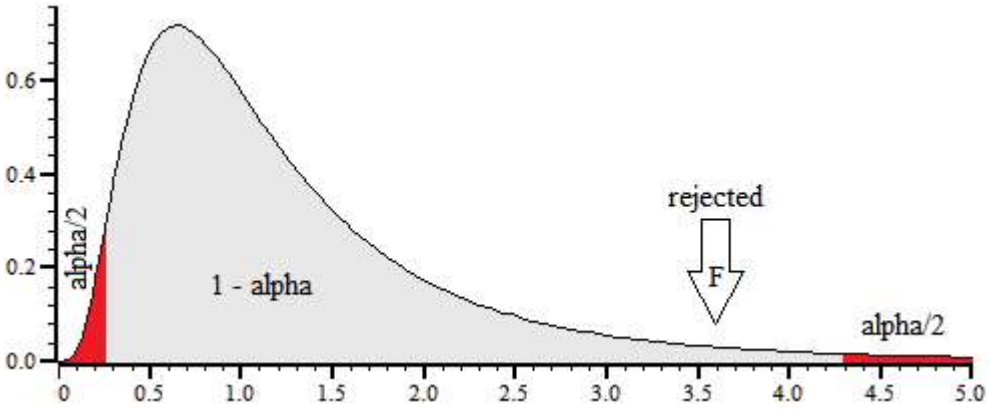
We use the script to test at an alpha level of 0.05:

```
FTestSameSd(11, 4.9082, 9, 2.5874, 0.05);
```

and the output is:

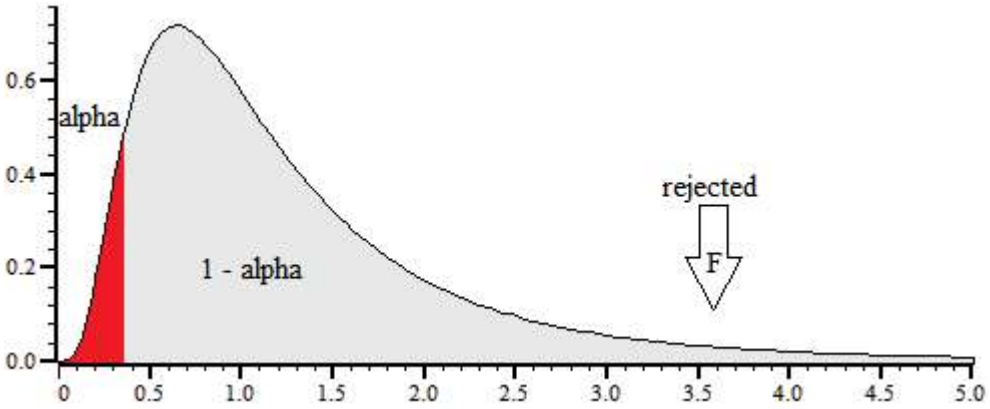
```
F-Test for same standard deviations
Number of samples   =       11        9
Standard deviations =   4.9082   2.5874
Test statistic (F)  =  3.59847
Prob (one-tailed)   = 0.0411439
Upper critical level at alpha/2 =  4.29513 (two-tailed test upper limit)
Upper critical level at alpha   =  3.34716 (sd1 > sd2 one-tailed limit)
```

```
Lower critical level at alpha   = 0.325557 (sd1 < sd2 one-tailed limit)
Lower critical level at alpha/2 = 0.259411 (two-tailed test lower limit)
```



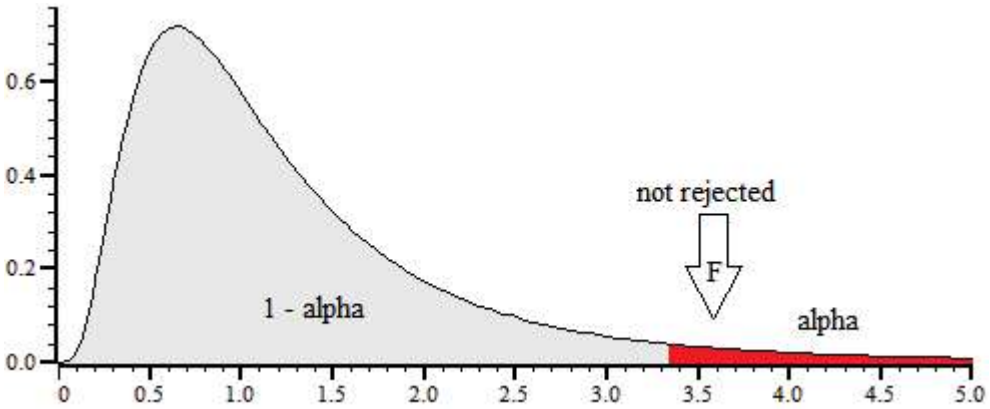
Two-tailed F-test

Given this result, for a two-tailed test the hypothesis that the standard deviations are different is rejected at the 5% level as the F statistic does not fall outside the range 0.259411 to 4.29513. The probability value returned (that the result occurred by chance) is for comparison with alpha for a one-tailed test; for a two-tailed test compare it with alpha/2 or double it for comparison with alpha .



One-tailed F test for less

For a one-tailed test that the standard deviation is less, the result is a clear rejection (as you would expect as the value was greater).



One-tailed F-test for greater

For a one-tailed test, the probability value is less than alpha, and we can see that the hypothesis that sample 1 standard deviation is greater than sample 2 is not rejected at the 5% level.

See also:
BetaI(), Binomial Distribution, Student's T Distribution

BinError()

This function is used in a result view to access the optional error (standard deviation about the mean) and bin counts (number of items added to each bin) arrays. Errors are enabled for a result view created with `SetAverage()` or `SetResult()` with 4 added to `flags%`; enable bin counts by adding 32 to `flags%`. To set a non-zero error the sweep count must be greater than 1. Use `Sweeps()` to set the count. The first command variant transfers data for a single bin, the others transfer an array of bins:

```
Func BinError(chan%, bin%{, newSD{, newCt%});
Func BinError(chan%, bin%, sd[]{, set%{, Ct%[]});
Func BinError(chan%, bin%, 0, set%, Ct%[]);
```

`chan%` The channel number in the result view.

`bin%` The first bin number for which to get or set the error information.

`newSD` If present, this sets the standard deviation for bin `bin%`.

`newCt%` If present, this sets the item count (if item counts are enabled) for bin `bin%`.

`sd[]` An array for standard deviations or 0 for no transfer. Transfers start at bin `bin%` in the result view. If the array is too long, extra bins are ignored. If there is no error information, setting has no effect and reading fills `sd[]` with 0.

`set%` If present and non-zero, the values in `sd[]` and `Ct%[]` are copied to the result view. If omitted or zero, values are copied from the view into `sd[]` and `Ct%[]`.

`Ct%[]` If present, used to transfer item count information. If there is no item count, setting has no effect and reading fills the array with the current sweep count.

Returns The first command variant returns the standard deviation at the time of the call. The other variants returns the number of bins copied. If there are 0 or 1 sweeps of data the standard deviation set will be zero.

To illustrate how errors are calculated, we will assume that we are dealing with an average that is set to display the mean of the data in each bin. In terms of the script language, if the array `s[]` holds the contribution of each sweep to a particular bin, the mean, standard deviation and standard error of the mean are calculated as follows:

```
var mean, sd:=0, i%, diff, sem;
for i%:= 0 to Sweeps()-1 do mean += s[i%] next; 'form sum
mean /= Sweeps();              'form mean data value
for i%:= 0 to Sweeps()-1 do
    diff := s[i%]-mean;        'difference from mean
    sd += diff*diff;           'sum squares of differences
    next;
sd := Sqrt(sd/(Sweeps()-1));   'the standard deviation
sem := sd/Sqrt(Sweeps());      'the standard error of the mean
```

We divide by `Sweeps()-1` to form the standard deviation because we have lost one degree of freedom due to calculating the mean from the data. If the result view has item counts per bin enabled (add 32 to `flags%` in `SetAverage()` or `SetResult()`) then replace `Sweeps()` by the item count for the bin.

See also:
`BinSize(), BinToX(), SetAverage(), SetResult(), Sweeps()`

BinomialC

This function calculates the Binomial coefficient ${}_nC_k$, which is the number of different ways to choose k items from n, which is `n!/(k!*(n-k)!)`. As n factorial grows rapidly as n increases, this can be difficult to compute for even a modest value of n.

```
Func BinomialC(n%, k%)
```

`n%` The number of items from which to choose. This must be greater than 0.

`k%` The number to choose, in the range 0 to `n%`.

Returns The binomial coefficient. The return is integral, but is returned as a real value as it can be very large, for example `BinomialC(34,17)` is 2333606220. Floating point numbers can represent integers exactly up to some 15 digits, after which accuracy cannot be maintained. If `n%` exceeds 1029 the result can be infinity, which means it is greater than `1.7977e+308`, the largest real number we can represent. If you really need results for very large `n%`, you can get the logarithm of the result with `LnGamma(n%+1)-LnGamma(k%+1)-LnGamma(n%-k%+1)`.

See also:
`BetaI(), GammaP() and GammaQ(), LnGamma()`

BinSize()

In a result view, this returns the x axis increment per bin. In a time view, the value returned depends on the channel type.

```
Func Binsize({chan%});
```

`chan%` In a time view this is the channel from which to return information. If you omit the argument, the function returns the file time resolution in seconds. In a result view, `chan%` is ignored, and should be omitted.

Returns In a time view, the sampling interval between points is returned for Waveform, WaveMark and RealWave channels or a negative number if the channel does not exist. Otherwise the underlying time resolution of the file in seconds is returned.

See also:
`BinToX(), XToBin()`

BinToX()

This converts bin numbers to x axis units in the current result view. If the current view is a time view, it converts the underlying Spike2 time units into time in seconds.

```
Func BinToX(bin);
```

`bin` A bin number in the result view. You can give a non-integer bin number without error. If you give a bin number outside the result view, the bin number is limited to the range of the result view before it is converted to an x axis value.

The x axis range of a result view is `BinToX(0)` to `BinToX(MaxTime())`. Do not confuse this range with `XLow()` to `XHigh()`, which is the visible range of the x axis in the current view.

In a time view, this is in the underlying time units. If the value is beyond the x axis range, it is limited to the x axis range. The value need not be integral, but you should note that all data items in the time view have time stamps that correspond with integral values of `bin`. The returned value is in seconds.

Returns It returns the equivalent x axis position.

See also:
`BinSize(), MaxTime(), XHigh(), XLow(), XToBin()`

BRead()

This reads data into variables and arrays from a binary file opened by `FileOpen()`. The function reads 32-bit integers, 64-bit IEEE real numbers and zero-terminated strings that are stored as UTF-8 (by default, but can be stored as UFT-16LE, see below).

```
Func BRead(&arg1|arg1[]|&arg1%|arg1%[]|&arg1$|arg1$[] {,...});
```

`arg` Arguments may be of any type. Spike2 reads a block of memory equal in size to the combined size of the arguments and copies it into the arguments. Strings or string arrays are read a byte at a time until a zero byte is read.

Returns It returns the number of data items for which complete data was read. This will be less than the number of items in the list if end of file was reached. If an error occurred during the read, a negative code is returned.

For reasons of backwards compatibility, although integers are now 64-bit in Spike2, integer reads are treated as signed 32-bit numbers and the values read are sign extended into 64-bits. If you want to read 64-bit integers, or strings encoded as UTF-16LE, see the notes for `BReadSize()`.

See also:
`FileOpen(), BReadSize(), BRWEndian(), BSeek(), BWrite()`

BReadSize()

This converts data into variables and arrays from a binary file opened by `FileOpen()`. The function reads 8, 16 and 32-bit integers and converts them to 32-bit integers, and 32 and 64-bit IEEE real numbers and converts them to 64-bit reals. It also reads strings from fixed-size regions in the file (zero bytes are ignored during the read). The read is from the current file position. The current position after the read is the byte after the last byte read.

```
Func BReadSize(size%, &arg|arg[]|&arg%|arg%[]|&arg$|arg$[]{,...});
```

`size%` The bytes to read for each argument. Legal values depend on the argument type:

Integer	1, 2, 4, 8	Read 1, 2 or 4 bytes and sign extend to 64-bit integer or read 8 bytes (no sign extend needed).
	-1, -2, -4	Read 1, 2 or 4 bytes and zero extend to 64-bit integer.
Real	4	Read 4 bytes as 32-bit real, convert to 64-bit real.
	8	Read 8 bytes as 64-bit real.
String	n	Read n bytes/characters into a string. Null characters end the string as seen by the script, but all n bytes are read..
	0	Read 1 character at a time until either a Null character or the end of the file is found.
	-n	Read 1 character at a time until either a Null character or n characters have been read (added at version 8.00).

`arg` The target variable(s) to be filled with data. `size%` applies to all targets.

Returns It returns the number of data items for which complete data was read. This will be less than the number of items in the list if end of file was reached. If an error occurred during the read, a negative code is returned.

Unicode

If you read a string and the `FileOpen() mode%` did not have 16 added, the string is assumed to be stored as UTF-8 and it is converted to UTF-16LE before being returned. Beware that this means that the number of bytes needed to hold the string may be more than the number of characters reported by Len(). If you add 16 to the `mode%` in `FileOpen()`, strings are read as 16-bit characters and the n refers to the count of 16-bit characters.

See also:
`FileOpen(), BRead(), BRWEndian(), BSeek(), BWrite(), BWriteSize()`

BRWEndian()

This gets and sets the "endianism" of binary data files. This affects numeric data used with `BRead()`, `BReadSize()`, `BWrite()` and `BWriteSize()`. PC programs normally use little-endian data (least significant byte at lowest address). Some systems, including the Macintosh, use big-endian data (most significant byte at lowest address). Binary files are little-endian by default.

Most users do not need to use this routine. Only use it if you are writing binary files for use on a big-endian computer or reading binary files that were generated with a big-endian system.

```
Func BRWEndian({new%});
```

new% Omit or set –1 for no change. Set 0 for little-endian and 1 for big-endian.

Returns The current endianism as 0 for little, 1 for big or a negative error code.

See also:
FileOpen(), BRead(), BReadSize(), BSeek(), BWrite(), BWriteSize()

BSeek()

This function moves and reports the current position in a file opened by FileOpen() with a type% code of 9. The next binary read or write operation to the file starts from the position returned by this function.

```
Func BSeek({pos% {, rel%}});
```

pos% The new file position as the byte offset the start, the current position, or the end of the file. If pos% is omitted, the position is not changed and the function returns the current position.

rel% This determines to what the new position is relative:

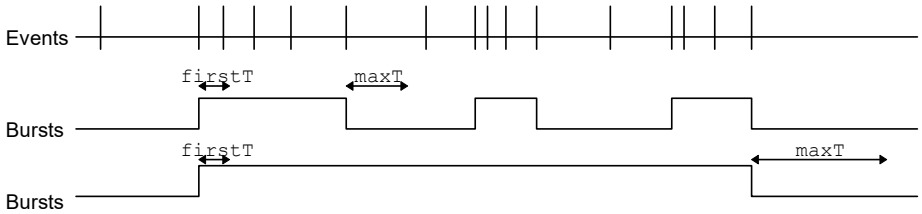
0	Relative to the start of the file (same as omitting the argument).
1	Relative to the current position in the file.
2	Relative to the end of the file.

Returns The new file position relative to the start of the file or a negative error code.

See also:
FileOpen(), BReadSize(), BRead()

BurstMake()

This function extracts burst start and end times from an event channel and writes them to a memory channel. Three parameters control the formation of bursts: the interval between the first two events in a burst, the interval between the last event in a burst and any following event, and the minimum events in a burst.



The diagram illustrates the method of forming bursts. In the first case, with a short maximum interval between events, the algorithm finds three bursts. In the second case with a longer period, the algorithm detects one burst. The command for this function is:

```
Func BurstMake(mChan%,eChan%,sTime,eTime,maxT{,firstT{,minE%}});
```

mChan% The channel number of an event, level event or marker channel created by MemChan() for the output. If this is a marker channel, the start of each burst is given a first marker code of 00 and the end has a first marker code of 01.

eChan% The channel number of an event channel to search for bursts.

sTime The start of the time range to search for bursts.

eTime The end of the time range to search for bursts.

maxT The maximum time between two events (after the first pair) for the events to lie in the same burst.

firstT The maximum time between the first two events in a burst. If omitted, maxT sets the time interval for the first pair of events.

minE% The minimum number of events that can make up a burst. The default is 2.

Returns The function returns the number of bursts found, or a negative error code.

See also:
BurstRevise(), BurstStats(), MemChan()

BurstRevise()

This modifies a list of times, indicating bursts and produces a new list of bursts that have inter-burst intervals and burst durations greater than specified minimum times (See BurstStats() for more information).

```
Func BurstRevise(mChan%, eChan%, sTime, eTime, minI, minD);
```

mChan% A memory channel created by MemChan() holding event data to which the output of this command is added. This can be the same channel as eChan%, but if it is, the output is generated by deleting unwanted events from the channel.

eChan% The event or marker channel to read burst times from.

sTime The start of the time range to revise.

eTime The end of the time range to revise.

minI The minimum interval between the end of one burst and the start of the next. Bursts with shorter intervals are amalgamated.

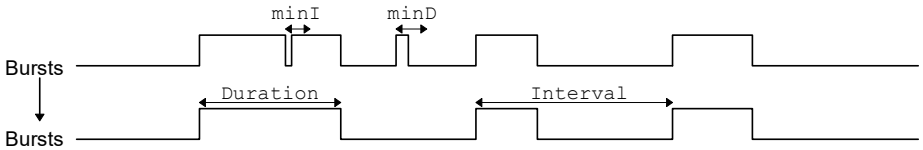
minD The minimum duration of a burst. Shorter bursts are deleted.

Returns The number of bursts in the output or a negative error code.

See also:
BurstMake(), BurstStats(), MemChan()

BurstStats()

This command returns statistics on bursts (possibly made by BurstMake()). Additional rules can be applied to the bursts before the statistics are calculated to amalgamate bursts that are too close together and to delete bursts that are too short. The statistics are the mean and standard deviation of the duration of the bursts and the intervals between the start of one burst and the start of the next.




```
Func BurstStats(eChan%, sTime, eTime, &meanI, &sdI, &meanD,
&sdD{,minI{,minD}});
```

eChan% The event channel containing burst data. The first event found in the time range is assumed to be the start of a burst, the second the end, and so on.

sTime The start of the time range to process.

eTime The end of the time range to process.

meanI This is a real variable that is returned holding the mean interval between the starts of bursts (as long as at least 2 bursts were found).

sdI This is a real variable that is returned holding the standard deviation of the mean interval (as long as at least 3 bursts were found).

meanD This is a real variable that is returned holding the mean burst duration (as long as at least 1 burst was found).

sdD This is a real variable that is returned holding the standard deviation of the burst durations (as long as at least 2 bursts were found).

`minI` This optional value sets the minimum interval between the end of one burst and the start of the next. It takes the value 0.0 if omitted. Bursts that are closer than this are amalgamated for the purpose of forming statistics.

`minD` This optional value (taken as 0.0 if omitted) sets the minimum burst duration. Bursts shorter than this (after amalgamations) are ignored in the statistics.

Returns It returns the number of bursts used for the statistics, or a negative error code.

This command is used in scripts that optimise the values of `minI` and `minD` for forming bursts. For example, when bursting is known to follow a cyclical pattern, one would like to find values of `minI` and `minD` that minimise the total coefficient of variance:

Total coefficient of variance = (`sdD/meanD`)^2 + (`sdI/meanI`)^2

Once suitable values are found, the `BurstRevise()` command can generate a memory channel holding bursts based on the optimised parameters.

See also:
`BurstMake(), BurstRevise(), MemChan()`

BWrite()

This function writes binary data values and arrays into a file opened by `FileOpen()` with a `type%` code of 9. The function writes 32-bit integers (for backwards compatibility with version 7), 64-bit IEEE real numbers and strings. The output is at the current position in the file. The current position after the write is the byte after the last byte written. To write 64-bit integers, or if you are concerned about the representation of Unicode strings, see `BWriteSize()`.

```
Func BWrite(arg1 {,arg2 {,...}});
```

`arg` Arguments may be of any type, including arrays. Spike2 fills a block of memory equal in size to the combined size of the arguments with the data held in the arguments and copies it to the file. An integer uses 4 bytes and a real uses 8 bytes. A string is written in UFT-8 format with a zero byte to mark the end. Use `BWriteSize()` to write a fixed number of bytes. The arguments are not modified, so `const` arrays can be written.

Returns It returns the number of arguments for which complete data was written. If an error occurred during the write, a negative code is returned.

The following example writes data from a channel of a data file as binary:

```
'Declare variables:
var ok%,chan%,fh%,bh%,close%:=0, fmt%:=0, fmt$[3];
fmt$[0] := "Integer"; fmt$[1]:="32-bit IEEE real";
fmt$[2] := "64-bit IEEE real";
if ViewKind()<>0 then
   fh% := FileOpen("", 0, 0, "File to dump as binary");
   if fh%<=0 then Message("No file to dump"); halt endif;
   close% := 1;                'say we should close file
endif;
fh% := View();                 'get the starting view handle
DlgCreate("Binary file dump");  'Start new dialog
DlgChan(1,"Choose the channel to dump",16511);
DlgList(2,"Format for the output",fmt$[]);
ok% := DlgShow(chan%, fmt%);    'ok is 0 if user cancels
if ok% <> 0 then               'dump if user selects a channel
   bh% := FileOpen("", 9, 1, "File to dump channel to");
   if bh% > 0 then BinDump(bh%, fh%, chan%, fmt%) endif;
endif;
View(bh%);FileClose();         'close the binary file
if close% then                 'if we opened the file..
   View(fh%);                  '...the move to it...
   FileClose();                '...and close it again
endif;
```

```
Proc BinDump(bh%, fh%, ch%, fmt%)     'binary file, data file, channel
docase
   case fmt% = 0 then IntDump(bh%, fh%, ch%);     'Integer format
   case fmt% = 1 then RealDump(bh%, fh%, ch%, 4); '4 byte real
   case fmt% = 2 then RealDump(bh%, fh%, ch%, 8); '8 byte real
endcase;
end;

Proc RealDump(bh%, fh%, ch%, bytes%)
const BSZ% := 8000;              'buffer size
var work[BSZ%];                  'work space
var t := 0, n%, kind%;           'start time, items read
View(fh%);                       'in data file view
kind% := ChanKind(ch%);
if (kind%=1) or (kind%=9) then   'waveform channel
   repeat
      n% := ChanData(ch%, work[], t, MaxTime(), t);
      if n% > 0 then             'if we got data, then
         View(bh%).BWriteSize(bytes%, work[:n%]);  'Output it
         t := t + n% * BinSize(ch%); 'time of next point
      endif;
   until n% <= 0;                'until no points left
else
   repeat
      n% := ChanData(ch%, work[], t, MaxTime());
      if n% > 0 then             'if we got data then...
         View(bh%).BWriteSize(bytes%, work[:n%]); 'Write it
         t := work[n%-1]+BinSize(ch%); 'next search start
      endif;
   until n% <= BSZ%;             'until buffer not full
endif;
end;

proc IntDump(bh%, fh%, ch%)
const BSZ% := 8000;
var work%[BSZ%];                 'work space
var t := 0, n%, kind%;           'start time, items read
View(fh%);                       'in data file view
kind% := ChanKind(ch%);
if (kind%=1) or (kind%=9) then   'waveform channel
   repeat
      n% := ChanData(ch%, work%[], t, MaxTime(), t);
      if n% > 0 then             'if we got data
         View(bh%).BWriteSize(2, work%[:n%]); 'Output it
         t := t + n% * BinSize(ch%); 'time of next point
      endif;
   until n% <= 0;
else                             'Times as 64-bit integer
   repeat                        '(needs version 8 or later)
      n% := ChanData(ch%, work%[], t, MaxTime());
      if n% > 0 then             'if we got any data
         View(bh%).BWriteSize(8, work%[:n%]); 'Write it
         t := BinToX(work%[n%-1]+1); 'next search start
      endif;
   until n% <= BSZ%;
endif;
end;
```

See also:

FileOpen(), BRead(), BReadSize(), BRWEndian(), BWriteSize()

BWriteSize()

This function writes variables or arrays as binary into a file opened by `FileOpen()` with a `type%` code of 9. It writes 8, 16, 32 and 64-bit integers and 32 and 64-bit reals and strings. It allows you to write formats other than the 64-bit integer and 64-bit real used internally by the script and to write variable-length strings into fixed-size fields in a binary file. Support for 64-bit integers was added at version 8.

```
Func BWriteSize(size%, arg1 {,arg2 {,...}});
```

`size%` Bytes to write for each argument (or array element if the argument is an array). Legal values of `size%` depend on the argument type:

Integer	1, 2, 4	Write least significant 1, 2 or 4 bytes.
	8	Write all 8 bytes of the integer.
	n	Any other value writes the least significant 4 bytes.
Real	4	Convert to 32-bit real and write 4 bytes.
	8	Write 8 bytes as a 64-bit real.
	n	Any other value writes 8 bytes as a 64-bit real.
String	n	Write n bytes (but see the Unicode notes, below). Pad with zeros if the string is too short.
	0	Write the full length of the string with a zero terminator.

`arg` The target variable(s) to be written to the file as data. `size%` applies to all targets.

Returns It returns the number of data items for which complete data was written or a negative error code.

Unicode

If you write a string, it is converted to UTF-8 first. Beware that this means that the number of bytes needed to hold the string may be more than the number of characters reported by Len(). There is an example of getting the equivalent UTF-8 string length in the description of `Len()`. If you add 16 to the `mode%` in `FileOpen()`, strings are written using 16-bit characters and the n refers to the count of 16-bit characters stored.

See also:

`FileOpen()`, `BRead()`, `BReadSize()`, `BRWEndian()`, `BWrite()`, Example to write a bitmap

Using BWriteSize() to save a bitmap

This example uses `BWriteSize()` to save a bitmap image in an integer matrix as a `.bmp` file (but please read the next paragraph before using this code). Each element of the bitmap array is an integer with bits 0-7 holding the blue colour, bits 8-15 holding the green and bits 16-23 holding the red and the rest of the data bits set to 0 (`0x00rrggbb`). If you want an Alpha channel (using bits 24-31 to specify opaqueness, with 0 for transparent and `0xff` for opaque, `0xaarrggbb`), add 2 to `flags%`. You can choose to have a bottom to top or top to bottom bitmap.

From Spike2 version [9.02] you can use the `ArrMapImage()` script command to map a real matrix to a bitmap using a colour scale. From Spike2 9.06 you can copy an array to the clipboard as an image with `EditCopy()` and save it as a file in a variety of formats with `EditImageSave()`.

```
'Write a .bmp file based on rgb colours in integer array
'bmpPath$ Path to the file to use. Ideally the name should end in .bmp
'bmp%     A matrix bmp%[nx][ny] with the nx=columns, ny is rows
'flags%   If bit 0 is set, bmp%[0][0] is top left, else bottom left
'         If bit 1 is set, use RGBA coding, else RGB
'return   0 if OK or a negative error code
func WriteBmp(bmpPath$, const bmp%[][], flags% := 0)
var xpix% := Len(bmp%[][0]);            ' pixels in the x direction
var ypix% := Len(bmp%[0][]);            ' pixels in the y direction

var binf% := FileOpen(bmpPath$, 9, 1);  ' open binary output file
if (binf% < 0) then return binf% endif;
```

```
'Write out the bitmap header. This is 14 bytes long.
BWriteSize(2, "BM");                    ' all bitmaps files start 'BM'
BWriteSize(4, xpix%*ypix%*4 + 14 + 108);' Size of .bmp file in bytes
BWriteSize(4, "");                      ' 4 app-defined bytes (0,0,0,0)
BWriteSize(4, 14 + 108);                ' offset to where the data starts

' Construct DIB header (BITMAPV4HEADER - this is 108 bytes long)
BWriteSize(4, 108);                     ' size of the DIB header
BWriteSize(4, xpix%);                   ' Width of the bitmap in pixels
BWriteSize(4, flags% band 1 ? -ypix% : ypix%); ' Height of the bitmap in pixels
BWriteSize(2, 1);                       ' number of color planes. Must be 1
BWriteSize(2, 32);                      ' number of bits per pixel
BWriteSize(4, flags% band 2 ? 3 : 0);   ' BI_BITFIELDS = 3 or BI_RGB = 0. No compression
BWriteSize(4, xpix%*ypix%*4);           ' bytes of bitmap data (or 0 as BI_RGB)
BWriteSize(4, 2835);                    ' horizontal pix/m = 72 pix/inch
BWriteSize(4, 2835);                    ' vertical pix/m = 72 pix/inch
BWriteSize(4, 0);                       ' # of palette colours. No palette used.
BWriteSize(4, 0);                       ' Important palette colours, 0 as unused

' Next 4 32-bit words set the Red, Green, Blue and Alpha mask in BI_BITFIELDS
if (flags% band 2) then
    BWriteSize(4, 0x00ff0000);          ' Red mask    You can swap the RGB
    BWriteSize(4, 0x0000ff00);          ' Green mask  masks around, if you
    BWriteSize(4, 0x000000ff);          ' Blue mask   need other arrangements.
    BWriteSize(4, 0xff000000);          ' Alpha mask  Set 0 if no Alpha channel
else
    BWriteSize(16, "BGRsBGRsBGRsBGRs"); ' Unused in BI_RGB mode
endif
    BWriteSize(52, "");                 ' fill the rest with 0s
' We have no palette, so the next thing in the file is the RGB(A) values
BWriteSize(4, bmp%);                    ' Write the bitmap
FileClose();
return 0;
end
```

C

```
Ceil()
Chan$()
ChanCalibrate()
ChanColour()
ChanColourGet()
ChanColourSet()
ChanComment$()
ChanData()
ChanDecorate()
ChanDelete()
ChanDuplicate()
ChanFit()
ChanFitCoef()
ChanFitShow()
ChanFitValue()
ChanHide()
ChanImage()
ChanIndex()
ChanKey()
ChanKind()
ChanList()
ChanMeasure()
ChanNew()
ChanNumbers()
ChanOffset()
ChanOrder()
ChanPenWidth()
ChanPixel()
ChanPort()
ChanProcessAdd()
ChanProcessArg()
```

ChanProcessClear()
ChanProcessCopy()
ChanProcessInfo()
ChanSave()
ChanScale()
ChanSearch()
ChanSelect()
ChanShow()
ChanTitle$()
ChanUnits$()
ChanValue()
ChanUndelete()
ChanVisible()
ChanWeight()
ChanWriteWave()
Chr$()
Colour()
ColourGet()
ColourSet()
Conditioner commands
CondFeature()
CondFilter()
CondFilterList()
CondFilterType()
CondGain()
CondGainList()
CondGet()
CondOffset()
CondOffsetLimit()
CondRevision$()
CondSet()
CondSourceList()
CondType()
Cos()
Cosh()
Count()
Cursor()
CursorX()
CursorActive()
CursorActiveGet()
CursorDelete()
CursorExists()
CursorLabel()
CursorLabelPos()
CursorNew()
CursorOpen()
CursorRenumber()
CursorSearch()
CursorSet()
CursorValid()
CursorVisible()

Ceil()

Returns the next higher integral number of the real number or array. `Ceil(4.7)` is `5.0`, `Ceil(4)` is `4`. `Ceil(-4.7)` is `-4`.

```
Func Ceil(x|x[]{[]...});
```

`x` A real number or a real array.

Returns 0 or a negative error code for an array or the next higher integral value.

See also:
`Abs()`, `ATan()`, `Cos()`, `Exp()`, `Floor()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`

Chan...()

Chan()

This function converts a string that starts with a single channel specification in the current Time, Result or XY view into a channel number. This deals with memory, virtual and duplicate channels. Channels referred to do not need to exist (except in the case of a duplicate channel specification, like `"2a"` for the first duplicate of channel 2). This allows you to know the channel number of a memory channel `Chan("m1")` or virtual channel `Chan("v1")` before you create it. Use `DupChan(0)` to find the start of the channel range reserved for duplicate channels. This function was added at Spike2 version [8.11].

```
Func Chan(spec${, &index%});
```

`spec$` A string holding a channel number, a memory channel specification, a virtual channel specification or a duplicate channel specification. The specification can be prefixed or followed by white space. If the `index%` argument is omitted, the string must not be followed by any character that could not be part of the channel (apart from white space), so `"m2"`, `" m2"` and `" m2 "` are fine, but `"m2,"` is not.

`index%` An optional integer variable that is returned holding the index into `spec$` at which the first character was found that was not part of a channel specification or -1 if all characters were used. If this is omitted, all characters of `spec$` must be used when decoding the channel number. The idea it to allow you to parse an arbitrary string that starts with a channel specification. If the string held `" m2 ,"`, `index%` would be returned holding 3 as the space was the first character that could not be part of the channel specification.

Returns The single argument version either returns a channel number or a fatal error code. The two argument version returns a channel number or 0 if the channel was not recognised; in either case `index%` is -1 if all characters were used or it is the index of the first character that could not be part of a channel specification.

Unlike `ChanList()`, this routine does not care if the channel exists or not, except in the case of duplicate channels (where a channel number cannot be predicted from the source channel and the duplicate code). This allows you to use this to find out which channel a particular memory or virtual channel would use if it existed. Note that version 9 onwards of Spike2 uses different channel numbers for memory, virtual and duplicate channels from version 8.

See also:
`Chan$(), ChanList(), DupChan()`

Chan$()

This function converts a channel number or a list of channel numbers into a string. Memory, duplicate and virtual channels are listed as they appear on screen (not just as numbers). If a channel does not exist in the current view, it is represented as a number.

```
Func Chan$(chan%|const chan%[]);
```

`chan%` Either a channel number or an array of integers in the same format as a channel specifier (the first element holds the number of items, followed by the channel numbers).

Returns A channel specification string, for example `"1,3,5..8,m2,v1a"`.

See also:
`ChanList()`

ChanCalibrate()

This function is equivalent to the Analysis menu Calibrate command. It changes the scale and offset of Waveform and WaveMark channels and rewrites RealWave channels so that user-defined data sections have user-defined values. The command is:

```
Func ChanCalibrate(cSpc,mode%,cfg%,t1,t2,units$,v1{,v2{,t3,t4}});
```

`cSpc` A channel specifier for the channels to calibrate. You cannot calibrate processed channels if the process changes the scale or offset or any processed RealWave channel.

`mode%` The calibration mode. The items in brackets are the required optional arguments.

- 0 Mean levels of two time ranges (`v2`, `t3`, `t4`).
- 1 Values at two time points (`v2`).
- 2 Set offset from mean of time range.
- 3 Set scale from mean of time range.
- 4 Square wave, upper and lower level (`v2`).
- 5 Square wave, amplitude (Size) only.
- 6 Peak to peak amplitude and mean (`v2`).
- 7 RMS amplitude about mean (`v2`).
- 8 Area under curve, assume zero at end.
- 9 Areas under curve, two time ranges (`v2`, `t3`, `t4`).
- 10 Slope of best-fit line to the data, the offset not changed.

`cfg%` If non-zero, the new calibration is saved in the sampling configuration.

`t1,t2` In all modes except mode 1, these are the start and end of the first time range. In mode 1 these times correspond to the two calibration values `v1` and `v2`.

`units$` The units to apply to the channel.

`v1,v2` The values in user units that correspond to the times or time ranges. `v1` is always used; `v2` is only used in modes 0, 1, 4, 6, 7 and 9.

`t3,t4` A second time range. These values are used in modes 0 and 9.

Returns The return value is the sum of the following values:

- 1 A channel in the list did not exist or was the wrong type.
- 2 A channel was processed OK.
- 4 Unknown or unimplemented calibration mode.
- 8 A time range had the end past the start or two time ranges overlapped or the two times in mode 1 were the same.
- 16 The `v1` and/or `v2` values were too big, too small or too similar.
- 32 Not enough data to process at least one channel in the list.
- 64 The data is unsuitable. For example, in mode 0 mean levels must differ by at least the standard deviation of the data around the mean.

See also:
Channel specifiers, `ChanOffset()`, `ChanScale()`, `ChanUnits$()`

ChanColour()

Deprecated. This returns and optionally sets the colour of a channel in a time or result view to a palette colour. This colour overrides the application colour set for the drawing mode of the channel. This function is provided for backwards compatibility and should only be used when a script must work with older versions of Spike2. Use `ChanColourGet()` and `ChanColourSet()` instead to set colours as RGB combinations.

```
Func ChanColour(chan%, item%{, col%});
```

`chan%` A channel in the time or result view.

`item%` The colour item to get or set; 0=background, 1=primary, 2=secondary colour.

`col%` If present, the new colour index for the item. There are 40 colours in the palette, indexed 0 to 39. Use -1 to revert to the application colour for the drawing mode.

Returns The palette colour index that is nearest to the item colour at the time of the call, -1 if no colour is set or a negative error code if the channel does not exist. **Beware**: if you set a colour index for an item and read it back you may not get the index you set if another index holds the same colour.

See also:
Colour dialog, `ChanColourGet()`, `ChanColourSet()`, `Colour()`, `PaletteGet()`, `PaletteSet()`, `ViewColour()`, `ViewUseColour()`, `XYColour()`

ChanColourGet()

Returns if the background, primary or secondary colour of a channel is overridden from the standard colour and returns the overridden RGB colour in a time, result or XY view. This was added at Spike2 version 7.07.

```
Func ChanColourGet(chan%, item%{, &r, &g, &b});
```

`chan%` A channel in the time, result or XY view.

`item%` The colour item to get; 0=background, 1=primary, 2=secondary colour for time and result views, the fill colour for an XY channel. In an XY view, for `item%` of 0, the channel number is ignored as the background is shared by all channels.

`r g b` If present, these variables are returned holding the colour as red, green and blue values in the range 0 to 1.0. If the colour is not overridden, these values will be 0.

Returns 1 if the channel colour is overridden, 0 if not, negative for no channel. If the channel colour is not overridden you can use `ColourGet()` to get the colours of particular items and `ViewColourGet()` to get the background view colour.

See also:
Colour dialog, `ChanColourSet()`,`ColourGet()`,`ViewColourGet()`

ChanColourSet()

Sets or clears the override of channel item RGB colours in a time, result or XY view. A change to a colour item that would make visible difference will cause the affected view to become invalid and it will repaint at the next opportunity. This was added at Spike2 version 7.07.

```
Func ChanColourSet(cSpc, item%{, r, g, b});
```

`cSpc` From version [10.02] onwards, this is a channel specifier that selects one or more channels in the time, result or XY view. Prior to this, this was `chan%`, a channel number. This is ignored in an XY view when `item%` is 0 (and can be set to 0). The change at 10.02 allows you to set colour overrides for several channels with one command.

`item%` The colour item to set; 0=background, 1=primary, 2=secondary colour for time and result views, the fill colour for an XY channel. In an XY view, for `item%` of 0, the channel number specifier is ignored as the background is shared by all channels.

`r g b` If these are all present, they set the override colour as red, green and blue values in the range 0 to 1.0. If any are absent, the colour override for the item in the channel is removed.

Returns 0 or a negative error code if the channel does not exist.

See also:
Colour dialog, `ChanColourGet()`,`ColourSet()`,`ViewColourSet()`

ChanComment$()

This returns or sets the comment string for a channel in a time or result view. It is an error to use this in any other view type. Channel comments are limited in length when they are written to data files (to 71 8-bit characters for a 32-bit smr file and to 2000 (was 100 before Spike2 8.09) Unicode characters for a 64-bit smrx file). If you set a comment of more than 100 characters and the user edits the comment with a version of Spike2 before 8.09, the comment will be truncated to 100 characters.

```
Func ChanComment$(chan%{, new$});
```

`chan%` A channel in the time or result view. In a result view, if the channel is a duplicate, the comment is returned from or set in the source channel.

`new$` An optional string with a new comment.

Returns It returns the comment string for the designated channel. If the channel does not exist, the function does nothing and returns an empty string.

See also:
`ChanTitle$(), FileComment$()`

ChanData()

Fills an array with waveform data from a waveform or RealWave channel or with event times from all other channel types. Use `NextTime()` and `LastTime()` to get data attached to WaveMark, RealMark and TextMark channels and marker codes. If you want the number of events between two times you should use `Count()`. You can also use `ChanMeasure()` to calculate commonly required values between two times. There are several command variants for use with different channel types. All the variants that support the `code%` argument were added at version 8.04.

```
Func ChanData(chan%, wave[]|t[], sTime, eTime{, &fTime});
Func ChanData(chan%, t[], sTime, eTime, code%[]{[]});
Func ChanData(chan%, t[], sTime, eTime, code%[]{[]}, data$[]);
Func ChanData(chan%, t[], sTime, eTime, code%[]{[]}, data[]{, index%});
Func ChanData(chan%, t[], sTime, eTime, code%[]{[]}, data[][]{, trace%});
Func ChanData(chan%, t[], sTime, eTime, code%[]{[]}, data[][][]);
```

`chan%` The data channel in the current time view to read data from.

`wave` A real or integer array. Real arrays collect waveform data in user units and event times in seconds. Integer arrays collect waveforms in ADC units and event times in the underlying time units (as returned by `Binsize()`). RealWave data is converted to integers using the channel scale and offset.

`t` A real or integer array used to collect times from Marker or extended Marker (TextMark, RealMark, WaveMark) channels. Real arrays get the time in seconds, integer arrays get the time in the underlying time units.

`sTime` The first data value returned is at or after this time in seconds.

`eTime` The last data point returned is before or at this time in seconds.

`fTime` This optional argument is a variable that is set to the time of the first data point.

`code%` Used with Marker-based channels to return the marker codes associated with each time. If this is a vector, the 4 marker codes are returned with the first code in the bits 0-7, the seconds in bits 8-15, the third in bits 16-23 and the fourth in bits 24-31. Bits 32-63 return the 32-bits of marker codes that are reserved for future use in 64-bit `.smrx` data files. These are currently 0 for sampled data but may be non-zero for imported data. If this is a matrix, the first code is in `code%[0][]`, the second in `code%[1][]` and so on up to `code%[3][]`. If the first dimension is larger than 4, `code%[4][]` is set to the (currently) spare 32-bits of marker code.

`data$` A vector of strings used with TextMark channels to return the text associated with each time.

`data` For RealMark channels this is either a real vector (to return a single point per channel) or a real matrix (to return multiple points per channel). If a matrix is used, the first dimension sets the number of points per channel to return, the second is the maximum number of RealMark data items to read.

For WaveMark data, this is either a 2D array (to return one trace per event) or a 3D array to return multi-trace data. The first dimension is the maximum number of points per trace to return, the second (for multi-trace data) is the number of traces and the last is the maximum number of events. If two dimensions are used with multi-trace data, the trace set by `trace%` is returned. This can be either an integer array (to get the data points as integer ADC values) or a real array to get the scaled into user units using the channel scale and offset values.

`index%` Used with RealMark channels to set the index of the item that is to be returned (the first is 0) when `data` is a vector.

`trace%` Used with multi-trace WaveMark data when `data` is a matrix to select the trace (0 for the first) to return. If omitted, the first trace is used.

Returns The number of data values placed in the array or a negative error code if the channel type does not match the supplied arguments. Only contiguous waveform data is returned; gaps terminate a read. When reading back Marker, RealMark, TextMark or WaveMark data, the maximum number of events is set by the smallest final dimension of the `t`, `code%` or `data` arrays.

Event channels

```
Func ChanData(chan%, t[], sTime, eTime{, &fTime});
```

You can use this variant with an event channel (or any Marker or extended Marker channel) to get the times of items ignoring any attached marker data. The `fTime` argument is usually not used as it is returned set to the same value as the first element of `t`. The following example reads events times as both seconds and then as ticks:

```
const nMax% := 1000;
var s[nMax%], t%[nMax%];       'Times in seconds and ticks
var n% := ChanData(3, s, 0, MaxTime());   'Read in seconds
var m% := ChanData(3, t%, 0, MaxTime());  'Read as integer ticks
```

Waveform and RealWave channels

```
Func ChanData(chan%, wave[], sTime, eTime{, &fTime});
```

This variant collects contiguous waveform data from a channel (that is data without a gap). Each read is terminated by either a gap, by filling the `wave[]` array or by reaching the end of the file. The time of the first read point (in seconds) is returned in (the optional) `fTime`. The `wave[]` array can be either real to get data in user units or integer to get data in ADC units. Waveform data is stored on disk as 16-bit integers that is scaled into user units by a scale and offset. RealWave data is stored on disk as 32-bit floating point data, but this can be scaled into the equivalent ADC units (which will usually lose some accuracy). The following example reads waveforms as both real and integer values:

```
const nMax% := 1000;
var wave[nMax%], adc%[nMax%], first; 'Real and integer wave, first time
var n% := ChanData(1, wave, 0, MaxTime(), first);   'Read in user units
var m% := ChanData(3, adc%, 0, MaxTime(), first);   'Read as integers
```

Marker data

```
Func ChanData(chan%, t[], sTime, eTime, code%[]{[]});
```

Marker data, or any extended marker type can be read with this command variant to get the marker codes associated with each event (ignoring any other attached data).

The `code%` array can either be a vector (for example `c%[100]`), in which case the 4 marker codes and the reserved 32-bit value associated with each marker are packed into each element as described above, or it can be a matrix (for example `c%[4][100]`) when the first dimension sets the number of codes to return and the second sets the number of markers to read. In the matrix variant, if the first dimension is less than 4 elements, then fewer codes are returned. If you only wanted the first code, you could use `c%[1][100]`. If the first dimension is set to 5 or more, elements `c%[4][]` are returned with the extra 32-bits of data that can be stored in a 64-bit marker. This extra data will be 0 for data recorded with Spike2, and will always be 0 for data read from a 32-bit smr file (as there is no extra data).

The maximum number of markers that will be returned is the smaller of the sizes of the t and the last dimension of the code% arrays. The following example reads Marker codes both as a combined value and as separate codes.

```
const nMax% := 1000;
var s[nMax%], c%[nMax%], c2%[4][nMax%]; 'Times, combined, separate
var n% := ChanData(5, s, 0, MaxTime(), c%);   'Read combined codes
var m% := ChanData(3, s, 0, MaxTime(), c2%);  'Read as separate codes
```

TextMark data

```
Func ChanData(chan%, t[], sTime, eTime, code%[]{[]}, data$[]);
```

This variant is used with TextMark channels to read the marker information and the text strings. The t and code% arguments are the same as for the Marker data variant. The maximum number of returned items is set by the smallest of the t, code% and data$ arrays. There is no way to not return the code% information, so if you don't want it, the most efficient thing to do is to provide an integer vector to hold the codes. This example reads up to 1000 text strings and prints the times, codes and text to the log window:

```
const nMax% := 1000;
var s[nMax%], c%[nMax%], t$[nMax%], i%;      'Times, codes, text
var n% := ChanData(3, s, 0, MaxTime(), c%, t$);   'Read the text
printlog("\n  Seconds     Codes Text\n");
for i% := 0 to n%-1 do
    printlog("%9.5f: %08x %s\n", s[i%], c%[i%], t$[i%]);
next;
```

RealMark data

```
Func ChanData(chan%, t[], sTime, eTime, code%[]{[]}, data[][]);
Func ChanData(chan%, t[], sTime, eTime, code%[]{[]}, data[]{, index});
```

These two variants are used to read RealMark data. If your RealMark channel has only one attached data point, or if you only want one data point returned per item, you can use the variant with data[] as a vector. Otherwise, use the variant with a matrix and the first dimension of data sets the maximum number of attached values to return and the second dimension is the maximum number of RealMark items to read. The maximum number of RealMark items returned is set by the smallest last dimension of the t, code% and data arrays. The example code reads data from a RealMark channel assumed to have 3 attached values per marker:

```
const nMax% := 1000;
var s[nMax%], c%[nMax%], v[nMax%], vAll[3][nMax%]; 'Times, codes, one, all values
var n% := ChanData(4, s, 0, MaxTime(), c%, vAll);  'Read all the values
var m% := ChanData(4, s, 0, MaxTime(), c%, v, 1);  'Read the second value
```

WaveMark data

```
Func ChanData(chan%, t[], sTime, eTime, code%[]{[]}, data[][]{, trace%});
Func ChanData(chan%, t[], sTime, eTime, code%[]{[]}, data[][]{[]});
```

These variants read WaveMark data. The t and code% arguments are the same as for the Marker data case. The data array is either two-dimensional to return data with a single trace or the trace set by trace% of data with multiple traces, or three-dimensional to return multi-trace data. The first dimension is the maximum number of data points per trace to read, the last dimension is the maximum number of WaveMark items to read. For a 3D array, the second dimension is the maximum number of traces to read. The maximum number of WaveMark items returned is set by the smallest last dimension of the t, code% and data arrays. The example code reads data from a WaveMark channel assumed to have 4 traces and 32 points per trace:

```
const nMax% := 1000;
var s[nMax%], c%[nMax%], w[32][4][nMax%];       'times, codes, all data
var n% := ChanData(6, s, 0, MaxTime(), c%, w);  'Read all 4 traces
var t0[32][nMax%], t1[32][nMax%];               'Space for single traces
var m% := ChanData(6, s, 0, MaxTime(), c%, t0); 'Read first trace only
m% := ChanData(6, s, 0, MaxTime(), c%, t1, 1);  'Read second trace
var adc%[32][2][nMax%];                         'Space for 2 traces as integers
m% := ChanData(6, s, 0, MaxTime(), c%, adc%);   'Read first 2 traces
```

Using ChanData()

You will often use ChanData() iteratively to work your way through a channel of data. You can find examples of this in the BWrite(), Count() and ChanNew() command descriptions and for several other commands if

you search this help for ChanData. The example for BWrite() is particularly helpful as it shows how to iterate through waveform and event data collecting the data as either real or integer numbers. When we return data in an array we only fill in array elements that correspond to data in the file; we do not set unfilled array elements to zero. If you use arrays that have more elements than are required for the data, these extra elements are not modified.

Backwards compatibility

Event times read as integers from 64-bit smrx files are 64-bit values. If you are migrating from earlier versions of Spike2 this should not cause you any problems. However, you should bear this in mind when writing scripts that must also run on earlier versions of Spike2 where all times are limited to 32-bit integer values. The command variants that read Marker and extended Marker data were added at Spike2 version [8.04].

See also:

Binsize(), ChanMeasure(), ChanScale(), ChanValue(), ChanWriteWave(), Count(), NextTime()

ChanDecorate()

This command is equivalent to the Channel Decoration dialog. It currently is used with RealMark data channels in a time view, but may be extended in the future to cover other channel types.

```
Func ChanDecorate(chan%, mode%{, flags%, ind1%{, ind2%}});
```

chan% A channel in the time view. This need not be a RealMark channel, but has no visible effect on other channel types.

mode% The channel decoration mode: 0=none, 1=error bars, 2=low-high range. Negative values of mode% return information and do not change the decoration mode: -1 = mode%, -2= flags%, -3=ind1%, -4=ind2%.

flags% This is the sum of:

- 1 The bar is drawn in the marker colour (otherwise in the channel secondary colour).
- 2 Draw a centre circle at the channel value.

ind1% The 0-based index of the data item in the RealMark channel to use as the error value in mode 1 and the low value in mode 2. If this argument is omitted, no change is made to the index. If you set an index that does not exist, no decoration is drawn.

ind2% The 0-based index of the data item in the RealMark channel to use as the high value when drawing a range bar. If omitted, no change is made. If you set an index that does not exist, no decoration is drawn.

Return Unless mode% is negative, the command returns the channel decoration mode at the time of the call. Otherwise, it returns the information requested by mode%.

Change

Before versions [10.02] and [9.10] it was not possible to read back the mode% value without setting it. Previously, the negative mode% values used to read the state were: -1= flags%, -2=ind1%, -3=ind2%. This change was made to be compatible with Signal and to make the argument read-back more logical. As no-one had complained about this we suspect that few, if any, scripts (apart from our test scripts) use the read back, so changing it should be painless.

See also:

DrawMode()

ChanDelete()

This deletes a channel from a time, result or XY view. You can make the user confirm time view channel deletion if the channel is stored in the file. Duplicated channels, and memory buffer channels are not confirmed. In a result view, you may only delete duplicate channels. In an XY view you cannot delete the last XY channel

as XY views must always have at least one channel. Channels are always numbered consecutively in an XY view, so if you delete a channel, the channel numbers of any higher numbered channels will change.

```
Func ChanDelete(cSpc {,query%});
```

`cSpc` A channel specifier for the channels to delete. In an XY view only channel numbers greater than 0 are allowed.

`query%` If present and non-zero, the user is asked to confirm the deletion if the channel is part of a time view. You cannot delete channels that are being sampled.

Returns 0 if the channel was deleted or a negative error code if the user cancelled the operation or tried to delete the last XY channel or for other problems. You cannot delete a time view sampled channel while sampling is in progress. From [10.01] you are allowed to delete a sampled channel once sampling has stopped.

Data files and maximum file time

In a data file, the `MaxTime()` command returns the maximum time in any channel. The data file holds the maximum time in any channel written to disk, but this value is not updated when you delete a channel. It can happen that you write data to the disk file at some large time on a channel, then delete this channel. Spike2 will still think that the file extends to the original time. From [11.00] you can use the `MaxTime(-1)` command to reassess the maximum time.

See also:

Channel specifiers, `ChanDuplicate()`, `ChanUndelete()`, `MemChan()`, `XYDelete()`, `XYSetChan()`

ChanDuplicate()

This duplicates a time or result view channel. It can also be used in the Edit WaveMark view to delete all duplicates of the current channel and generate duplicates with suitable marker filter setting for each template. Use `DupChan()` to find duplicates of a channel. There are two command variants:

Duplicate Time or Result view channel

This duplicates a channel in a Time or Result view.

```
Func ChanDuplicate({chan%{, flags%});
```

`chan%` The channel number to duplicate; this channel must exist. You are allowed up to 52 duplicates of one channel.

`flags%` This arguments was added at version [10.17] and has the value 0 if omitted. It is the sum of:

1 Copy the visibility of `chan%` to the duplicate. Without this, the new channel is hidden.

2 Delete all duplicates of `chan%` before creating a duplicate. Without this, existing duplicates are unaffected. If `chan%` is a duplicate no channels are deleted.

Returns The channel number of the duplicate or a negative error code (usually for too many duplicates).

The duplicated channel is not displayed unless `flags%` has 1 added. You can use `ChanShow()` after the call to make it visible. The duplicated channel inherits the drawing mode, decoration mode, axis ranges and colour overrides of `chan%`.

The following example duplicates a time or result view channel and makes it visible:

```
var ch%;ch% := ChanDuplicate(1);  'create a duplicate
ChanShow(ch%);                    'make visible
```

Make filtered WaveMark channels

This is used when an Edit WaveMark dialog is current and the current channel is not a duplicate. It is equivalent to the Duplicate button in the dialog. This deletes all duplicates of the channel and creates a duplicate data channel in the time view for each template code set in the dialog. Each created channel has the channel Marker Filter set to display one of the templates. The duplicated channels are made visible.

```
Func ChanDuplicate()
```

Returns The number of duplicates created or -1 if the current channel is a duplicate.

Channel titles

Each duplicated channel inherits the title from the original, but can store its own title. If you set the duplicated channel title, this overrides the original. To revert to the original title, set the duplicated channel title to an empty string. The channel units and comment are always the same as the original channel.

See also:

`ChanShow(), ChanDelete(), DupChan(), MemChan()`

ChanFit()

This function together with `ChanFitCoef()` and `ChanFitShow()` incorporates the functionality of the Analysis menu Fit Data dialog. `ChanFit()` has three variants that: initialise ready for a new fit, perform the fit and return information about the last fit. The current window must be a time, result or XY view to use these functions. You can use the `FitData()` command to fit to data in an array.

Initialise fit information

This command associates a fit with a channel. The fit parameters and the coefficient limits are reset to their default values, the coefficient hold flags are cleared and any existing fit for this channel is removed.

```
Func ChanFit(chan%, type%, order%);
```

`chan%` The channel number to work on. Each channel in a time, result or XY view can have one fit associated with it. From version 8.03 you can fit to a WaveMark channel drawn as a waveform.

`type%` 0=Clear fit, 1=Exponential, 2=Polynomial, 3=Gaussian, 4=Sine, 5=Sigmoid.

`order%` The order of the fit. This is 1 or 2 for an exponential or Gaussian fit, 1 for a Sine or Sigmoid fit and 1 to 5 for a polynomial fit. If `type%` is 0 this should also be 0.

Returns 0 if the command succeeded.

Perform the fit

This variant of the command does the fit set by the previous variant.

```
Func ChanFit(chan%, opt%, start|start$, end|end${, ref|ref${, &err{, maxI%
{,&iTer%{, covar[][]}}}}});
```

`chan%` A channel number in the current view that has had a fit initialised.

`opt%` This is the sum of:

- 1 Estimate the coefficients before fitting, else use current values.
- 2 Display the fitted data. Use `ChanFitShow()` to change the displayed range.

`start` This is the start of the fit range as a value or as a dialog expression string that is to be evaluated.

`end` The end of the fit range in x axis units as a value or a string to evaluate.

`ref` The reference time as a value or a string to evaluate. If omitted start is used.

`err` If present, this optional variable is updated with the chi-squared or least-squares error between the fit and the data.

`maxI%` If present, this sets the maximum number of iterations. If omitted, the current number set for the channel is used. The system default number is 100.

`iTer%` If present, this integer variable is updated with the count of iterations done.

`covar` An optional two dimensional array of size at least `[nCoef][nCoef]` that is returned holding the covariance matrix when the fit is complete. It is changed if the return value is -1, 0 or 1. However, the values it contains are probably not useful unless the return value is 0.

Returns 0 if the fit is complete, 1 if `maxI%` iterations done, or a negative error code: -1=the fit is not making progress (results may be OK), -2=the fit failed due to a singular matrix, -5=the fit caused a floating point error, -6=too little data for the number of coefficients, -7=unknown fitting function, -8=ran out of memory during the fit (too many data points), -9=the fit was set up with bad parameters.

Get fit information

This variant of the command returns information about the current fit set for a channel.

```
Func ChanFit(chan%{, opt%});
```

`chan%` The channel number of the fit to return information about.

`opt%` This determines what information to return. If omitted, the default value is 0. Positive values return information about the fit that is set-up to be done next. Negative values return information about the last fit that was done and that can be displayed. The returned information for each value of `opt%` is:

opt%	Returns	opt%	Returns
0	Fit type of next fit	1	Fit order of next fit
-1	1=a fit exists, 0=no fit exists	-9	User-defined x draw start
-2	Type of existing fit or 0	-10	User-defined x draw end
-3	Order of existing fit	-11	1=chi-square, 0=least-square
-4	Chi or least-squares error	-12	Last fit result code
-5	Fit probability (estimated)	-13	Number of fitted points
-6	X axis value at fit start	-14	Number of fit iterations used
-7	X axis value at fit end	-15	R-square value
-8	Reference x value	-16	Adjusted R-square value

Returns The information requested by the `opt%` argument or 0 if `opt%` is out of range.

See also:

Fit Data dialog, More about fitting, Dialog expressions, `ChanFitCoef()`, `ChanFitShow()`, `ChanFitValue()`, `FitExp()`, `FitPoly()`

ChanFitCoef()

This command gives you access to the fit coefficients for a channel in the current time, result or XY view. You can return the values from any type of fit and set the initial values and limits and fix values for iterative fits. There are two command variants:

Set and get coefficients

This command variant lets you read back the current coefficient values and set the coefficient values and limits for iterative fitting:

```
Func ChanFitCoef(chan%{, num%{, new{, lower{, upper}}}});
```

`chan%` The channel number of the fit to access.

`num%` If omitted, the return value is the number of coefficients in the current fit. If present, it is a coefficient number. The first coefficient is number 0 and the coefficients are numbered to match their order in the Fit Data dialog. With `num%` present, the return value is the coefficient value of the current fit, or if there is no fit, the coefficient value that would be used as the starting point for the next iterative fit.

`new` If present, this sets the value of coefficient `num%` for the next iterative fit on this channel.

`lower` If present, this sets the lower limit for coefficient `num%` for the next iterative fit on this channel. There is currently no way to read back the coefficient limits. There is no check that the limits are valid.

`upper` If present, this sets the upper limit for coefficient `num%` for the next iterative fit on this channel.

Returns The number of coefficients or the value of coefficient `num%`.

Get and set the hold flags

This variant sets the hold flags (equivalent to the Hold check boxes in the Fit Data dialog Coefficients tab).

```
Func ChanFitCoef(chan%, hold%[]);
```

`chan%` The channel number of the fit to access.

`hold%` An array of integers to correspond with the coefficients. If the array is too long, extra elements are ignored. If it is too short, extra coefficients are not affected. Set `hold%[i%]` to 1 to hold coefficient `i`

% and to 0 to fit it. If hold%[i%] is less than 0, the hold state is not changed, but hold%[i%] is set to 1 if the corresponding coefficient is held and to 0 if it is not held. Because hold%[] could be modified, it may not be a const array.

Returns This always returns 0.

See also:
ChanFit(), ChanFitShow(), ChanFitValue(), FitExp(), FitPoly()

ChanFitShow()

This controls the display of data fitted to a channel in the current time, result or XY view.

```
Func ChanFitShow(chan%{, opt%{, start|start${, end|end$}});
```

chan% The channel number of the fit to access.

opt% If present and positive, this is the sum of:

1 Display the fitted data (if not added, any fit on the channel is hidden).

2 Use the user-defined display range rather than the fitting range.

If opt% is omitted or positive, the return value is the current option value. Use negative values to return the user-defined display range: -1=return the start, -2=return the end.

start If present, this is an x axis value or a string holding a dialog expression to be interpreted as an x axis value that sets the start of the user-defined display range.

end If present, it sets the end of the user-defined display range as start sets the start. The allowed range is limited to 4 times the width of the fitted data before and after the fitted data.

Returns The current opt% value or the information requested by opt%. If there is no fit defined for the channel, the return value is 0.

See also:
Dialog expressions, ChanFit(), ChanFitValue(), ChanFitCoef(), FitExp(), FitPoly()

ChanFitValue()

This function returns the value at a particular x axis value of the fitted function to a channel in the current time, result or XY view.

```
Func ChanFitValue(chan%, x{, &got%});
```

chan% The channel number of the fit to access.

x The x axis value at which to evaluate the current fit. Some fitting functions can overflow floating point range if x is outside the fitted range of the function.

got% If present, this is returned set to 1 if there was a fit on the channel and we got a value, 0 if not. This argument was added at Spike2 version [9.03].

Returns The value of the fitted function at x. If the result is out of floating point range, the function may return a floating point infinity or a NaN (Not a Number) value or a 0. If there is no fit, the result is 0.

See also:
ChanFit(), ChanFitShow(), ChanFitCoef(), FitExp(), FitPoly()

ChanHide()

Hide a channel, or a list of channels. Hiding a channel that doesn't exist has no effect.

```
Proc ChanHide(cSpc {,cSpc...});
```

cSpc A channel specifier for the channels to hide.

See also:
Channel specifiers, ChanShow(), ChanList(), ChanVisible(), View menu show/hide

ChanImage()

You can set a background image behind each channel in a Time, Result or XY view. Images are read from .bmp, .jpg, .jpeg, .gif, .png, .tif or .tiff files on disk. See the View menu Channel Image command for details. There are three command variants:

```
Func ChanImage(chan%, path$);
Func ChanImage(chan%, mode%, opac{, xl, yl, xh, yh});
Func ChanImage(chan%, get%{, &path$});
```

chan% A channel number in the current view. In an XY view, all channels share the same bitmap.

path$ The file name to the image. The first command variant sets the image, the third variant reads back the file name. Setting an empty name releases any memory used to cache the bitmap within the program. Setting an image does not change the display mode; use the second variant to make sure that the image is visible. Set the path to "<CB>" to use whatever image happens to be on the clipboard.

mode% There are three display modes that you can set in the second command variant: 0=no display, 1=fill background, 2=fill rectangle. You can also set mode -1, meaning no mode change.

opac You can control the image opacity in the range 0.0 (transparent) to 1.0 (opaque). You can also set the value -1 for no change.

xl-yh When mode% is 2, these four arguments set the rectangle, in x and y axis units, that contains the bitmap image.

get% This is used in the third command variant to read back the current settings. You can set -1 to read the mode, -2 to read the opacity, and -3 to -6 to read the xl, yl, xh and yh values.

Returns The first variant returns 1 if a bitmap was read, 0 if it was not (either file not found, or no image was set) or a negative error code. The second returns 0, the third returns the requested information.

We do not provide any method to size a channel area so a bitmap becomes a specific size. However, you can use ChanPixel() to find the scaling between x and y axes and pixels. It is possible to then iteratively change the window size (or adjust the channel weight with ChanWeight()) to get the desired effect.

See also:
ChanColour(), ChanPixel(), ChanWeight(), ArrMapImage()

ChanIndex()

Get or set the index associated with a channel in a time view. RealMark channels use the index to select the data value to display. You can set this value interactively with the Data index field of the Draw Mode dialog.

```
Func ChanIndex(chan%{, index%});
```

chan% A channel number. Currently, this is useful only with RealMark channels.

index% If present and positive, this sets the zero-based channel index. If omitted or -1, the command returns the current index. If -2, the command returns the number of index values for the channel.

Returns Either the channel index at the time of the call, or if index% is -2, the number of possible index values.

See also:
MarkInfo()

ChanKey()

Channel keys display addition information about a channel or the view. There are currently three types of key:

Type	View kind	Description
`XY`	XY	Identifies channels in an XY view
`Sono`	Time	Used with a waveform drawn in sonogram mode to indicate intensity
`ColS`	Any	Used with background bitmaps from `Spline2D()` to show z values as a colour scale. This type is created by scripts, not interactively. New in Spike2 version [9.06]. Once created, the key persists until the next `ViewStandard()` call or use of the View menu Standard Display command.

This command is used to control keys in a generic way; there is also the `XYKey()` command that works only with XY keys. There are four command variants:

Create or modify a key

The first variant creates a key or modifies an existing key. The remaining commands will not work unless the key already exists.

```
Func ChanKey(chan%, type$, flags%{, x, y{, s{, w, h}}});
```

`chan%` This must be set to a valid channel number in the current view for time and result views. In an XY view this should be set to 0 for the XY key and we suggest -1 for the colour scale key. Keys are identified by the channel number, so we assume that only 1 key exists at a time for each channel. XY views can have both an XY key and a colour scale key active, as long as they have different channel numbers.

`type$` The type of the key to create. This is case insensitive and we currently support `"Sono"`, `"XY"` and `"ColS"`. If you set "Sono", the current view must hold a channel drawn in Sonogram mode.

`flags%` The sum of a set of values that control the appearance (or non-appearance) of a key. The flag values were chosen to make `flags%` set to 1 generate a useful result. The value is the sum of:

Value	Meaning
1	Make the key visible. If not added, the key is hidden.
2	Controls the key background. In an XY view, add for a transparent key, omit for the view background. For Sonograms or a colour scale key, add for the view background colour, omit for the colour at the left hand end of the colour scale.
4	Add to draw a border, omit for no border.
8	Sets the orientation of sonogram and colour scale keys. Add for vertical key, omit for horizontal.
16	Sets the position of the key. Add for view-relative, omit for channel-relative.
32	Used with vertical sonogram and colour scale keys. Add to draw axis numbers on the other side of the axis (added at Spike2 version [9.06]).
64	Ignores the key size when calculating the key position. Setting this flag allows you to position the top left corner anywhere in the visible area (so the right size and bottom edge of the key can go off the visible area). Added at [10.10] for compatibility with an old script, not recommended for general use.

From Spike2 version [9.06], you can use `ChanKey(chan%, 0, flags%)` to set the `flags%` of an existing key. Previously the only way to set the flags from a script was to recreate the key.

`x,y` The key position as a percentage of the available space (space less the key size unless 64 is added to `flags%`). The x space runs from left to right, and is the space used by the x axis. The y space runs from top to bottom and if `flags%` has 16 added it is the entire view height, otherwise it is the channel height. The scaling is arranged so that 0,0 has the key at the top left and 100,100 has the key at the bottom right. If these arguments are omitted they are set to 0, 0 (top left).

`s` The font size in points to use for text in the key and for scaling the size of the key. If omitted or set to 0, a default value is used that is based on the size of the text in the view that the key belongs to. You

can request any size you like, but there is a limit on the maximum and minimum font sizes that will be used. Negative sizes are reserved and currently have no effect.

`w,h` These are not currently used. They are reserved for future channel keys where the size is not calculated from the key content.

Returns The `flags%` value before it was modified by the call or -2 if the key type is unknown or incompatible with the view or -1 if this is an XY key and the channel is other than 0.

Modify the flags

The second variant sets the flags, but does not change any other settings, and returns the previous `flags%` value. You can read back the flags with `ChanKey(chan%, -2)`.

```
Func ChanKey(chan%, 0{, flags%});
```

`chan%` The channel number, as above.

`flags%` The flag values, as above. If omitted it defaults to 0, which hides the key. Negative values of `flags%` have no effect.

Returns The `flags%` value before it was modified by the call or -3 if there was no key.

Set key-specific values

The three key types (Sonogram, XY and Colour Scale) have specific parameters that can be set. Setting a value that a key does not know about has no effect. You can only set a value when the key is visible. Note that `ChanKey(chan%, 0, value)` is the modify the flags variant, above.

```
Func ChanKey(chan%, set%, value{, value$});
```

`chan%` The channel number, as above.

`set%` Identifies the parameter to set. The parameters are numbered from 1 upwards. Currently defined values are:

Key	#	Use
Sonogram	1	Sets the y axis value that corresponds to 0 dB in the sonogram key. If you set 0 or a negative value, the 0 dB point is set to the default, the equivalent of a 1 bit change of a Waveform channel. For reference, 1 bit is: `Abs(ChanScale(chan%)/6553.6)` we need the `Abs()` in case the scale is negative; the offset value is not relevant as a 1 bit change is a difference of two values and any offset is cancelled out.
XY	1	Set non-zero to hide the example trace drawn in the XY key for each channel. Set 0 to show it.
Colour scale	1	Set the colour scale number to display. If you do not set a colour scale, scale 0 (Monochrome) is selected.
	2	Set the colour scale low limit for labelling the axis. If you do not set this value, 0.0 is used. There is space for 4 digits in the axis.
	3	Set the colour scale high limit for labelling the axis. If you do not set this value, 1.0 is used. There is space for 4 digits in the axis.
	4	Set the units of the axis with `value$`. This is blank if you do not set it. There is space for 4 or five characters.

Negative values of `set%` have no effect. See Modify the flags, above, for `set%` value of 0.

`value` A value to set as determined by `set%`. If you are setting a string with `value$`, `value` is ignored and should be 0.0 to preserve future compatibility.

`value$` A string value to set, as determined by `set%`.

Returns The `flags%` value or -3 if there was no key.

Get key values and information

This variant reads back the key-specific values set by `ChanKey(chan%, set%, value{, value$})` and other key information. Note that `ChanKey(chan%, 0)` is part of second variant, above. `ChanKey(chan%, 0, &get$)` is documented, but should not be used and will likely be removed in a future version.

```
Func ChanKey(chan%, get%{, &get$});
```

`chan%` The channel number, as above.

`get%` Identifies the value to return. Negative values read back channel key settings:

`get%`	Returned value
1..n	The key-specific value set by `ChanKey(chan%, set%, value{, value$})`
0	See Modify the flags, above.
-1	Sets `get$` (if present) to the key type, returns 0 if no key type, 1 if a key type (e.g. "`XY`").
-2	The current `flags%` value.
-3, -4	The current key `x` and `y` positions as percentage of the available space. These values will likely not match the values set in the create key call due to the screen pixel resolution. `ChanPixel()` gives you access to pixel sizes.
-5	The current key scale.
-6,-7	The current key width and height in pixels.

`get$` Used to retrieve string values as required by `get%`, matching `value$`. Set empty if no string value to return.

Returns The value requested by `get%`. Unknown `get%` values return as 0. If there is no key, the return value is -3.

See also:
`XYKey()`, Sonograms

ChanKind()

This returns the type of a channel in the current time, result or XY view.

```
Func ChanKind(chan%)
```

`chan%` The channel number.

Returns A code for the channel type or -3 if this is not a time, XY or result view:

0	None/deleted	3	Event (Event+)	6	WaveMark	9	Real wave
1	Waveform	4	Level	7	RealMark	120	XY channel
2	Event (Event-)	5	Marker	8	TextMark	127	Result channel

The result and XY codes changed at version 3.16. Use `ViewKind()` to detect result and XY views if the script must be compatible with all versions of Spike2.

Here are some script definitions that may be useful, or at least save you some typing.

```
const kChanOff%:=0, kWaveform%:=1, kEventFall%:=2, kEventRise%:=3,
      kLevel%:=4, kMarker%:=5, kWaveMark%:=6, kRealMark%:=7,
      kTextMark%:=8, kRealWave%:=9, kXYChan%:=120, kResultChan%:= 127;

' This function convert a value returned by ChanKind() or SampleChanInfo()
' into a string representing the channel kind (useful for reports).
Func ChanKind$(kind%)
docase
case kind%=kChanOff% then return "Off";
case kind%=kWaveForm% then return "Waveform";
case kind%=kEventFall% then return "Event-";
case kind%=kEventRise% then return "Event+";
case kind%=kLevel% then return "Level";
case kind%=kMarker% then return "Marker";
case kind%=kWaveMark% then return "WaveMark";
case kind%=kRealMark% then return "RealMark";
case kind%=kTextMark% then return "TextMark";
case kind%=kRealWave% then return "RealWave";
case kind%=kXYChan% then return "XYChan";
case kind%=kResultChan% then return "Result";
else return "Unknown";
```

```
endcase;
end;
```
See also:

`ChanList(), MarkInfo(), MemChan(), SampleChanInfo(), ViewKind()`

ChanLinPred()

This command is equivalent to the Analysis menu Linear Predict... command. It replaces a short range of Time view Waveform or RealWave channels or Result view channels with data based on a linear prediction of the preceding and following data. This command was added to Spike2 at version [10.08]. See the `LinPred()` command for a discussion of the method and some references.

```
Func ChanLinPred(cSpc, x1, x2, x3{, x4});
```

`cSpc` A channel specifier for the Time view or Result view channels to process. Time view channels must be Waveform or RealWave. All channels must be writeable. A memory channel is always writeable. You can replace data in a waveform channel, but you cannot write data into a gap in a file-based channel (unless the gap is created by the Skip NaN channel process).

`x1` In x axis units (seconds in a time view). The start of the forward data that extends up to but not including `x2`. In a time view, if this range includes a gap, only the contiguous data that ends at `x2` is used. If `x1` is >= `x2` there is no forwards prediction data.

`x2` The start of the region that is to be replaced, in x axis units (seconds in a time view). Ideally, the replacement region is short compared to the regions used for prediction. There is an (arbitrary) limit of 1024 points on the number of points that we allow this algorithm to replace. If you attempt to replace more than this number, no change is made to the data.

`x3` The replace region runs up to, but not including `x3` (in x axis units). This is the start of the backwards prediction area, if `x4` is provided and is greater than `x3`.

`x4` Defaults to the value of `x3` if omitted. The backward prediction data runs up to, but not including `x4`. If `x4` is <= `x3` there is no backwards prediction data.

Returns The number of successfully processed channels or a negative error code. 0 is possible if there are no suitable channels in the specification.

- -1 No prediction data found or error reading it.
- -2 There was a gap between the end of the forward prediction data and the start of the replace region or between the end of the replace region and the start of backwards prediction data. Increase the size of the replacment area.
- -3 The replacement area is too large (current maximum is 1024 points).
- -4 The replacement area of a disk-based channel has a gap (so cannot be filled). Copy the channel to a memory channel, fill it, then save as a disk channel.
- -5 The channel cannot be modified (virtual channel, or read-only channel, or channel being sampled).
- -6 Out of memory.
- -7 Error writing the generated data.

See also:

`LinPred()`

ChanList()

This function fills an array with a channel list for a time, result or XY view and can manipulate channel lists. The result can be filtered to a subset of the available channels. There are two command variants to obtain lists of channels and a variant to manipulate channel lists:

Fill an array with channels from the current view

To use this variant, the current view should be a time, result or XY view.

```
Func ChanList(list%[]{, types%});
Func ChanList(list%[], str${, types%});
```

`list%` An integer array to fill with channel numbers. Element 0 is set to the number of channels returned. The remaining elements are channel numbers. If the array is too short, enough channels are returned to fill the array. It is unnecessary to list all the channel numbers for an XY view, since they are numbered contiguously. Channels are added to the list in ascending channel order. From [10.12] you are allowed a zero length array, which is interpreted as a list of size 0.

`types%` This specifies the channels to return. If omitted, all channels are returned. Select channel types by adding the following codes, which are given as decimal and hexadecimal. XY views only support codes 1024 and 2048.

1	0x1	Waveform or result view channel
2	0x2	Event+ and Event- channels
4	0x4	Event +- channels (level data)
8	0x8	Marker channels
16	0x10	WaveMark data
32	0x20	TextMark data
64	0x40	RealMark data
128	0x80	Unused/deleted disk channels
256	0x100	Deleted channels on disk
512	0x200	Real wave channel

If none of the above values are used or this is an XY view, the list includes all channels. Add the following codes to exclude channels from the list:

1024	0x400	Exclude visible channels
2048	0x800	Exclude hidden channels
4096	0x1000	Exclude time view disk channels but not their duplicates
8192	0x2000	Exclude memory channels but not their duplicates
16384	0x4000	Exclude duplicated channels
32768	0x8000	Exclude selected channels
65536	0x10000	Exclude non-selected channels
2097152	0x200000	Exclude virtual channels but not their duplicates

`str$` A channel specifier such as `"1..10,v1,m1"` or `"v1a"`. You can also match channels by the contents of their titles, units and comments. Only channels that exist in the current view are returned in `list%`; you cannot match deleted or unused channels. If `types%` is provided, only channels that match both the string and `types%` are returned in `list%`. If the specifier contains an error, channels up to the error are returned in `list%`, and the function return value is set to -1.

Returns The number of channels that would be returned if the array were of unlimited length or 0 if the view is not the correct type or -1 if a channel specifier was given that could not be parsed.

Manipulate channel lists [9.02]

Although you can manipulate channel lists from a script, this command variant simplifies the process by providing list operations. Note that this command makes no reference to a view to check that the channel numbers in the lists are valid. However, duplicated channel numbers in the lists are amalgamated and channel numbers other than 1 to 32767 are ignored.

```
Func ChanList(list%[], list1%[]{, op%});
```

`list%` A channel list that is to be modified. On entry, `list%[0]` is set to the number of following channel numbers. On exit, the list has the same format. If the array is too short, enough channels are returned to fill the array. From [10.12] you are allowed a zero length array, which is interpreted as a list of size 0.

`list1%` The list that is used to modify `list%`. This is in the same format as the `list%` array.

`op%` This argument defaults to 1 if omitted and sets the operation to perform:

`op%`	Operation
-1	Subtract. Remove channels from `list%` that are in `list1%`.
0	Intersect. Set `list%` to the channels that are in both `list%` and in `list1%`.

1 Add. Set `list%` to the channels that are in `list%` or in `list1%`.

2 Not intersect. Set `list%` to the channels in `list%` and `list1%` that are not in both lists.

Return -1 if either source array cannot be interpreted as a channel count followed by channel numbers. Otherwise, the return value is the size of the result of the operation (even if it exceeds the size of `list%`).

Number of channels in a data file

You can use the first variant of this command to find the number of channels in a data file:

```
var list%[1], nCh%;
nCh% := ChanList(list%, 0x2063ff); 'number of channels in a data file
```

This works by counting channels of all types and unused and deleted channels, but excludes memory channels, duplicated channels and virtual channels.

See also:

`ChanKind(), ChanOrder[], ChanShow(), DlgChan()`

ChanMeasure()

This takes a wide range of measurements on a channel in a time or result view. It is equivalent to the interactive cursor regions measurements, but takes the time range to process from user-supplied arguments rather than cursor positions. If the function you want is not provided here you can use `ChanData()` to get the individual data values.

```
Func ChanMeasure(chan%, type%, sPos, ePos{, &data%{, kind%}});
```

`chan%` The number of the channel to measure.

`type%` The type of measurement to take, see the documentation of the Cursor regions measurements for details of these measurements. Note that codes 1 and 5 are the same when used from a script as there is no area to subtract as there can be in the Cursor Regions dialog. The possible values are:

1 Area	9 Minimum	17 Mean in X
2 Mean	10 Peak to Peak	18 SD in X
3 Slope	11 RMS Amplitude	19 Mean of absolute values
4 Sum	12 Standard deviation	20 Standard Error of the Mean
5 Area (scaled)	13 Absolute maximum	21 Median [10.13]
6 Curve area	14 Peak	22 Median size [11.00]
7 Modulus	15 Trough	
8 Maximum	16 RMS Error	

`sPos` The start position for the measurement in x axis units. If `sPos` is greater than or equal to `ePos`, the result is 0. In a result view, it is a common mistake to use `Cursor(1)` when you meant `BinToX(Cursor(1))` as the `Cursor()` command in a result view returns a bin number, not an x axis position.

`ePos` The end position in x axis units. In a result view, this is converted to a bin number which must be greater than the bin number obtained from `sPos`.

`data%` Optional variable returned as 1 if a valid result was obtained or as 0 if there is no result (equivalent to a blank cell in the Cursor Regions dialog).

`kind%` This optional variable forces a WaveMark channel to be treated as a waveform or as events for this measurement. If 0 or omitted, the channel is treated as a waveform if drawn in waveform, WaveMark or Cubic Spline mode, otherwise it is treated as an event channel. 1 forces waveform and 2 forces event.

Returns The function returns the requested measurement value.

Notes

Measurement types 21 and 22 are computed for waveform and RealWave channels only.

Measurement type 22, *Median size*, is a robust (not affected by occasional extreme values) measure of the amplitude of the channel about the median. In script language terms, it could be calculated for array `data[]` with:

```
var medData := ArrMedian(data[]);
ArrSub(data[], medData);
Abs(data[]);
var size := ArrMedian(data[]);
```

See also:
`ArrSum()`, `ChanData()`, `ChanValue()`, `Count()`, `Cursor()`, `XToBin()`
Cursor Regions dialog, Waveform channels and Result view measures, Event measures

ChanNew()

This function creates a new channel in the current time view. Unlike a memory channel, the created channel is permanent; any data written to it occupies disk space. You can use this to create channels for use with `ChanWriteWave()` and `ChanSave()`. If you just want to copy a channel, see the `ChanSave()` command.

```
Func ChanNew(chan%, type%{, size%{, binsz{, pre%{, trace%}}}});
```

`chan%` The channel number to use in the range 1 to the number of channels in the data file (usually 32) or 0 for the first unused disk-based channel. You cannot use this routine to overwrite an existing channel; use `ChanDelete()` to remove it first.

`type%` The type of channel to create. Codes are:

1	Waveform	4	Level (Event+-)	7	RealMark
2	Event (Event-)	5	Marker	8	TextMark
3	Event (Event+)	6	WaveMark	9	Real wave

`size%` Used for TextMark, RealMark and WaveMark channels to set the maximum number of characters, reals or waveform points to attach to each item. Historically, with 32-bit data files, this used to set the disk buffer size in bytes for other channel types; we recommend that you use 0 for the default buffer size. You can omit this for channel types 2, 3 and 4.

`binsz` Must be used for waveform and WaveMark data to specify the time interval between the waveform points. This is rounded to the nearest multiple of the underlying time resolution. If you set this 0 or negative, the smallest bin size possible is set. You can omit this for types 2, 3 and 4.

If `binsz` is not a multiple of the underlying time resolution times *time per ADC* for the file, versions of Spike2 before 4.03 cannot read it.

`pre%` This must be present for WaveMark data to set the number of pre-trigger points.

`trace%` Optional, default 1, the number of interleaved traces for WaveMark data.

Returns The channel number if the channel was created, or a negative error code.

The channel has default title, units and comment. Use `ChanTitle$()`, `ChanUnits$()`, `ChanComment$()`, `ChanScale()` and `ChanOffset()` to set these. The following code creates a copy of channel `wFrom%` (a waveform channel) in channel `wTo%`:

```
func CopyWave%(wFrom%, wTo%) 'Copy waveform to a memory channel
var err%, buffer%[8192], n%, sTime := 0.0;
if ChanKind(wFrom%)<>1 then return -1 endif; 'Not a waveform!
if ChanKind(wTo%)<>0 then ChanDelete(wTo%) endif; 'Check if used
ChanNew(wTo%, 1,0,BinSize(wFrom%));      'Create waveform channel
if err%= wTo% then                       'Created OK?
  ChanScale(wTo%, ChanScale(wFrom%));    'Copy scale...
  ChanOffset(wTo%, ChanOffset(wFrom%));  '...and offset...
  ChanUnits$(wTo%, ChanUnits$(wFrom%));  '...and units
  ChanTitle$(wTo%, "Copy");              'Set our own title
  ChanComment$(wTo%, "Copied from channel "+Str$(wFrom%));
  repeat
    n% := ChanData(wFrom%, buffer%[], sTime, MaxTime(), sTime);
    if n% > 0 then                       'read ok?
```

```
      n% := ChanWriteWave(wTo%, buffer%[:n%], sTime)
    endif;
    if n% > 0 then sTime += n% * BinSize(wTo%) endif;
  until n% <= 0;
  ChanShow(wTo%);                          'display new channel
endif;
return err%;   'Returns 0 if created OK
end;
```

See also:
`ChanData(), ChanDelete(), ChanSave(), ChanWriteWave(), FileNew(), MemChan()`

ChanNumbers()

This command shows and hides channel numbers in the current view and gets the channel number state. It is not an error to use this with data views that do not support channel numbers, but the command has no effect.

```
Func ChanNumbers({show%});
```

`show%` If present, 0 hides the channel number, and 1 shows it. Other values are reserved (and currently have the same effect as 1).

Returns The channel number display state at the time of the call.

See also:
`YAxis(), YAxisMode()`

ChanOffset()

Waveform and WaveMark data is stored as 16-bit integers with a scale factor and offset to convert to user units; this function gets and/or sets the y axis value that corresponds to a 16-bit waveform data value of zero. RealWave channels (which store data as 32-bit floating point values) also have a scale and offset; in this case the scale and offset convert the data to 16-bit values when required. The y axis user units for a channel are:

y axis value = (16-bit value)*`scale`/6553.6 + `offset`

RealWave data is stored as 32-bit floating point values, but the channel scale and offset are still available and are used when the channel data needs to be converted into 16-bit integers:

16-bit value = (y axis value - `offset`) * 6553.6 / `scale`

```
Func ChanOffset(chan% {,offset});
```

`chan%` The channel number.

`offset` If present, this sets the new channel offset. There are no limits on the value.

Returns The channel offset at the time of the call for waveform, RealWave or WaveMark channels, or 0.

See also:
`ChanCalibrate(), ChanScale(), Optimise()`

ChanOrder()

This command changes the order of channels in a time or result view, groups channels so that they share a common y axis, sets the channel sorting order and gets list of visible channels in screen order from top to bottom. Before [10.07] all channels in a group had to have a y axis. From [10.07] onwards, only the head channel of a group needs to have a y axis. Any changes to a group that result in a channel without a y axis becoming the head of a group will eject the channel from the group.

There are four command variants:

Move channels

The first moves channels relative to a channel or to a channel position in the list of all channels in the view:

```
Func ChanOrder(dest%, pos%, cSpc);
```

`dest%` The destination channel number or a value less than 1 indicating the position in the list of all channels including hidden channels but excluding channels in `cSpc`. As a position, 0 is the top channel, -1 is the next, and so on. If there are `n` channels, values of `-n` or less are taken to mean the bottom channel. If you drop on top of an ungrouped channel, it must have a y axis as it becomes the head of a group.

`pos%` The drop position relative to the destination channel: -1=above, 0=on top, 1=below. If you drop between grouped channels, dropped channels become group members.

`cSpc` A channel specifier for the channels to move.

Returns The number of moved channels.

Get information and ungroup channels

This variant gets grouping information and can convert all the channels in a group to ungrouped channels.

```
Func ChanOrder(dest%, opt%);
```

`dest%` A channel number in the view that identifies a group. A group is either a single channel, or a set of channels that share a single y axis.

`opt%` 0=returns the number of channels in the group that `dest%` belongs to or 0 if not grouped. 1-`n` returns the channel number of the n^{th} channel in the group or 0 if no channel. -1 ungroups the group and returns the number of changed channels.

Returns The number of channels in the group, the nth channel in the group, or the number of changed channels (including the group head), depending on `opt%`.

Sort channels

The third command variant sorts channels into numerical order *for subsequent use of this command*. The order you set here has no effect on the channel ordering used by `ViewStandard()`. To change that, use the `Profile()` command and set `"Low channels at top"`.

```
Func ChanOrder(order%);
```

`order%` Set -1 for low, 1 for high numbered channels at the top and 0 to use the default channel ordering set by the Edit menu Preferences.

Returns -1 if low-numbered channels or 1 if high-numbered channels were previously at the top.

Get ordered channel lists

The fourth variant gets channel lists, ordered from top to bottom of all channels, all channels that start a group, and all channels within the n^{th} group.

```
Func ChanOrder(list%[]{, sel%});
```

`list%` An array that is filled with a channel list. `list%[0]` holds the number of items that follow in the list. If `list%[0]` holds `n`, `list%[1]` to `list%[n]` hold channel numbers. The returned channel numbers are in display order, with channels at the top of the screen first. If the list is too small to hold all the channels, a valid list of the channels that will fit is returned (excess channels are ignored). The function return value is the full number of channels that would have been returned if there was room for them.

`sel%` Optional, taken as 0 if omitted.

`sel%`	Action
>0	If `sel%` is greater than 0, say `n`, the list is returned holding all the channels in the nth group.
0	The list holds all the visible channels in the view.
-1	The list contains the first channel of each channel group; if there are no groups, `list%[0]` holds 0.

-2 The list holds all the channels in the view, even if they are hidden. This variant was added at Spike2 version [9.02].

Returns The number of channels that match the option. This can be greater than `list%[0]` if the `list%` array is too small.

See also:
Channel specifiers, `ChanList()`, `ChanWeight()`, `ViewStandard()`, `Yaxis()`, `YaxisLock()`

ChanPenWidth()

This sets and gets the pen width for a channel in a Time or Result view; see the `XYDrawMode()` command for XY views. This script command is the equivalent of the View menu channel Pen Width dialog.

```
Func ChanPenWidth(cSpc{, new});
```

`cSpc` A channel specifier for one or more channels.

`new` If present, the new pen width for the specified channels, in points. A point is 1/72 of an inch, which is at least 1 screen pixel. If you set a negative width, the channel uses the pen width set in the Edit menu Preferences for data. A width of zero sets the thinnest pen (1 pixel) possible.

Returns The command returns the current pen width setting for the first channel that exists in the channel specification, or 0 if no channel exists.

See also:
Channel specifiers, Edit menu preferences, Pen width dialog, `XYDrawMode()`

ChanPixel()

This command gets screen pixel information about the current time, result or XY view. All positions are given with respect to a rectangle that has the 0,0 point at the top, left corner with the x co-ordinates increasing to the right and the y co-ordinates increasing downwards. In the following description, the rectangles that we refer to are:

Desktop The rectangle that encloses all your monitors.

View The rectangle that encloses the drawing area of the current time, result or XY view. This does NOT include the frame around the view.

Channel The rectangle that encloses all the data for a particular channel

Superview The channel superview is the rectangle that encloses all the channel displays (not including any x or y axes). In the future we may allow channels to be arranged in a grid, in which case the superview for a channel will be the rectangle that includes all the channels in the column of the grid that includes the nominated channel.

```
Func ChanPixel(chan%, &x, &y{, mode%{, &w|xp{, &h|yp}});
```

`chan%` A channel in the current Time, Result or XY view. This must be supplied as a valid channel for mode 2, but can be set to 0 for modes 0 and 1 if you do not want the y information.

`x` This is a real variable that is set depending on the value of `mode%`.

`y` This is a real variable that is set depending on the value of `mode%`. When the y value depends on a channel, the channel must both exist and be visible.

`mode%` An optional variable, taken as 0 if omitted, that determines the returned values:

0 `x` is set to the number of x axis units equivalent to a move of one pixel to the right. For a WaveMark channel drawn in Overdraw WM mode, this is in terms of the width of the WaveMark. `y` is set to the number of user units that are equivalent to a movement of one pixel up. If there is no y axis or `chan%` does not exist or is not visible, it is set to 0. Normally the x and y

axes are in linear mode, but they can be set to logarithmic mode, in which case the returned values are the log increment that corresponds to one pixel.

You will need this information if you use the `ToolbarMouse()` or `DlgMouse()` function to get the mouse position and want to locate data elements that are close to the mouse pointer.

1 `x` and `y` (if `chan%` exists and is visible) are set to the pixel positions relative to the channel superview of the axis positions `xp`, `yp`. in channel `chan%`. You can find the position of the channel superview with respect to the view with `mode%` set to 3.

2 `x`, `y`, `w` and `h` are returned holding the channel rectangle of channel `chan%`, which must exist and be visible. `x`,`y` are relative to the channel superview.

3 `x`, `y`, `w` and `h` are returned holding the channel superview position and size. The position is relative to the view. Currently there is only one superview, so `chan%` is ignored. In the future we may allow channels to be arranged in a grid, and the channel number will matter as it will determine which superview is required.

4 `x`, `y`, `w` and `h` are returned holding the view position and size. The position is relative to the Windows desktop. `chan%` is ignored and should be 0.

5 `x`, `y`, `w` and `h` are returned holding the x axis position and size. The position is relative to the view. Currently there is only one x axis, so `chan%` is ignored and should be 0. In the future we may allow channels to be arranged in a grid, and the channel number will matter as it will select the x axis to use.

6 `x`, `y`, `w` and `h` are returned holding the y axis position and size for the channel `chan%` (which must exist and be visible). The position is relative to the view.

7 `x`, `y` are returned set to the sizes (in pixels) of a representative character that will be used for a cursor label. This is to allow you to position a vertical cursor label above or below a specific value. The channel is currently ignored.

`w,h` Real variables that are set to the width and height of the rectangle selected by `mode%`.

`xp,yp` The x and y positions in axis units to map to pixels when `mode%` is 1.

Returns A set of flags indicating which values were returned and if the units were modified, or -1 if the command could not be used (usually because a channel number was required and does not exist). Flag values are:

1 The x axis value is set (this should always be set).

2 The y axis value is set (not set if the channel does not exist, is invisible or does not have a y axis).

4 In mode 0, the x axis is in log mode, so the x value is a log increment per pixel.

8 In mode 0, the y axis is in log mode, so the y value is a log increment per pixel.

16 In mode 1, the `xp` value is outside the visible channel rectangle on screen.

32 In mode 1, the `yp` value is outside the visible channel rectangle on screen.

Example

This example calculates the cursor label position to pass to `CursorLabelPos()` to position the cursor label above the intersection of the cursor with the data on a channel. The top of the label is positioned as a percentage of the length of the displayed vertical cursor (regardless of the label height). We return 0 if the view is too small to have any height.

```
const kBad := -1e308;                    'A very large negative number

' c%    The cursor, which must be visible, else returns kBad.
' ch%   The channel, which must be visible with a y axis, else kBad.
' offs  The number of label heights to move down by. 0 places the top
'       of the label at the intersection. Negative values move up.
' Returns  The position for CursorLabelPos() or kBad if invalid.
'       Smaller negative values can occur with negative offs values
'       and if channel data goes beyond the channel area.
func CurLabelCross(c%, ch%, offs)
if not ChanVisible(ch%) then return kBad endif;  'We need a visible channel
if not CursorVisible(c%) then return kBad endif; 'and a visible cursor
```

```
var x := Cursor(c%);                    'the cursor position
var y := ChanValue(ch%, x);             'Get the crossing point

var xp,yp;                              'get pixel position of value
var flags% := ChanPixel(ch%, xp, yp, 1, x, y); ' pixels of the value
if not (flags% band 2) then return kBad endif; 'if no y value (no y axis)

var sx, sy, sw, sh;                     'superview pixel position and size
ChanPixel(ch%, sx, sy, 3, sw, sh);      'Get superview pixels

var lx, ly;                             'label character width, height
ChanPixel(ch%, lx, ly, 7);              'Get label character size

return sh>0.0 ? (100.0*(yp + offs * ly))/sh : 0.0; 'Calculate percentage
end;
```

All kinds of variants on this are possible. For example, you might want the label to be below the intersection (`offs` of 0) when the intersection value was negative and above (`offs` of -1) when it was positive. The following code exercises this function. This assumes that channel 1 exists and is a suitable channel (say a waveform).

```
const nCursor% := 1;                    ' The cursor number
const nCh% := 1;                        ' The channel number
var lastCurPos := -1;                   ' Only update if cursor moves

Func Idle%()
var curPos := Cursor(nCursor%);         ' Get cursor position
if (curPos <> lastCurPos) then          ' Has it changed?
    var perCent := CurLabelCross(nCursor%, nCh%, -1);
    if (perCent >= kBad) then           ' Got a position?
        CursorLabelPos(nCursor%, perCent); ' Position the label
    endif
    lastCurPos := curPos;               ' remember the last position
endif;
return 1;                               ' keep going
end;

ChanShow(nCh%, 1);                      ' Ensure visible
CursorVisible(nCursor%, 1);             ' Ensure visible
CursorLabel(4, nCursor%,  Print$("Ch%d=%%v(%d)", nCh%, nCh%)); ' display "Ch1=value"
ToolbarSet(0, "", Idle%);               ' Set an idle routine
ToolbarSet(1, "&Quit");                 ' and a quit button
Toolbar("Drag cursor 1 around", 1023);  ' Go
```

Most of this code is setting up the cursor and to prevent updates if the cursor does not move or the channel is unsuitable. You could use a much simpler idle function:

```
Func Idle%()
CursorLabelPos(nCursor%, CurLabelCross(nCursor%, nCh%, -1));
return 1;
end;
```

However, this generates a very flickery display (as the idle routine runs very often).

See also:

`DlgMouse()`, `ToolbarMouse()`

ChanPort()

This returns the physical hardware port that sampled data in a time view channel.

```
Func ChanPort(chan%);
```

`chan%` The channel number in the current time view.

Returns The physical data port or -1 if the view is not a time view, or if the channel was not sampled. Both event and ADC ports are enumerated from 0. WaveMark channels with multiple traces will report the port of the first trace.

See also:
`SampleEvent(), SampleWaveform(), SampleWaveMark()`

ChanProcessAdd()

This adds a channel process to a waveform or RealWave channel in a time view, matching the effect of the Add button in the Channel Process dialog.

```
Func ChanProcessAdd(chan%, PType% {, arg1, arg2, ...});
```

`chan%` A waveform or RealWave channel number in the current time view. You can also add a time shift to any event-based channel.

`PType%` The process type. The following process types are currently defined:

- 0 Rectify. Positive values are unchanged, negative values are negated. Data from waveform channels with a non-zero channel offset may be limited.
- 1 Smooth. This has one argument, a time range in seconds. If omitted, it is set to 10 times the channel sample interval. The output at time `t` is the average of the input data from times `t-arg1` to `t+arg1`.
- 2 DC Remove. There is one argument, a time range in seconds. If omitted, it is set to 100 times the channel sample interval. The output at time `t` is the input minus the mean input from `t-arg1` to `t+arg1`.
- 3 Slope. There is one argument, a time range in seconds. If omitted, it is set to 3 times the channel sample interval. The slope at time `t` is calculated from the points in the time range `t-arg1` to `t+arg1`. If you apply this process the channel scale and y axis units change and the channel offset becomes 0. The output is in input units per second.
- 4 Time shift. There is one argument, a time shift in seconds. If omitted it is set to 0.1 seconds. A positive value shifts the trace right in the window, a negative value shifts it left.
- 5 Down sample. There is one argument, the down sample ratio. If omitted, it is set to 1 (take every point).
- 6 Interpolate. Argument 1 is the sample interval, 2 is the alignment. Omitted arguments are set to match the source channel.
- 7 Chan match. Argument 1 is the channel to match. If omitted, this is set to `chan%`.
- 8 RMS amplitude. Argument 1 is the time range. If omitted, it is set to 10 times the channel sample interval.
- 9 Median filter. Argument 1 is the time range. If omitted, it is set to 10 times the channel sample interval.
- 10 Fill gaps. Argument 1 is the maximum gap to interpolate across, 2 is the fill level for larger gaps. Omitted arguments are taken as 0.
- 11 Skip NaN. Removes non-numbers from RealWave data, leaving gaps.
- 12 Debounce. Argument 1 is the minimum acceptable gap between events. If omitted, it is set negative, which has no effect.

`arg#` Optional process arguments to match the Channel Process dialog arguments for each process type. Each process has default arguments. It is an error to provide too many arguments or for an argument to be incorrect for the process.

Returns The index of the added process in the list of processes for the channel.

See also:
More about channel processes, `ChanProcessArg(), ChanProcessClear(), ChanProcessInfo()`

ChanProcessArg()

This gives you access to the parameters of channel processes added by the Channel Process dialog or by the `ChanProcessAdd()` command.

```
Func ChanProcessArg(chan%, id% {,n% {,arg}});
```

`chan%` The channel number in the current time view.

`id%` The process index. The first one added is number 1, the second is 2, and so on.

`n%` An optional argument number. The first argument is 1, the second 2, and so on.

`arg` If present, changes process argument `n%`. Ignored if out of range or illegal.

Returns If `n%` is omitted the function returns the number of arguments process `id%` uses. If `n%` is present it returns the value of argument `n%` at the time of the call.

See also:
`ChanProcessAdd(), ChanProcessClear(), ChanProcessInfo()`

ChanProcessClear()

This removes one or all processes from a channel or all processes from all channels.

```
Func ChanProcessClear({chan%{, id%}});
```

`chan%` A channel number in the current time view. If you omit this argument, or set it to -1, all processes for all channels are removed. From Spike2 version [9.04] you can also use -2 for visible channels and -3 for selected channels.

`id%` The index of the process to delete. Set 0 to delete all processes for channel `chan%`. After deleting, any remaining processes are renumbered.

Returns 0 if at least one process was deleted, -1 if no process was deleted.

See also:
`ChanProcessAdd(), ChanProcessArg(), ChanProcessInfo()`

ChanProcessCopy()

This copies all the channel processing from a source channel to a list of target channels, or clears all processing from a list of channels. You might use this when setting up a group of similar channels by setting the channel processes for the first channel, then copying them to the other channels.

```
Func ChanProcessCopy({cSpc{, src%}});
```

`cSpc` A channel specifier for the target channels.

`src%` If omitted or 0, all processing for the channels defined by `cSpc` is cleared. Otherwise it is the channel to copy the processing from.

Returns 0 if at least one process was deleted, -1 if no process was deleted.

See also:
Channel specifiers, `ChanProcessAdd(), ChanProcessArg(), ChanProcessClear(), ChanProcessInfo()`

ChanProcessInfo()

This returns information about processes attached to a channel.

```
Func ChanProcessInfo(chan% {,id% {, arg%}});
```

`chan%` The channel number in the current time view.

`id%` Omit this to return the count of processes for the channel or set it to the index of the process to return information on. The first added process is number 1.

`arg%` Omit this to return the type of process index `id%`; see `ChanProcessAdd()` for the type codes. If present, the command returns the type of argument `arg%`: 0=integer, 1=real, 2=string (unused), 4=channel. The first argument is number 1.

Returns The information requested or a negative error code.

See also:
`ChanProcessAdd(), ChanProcessArg(), ChanProcessClear()`

ChanSave()

This function (equivalent to the Analysis menu Save channel... command) copies data with an optional time shift from source channels in the current time view to disk-based or memory buffer destination channels in any time view. If a destination channel exists, `ChanSave()` adds data to it, otherwise it creates a disk-based channel to match the source channel. You can set the number of disk-based channels in a new data file with the `nChans%` argument of `FileNew()`.

The source data must be compatible with the destination. All event-based data types are mutually compatible. Destination data that is not present in the source is set to 0. Event times are set as close to the original times as the time bases of the source and destination views allow. If the conversion causes multiple events to fall at the same time, the converter adds 1 tick to separate the times. However, it will not do this for more consecutive events that the ratio of the two time bases.

If the source channel has a marker filter set, only events in the source channel that match the marker filter are copied to the destination channel. If the destination channel has a marker filter, the filter is unchanged for an existing channel and is set to accept all data for a new channel.

From version [9.01], when creating new destination channels, if the source channel has no channel comment, Spike2 will create one if the source channel is a Virtual channel (when it will save the channel expression) or if the source channel has Channel Processes applied (when it will list them), or if the source channel has a Marker filter applied. The comment will also hold the original source channel if it differs from the destination channel.

Waveform and RealWave channels are compatible; conversion between them uses the channel scale and offset values. Waveform to RealWave conversion is always OK; RealWave to Waveform conversion always loses information and can yield nonsense unless the channel scale and offset are chosen carefully.

If the source and destination Waveform, RealWave or WaveMark data sample rates do not match, the data is interpolated using cubic splines.

If a destination disk-based channel already contains data, any added event-based data must be written after all data already contained in the channel. With Waveform and RealWave channels you can overwrite existing data and add data to the end of the channel. You cannot fill in gaps in wave channels; values written into gaps in previously written data are ignored.

From version [10.18] you can write data from a channel to itself to append to existing data, overwrite waveform data and add to memory channels. The source and destination time ranges must not overlap or you will get error -7.

```
Func ChanSave(cSpc, dest%{, dh%{, sTime{, sTo{, dTime{, flags%}}}});
```

`cSpc` A channel specifier for the source channels. If you set multiple channels, `dest%` must be set to 0 or -1.

`dest%` This sets the destination channel or channels in the view identified by `dh%`. If `cSpc` refers to a single channel, this can be a channel number of a disk-based channel or an existing memory channel; otherwise `dest%` must be 0 or -1.

If `dest%` is 0, the lowest unused disk-based channels are used. If `dest%` is -1, the same channel number as the source is used. If the destination channel does not exist, it is created using the source channel settings.

`dh%` The handle of the destination view. Omit or set 0 to use the current time view. See the examples, below, for how to create a new file to copy some or all channels into.

`sTime` The start time to read source data. This is 0.0 if omitted or negative.

`sTo` Source data is read up to this time. If omitted or negative, the end of the data is used.

`dTime` The destination time. If this is omitted, `sTime` is used. The source data is time shifted by `dTime-sTime` before being copied to the destination channels.

`flags%` Optional, new in [10.03], default value 0. Currently controls the display of the progress dialog during long operations (more than a couple of seconds). Values are:

- 0 No progress dialog.
- 1 Display progress dialog without a Cancel button.
- 2 Display progress dialog with a Cancel button. On cancel, created channels are deleted.
- 3 Display progress dialog with Cancel button and ask user if created channels should be deleted on cancel.

Returns With a single source channel, the result is the destination channel number. With multiple source channels, the result is the number of channels copied without error. Negative results are: -1 = `cSpc` illegal, -2 = `dest%` illegal, -3 = `dh%` is not a time view handle, -4 = destination channel could not be created or was not compatible with the source, -5 = an error occurred when copying data, -6 the user cancelled the operation, -7 there is a problem with time ranges when writing a channel to itself.

Changes at version [10.18]

Prior to Spike2 10.18 you could not write a channel to itself. From 10.18 you are allowed to do this, but the source and destination time ranges must not overlap. Also, if a channel is not a Waveform or RealWave, all writes must be after all existing data in a channel.

Append files example

This command has many applications. This example creates a new file and adds the contents of two data files to it. We assume the files hold the same types of channels. In this case we create an output file with the same time resolution as the first input file. This example does not check for errors (`fh%` should be > 0 after open, for example).

```
var fh%,dh%;     'file handles of the source and destination files
fh% := FileOpen("file1.smr", 0, 0);    'open file1, assumes fh% OK
dh% := FileNew(7, 1, View(fh%).Binsize()*1e6, 1, 1);  'make output
View(fh%);ChanSave(-1, -1, dh%);FileClose(0,-1);   'copy and close
fh% := FileOpen("file2.smr", 0, 0);         'open file2 and append
ChanSave(-1, -1, dh%, 0, MaxTime(), View(dh%).MaxTime()+1.0);
FileClose(0,-1);                                    'close file2
View(dh%);ChanShow(-1);Draw(0, MaxTime());     'Display the result
```

Resample Waveform channels

If you want to replay an arbitrary waveform through the 1401 DACs while sampling, the waveform must have a sample interval that the 1401 can achieve. For example, suppose that you want to replay a sound waveform that was sampled at 44.1 kHz. This is a sample interval of approximately 22.676 microseconds, which is not achievable with the 1401 DACs. When you import this data, the underlying time base of the file will be set to a non-integral number of microseconds so that the data will display correctly. One way around this is to resample the data to a 1 microsecond time base:

```
var fh%,dh%;     'file handles of the source and destination files
fh% := FileOpen("file1.smr", 0, 0); 'open file1 with 44.1 kHz data
dh% := FileNew(7, 1, 1, 1, 1);      'make output, 1 us clock ticks
View(fh%);ChanSave(-1, -1, dh%);FileClose(0,-1);   'copy and close
View(dh%);ChanShow(-1);Draw(0, MaxTime());     'Display the result
```

The resampling process uses cubic spline interpolation, which will introduce some distortion into the signal as frequency components approach the Nyquist frequency (half the sampling rate).

Beware files with more than 32 channels

If the source channel has more than 32 channels, you will need to modify the `FileNew()` commands in the previous examples so that the destination file has at least the minimum required number of channels (usually you want more channels than this). The following example creates the minimum number of channels in the new file to match the source:

```
var fh%,dh%;     'file handles of the source and destination files
fh% := FileOpen("file1.smr", 0, 0);                    'open file1
var chans%[1], nCh%;   'dummy list, number of channels in the file
nCh% := ChanList(list%, 0x2063ff);         'get number of channels
if (nCh% < 32) then nCh% := 32 endif;             'Make sure valid
dh% := FileNew(7, 1, 1, 1, 1, nCh%);   'create file, nCh% channels
...
```

Big files and 64-bit files

In addition to the extra argument to `FileNew()` to set the number of channels, there is the `big%` argument to control the file type. Setting the file type determines how big the file can be in both time and disk space. If you set a 64-bit file (recommended), only Spike2 versions from 8 onwards can read the result. If you set a 32-bit "big file", Spike2 version 6 onwards can read it but only 7 onward can modify it. If you omit the `big%` argument, you get a 64-bit file.

Progress dialog and Cancel [10.03]

If you enable the progress dialog, it displays during long operations (more than a second or so). You also have the option to enable a Cancel button to stop long operations early. When operations are canceled, channels created by `ChanSave()` are deleted unless the `flags%` value is 3, when the user is asked if they want to delete the created channels. Channels that already existed before the `ChanSave()` command will not be deleted so any changes made by overwriting data will persist.

See also:

Channel specifiers, `ChanNew()`, `ChanOffset()`, `ChanScale()`, `ChanWriteWave()`, `FileNew()`

ChanScale()

In a time view, this function gets and/or sets the scaling between the 16-bit integer data used to store waveform and WaveMark data and the real units of the channel. RealWave channels (which store data as 32-bit floating point values) also have a scale and offset; in this case the scale and offset convert the data to 16-bit values when required. The 16-bit waveform data has values between +32767 and -32768. The y axis user units for a channel are:

```
y axis value = (16-bit value)*scale/6553.6 + offset
```

With the standard ±5 Volt 1401 ADC inputs, 6553.6 is equivalent to 1 Volt at the 1401 input. In this case, a scale of 1.0 and an offset of 0.0 gives a y axis calibrated in Volts.

```
Func ChanScale(chan% {,scale});
```

`chan%` The channel number.

`scale` If present, sets the channel scaling. We suggest you don't set 0.0 as a scale!

Returns The scale at the time of the call for waveform or WaveMark channels, or 0.

You can set the channel scaling interactively from the Channel Information dialog.

See also:

`ChanCalibrate()`, `ChanOffset()`, `Optimise()`

ChanSearch()

This searches a time or result view channel for a user-defined feature. It is the same as an active cursor search, but does not use or move cursors. Searches are done in the current channel drawing mode. WaveMark data drawn as a waveform is searched as a waveform; to search it as events, set an event drawing mode. There is no need for the channel to be visible; you only need to set the display mode before searching.

```
Func ChanSearch(chan%, mode%, sT, eT{, sp1{, sp2{, width{, flag%{,
lv2}}}}});
```

`chan%` The number of a channel in the time or result view to search.

`mode%` This sets the search mode, as for the active cursors. Modes in italics (15 and 16) cannot be used: mode 16 is not relevant and mode 15 can be emulated by mode 6, 7 or 8. See the active cursor mode dialog documentation for details of each mode.

1 Maximum value	9 Steepest rising	17 Turning point
2 Minimum value	10 Steepest falling	18 Slope%
3 Max excursion	11 Steepest slope (+/-)	19 Outside thresholds
4 Peak find	12 Slope peak	20 Inside thresholds
5 Trough find	13 Slope trough	21 Gap start
6 Threshold	14 Data points	22 Gap end
7 Rising threshold	15 *Repolarisation %*	
8 Falling threshold	16 *Expression*	

`sT,eT` The search start and end positions (in seconds in a time view or bins in a result view). If `eT` is less than `sT`, the search is backwards.

`sp1` This is the amplitude for peaks, threshold level for threshold crossings, baseline level for maximum excursion and number of points for mode 14. It is in the y axis units of the search channel (y axis units per second for slopes). If omitted, the value 0.0 is used. Set it to 0 if `sp1` is not required for the mode.

`sp2` This is hysteresis for threshold crossings and percent for `Slope%`. If omitted, the value 0.0 is used. Set it to 0 if `sp2` is not required for the mode.

`width` This is the width in seconds/bins for slope measurements. It sets the minimum time that the data must be above/below a level for threshold measures. It sets the maximum allowed width of a peak or trough. It sets the minimum gap width to detect in gap detect modes. If omitted, 0 is used. Set 0 if `width` is not required for the mode or you do not want a time constraint.

`flag%` This is the sum of flag values and is 0 if omitted. At the moment, the only value defined is 1=ignore gaps in waveform data (ignored in modes 21 and 22). If this value is not set, a search of a waveform channel will stop at a gap in the data.

`lv2` This is the second level for modes 19 and 20. If omitted, 0 is used.

Returns A positive time or bin number if the search succeeds, -1 if it fails or a negative error code.

See also:
Active cursors, Cursor mode dialog, `ChanMeasure()`, `ChanValue()`, `CursorActive()`

ChanSelect()

This function is used to report on the selected/unselected state of a channel in a time or result view, and to change the selected state of a channel.

```
Func ChanSelect(chan% {,new%});
```

`chan%` The channel number or -1 for all visible channels. Hidden channels and non-existent channels always appear to be unselected.

`new%` If present it sets the state: 0 for unselected, not 0 for selected. If omitted, the state is unchanged. Attempts to change invisible channels are ignored.

Returns The channel state at the time of the call, 0 for unselected, 1 for selected. If you set `chan%` to -1, the function returns the number of selected channels.

You can get a list of all selected channels with `ChanList()`.

See also:
`ChanHide()`, `ChanList()`, `ChanOrder()`, `ChanShow()`, `ChanWeight()`

ChanShow()

Display a channel, or a list of channels in a time, result or XY view. Turning on a channel that is on has no effect. Turning on a channel that doesn't exist has no effect.

```
Proc ChanShow(cSpc {,cSpc...});
```

`cSpc` A channel specifier for the channels to show.

See also:
Channel specifiers, `ChanHide()`, `ChanList()`, `ChanVisible()`, `ChanZoom()`, View menu show/hide

ChanTitle$()

This returns or sets the channel title string in a time, result or XY view. In an XY view the channel titles are visible in the Key window. Channel titles are limited in length when they are written to data files (to 9 8-bit characters for a 32-bit smr file and to 20 Unicode characters for a 64-bit smrx file). It usually makes sense to keep channel titles short.

Setting a duplicate channel title does not change the original channel. Setting the original channel title sets the titles of duplicates unless the duplicates have their own title. If you set the title of a duplicate to `""` (an empty string), the title reverts to the title of the channel it duplicates.

```
Func ChanTitle$(chan%{, new$});
```

`chan%` The channel number. In XY views you can also use 0 to change the overall (displayed) y axis title.

`new$` An optional string holding the new channel title.

Returns The channel title string for the channel. If the channel does not exist, the function does nothing and returns an empty string.

RealMark channel titles

RealMark channels hold a time stamp with a list of attached data values. From Spike2 version [9.02] we allow you to store multiple sets of titles (and units) for RealMark channels in 64-bit `smrx` files. We do this by storing the titles is the file as:

`"Default|title0|title1|title2|..."`

That is, we store a list of titles, with each title separated by the vertical bar character; `n` vertical bars generate `n`+1 title fields. The first field holds the default title to use for all items that do not have a title explicitly set. This is followed by the fields for each item. An empty field means, 'use the default'. For example, if we had a RealMark channel holding 4 attached items, being O_2 saturation, then systolic, diastolic and mean blood pressure, we could set the titles as:

`"Bloods|O2|Sys|Dia|Mean"`

The default is `Bloods` to describe the overall channel contents. This is followed by the titles for the individual items. Any titles that are omitted are set to the default title, so if we were less specific with the titles we could have:

`"Pressure|%"`

This sets the channel title, and the title of all items except the first to `"Pressure"`.

To handle RealMark channels, there is a command variant to manipulate the individual titles for each item without the need to deal with formatting the string and managing the vertical bar characters. If you use the variant without the `index%` argument, this sets the entire channel title string.

```
Func ChanTitle$(chan%, index%{, new$});
```

chan% The channel number of a RealMark channel in a time view. If you use this command variant with any other channel type or view type, the index% is ignored and the command behaves as the first variant described above.

index% This is the 0-based index to the item title we want to get or set. You can also use the special item values:

-3 Get or set the entire title string including any vertical bars.

-2 Get or set the title for the current item (as set by ChanIndex()).

-1 Get or set the default title string.

new$ An optional string holding the new y axis title for the channel and index. If this string includes a vertical bar characters, the index% value is ignored and is treated as if it were -3.

Returns It returns the previous title string of the designated channel as selected by index%. If the channel does not exist or is not of a suitable type, or index% is out of range, the function does nothing and returns an empty string.

Backwards compatibility

If you set different titles for RealMark channel indices, this is not backwards compatible. Versions of Spike2 before 9.02 will see such saved titles as the full string including any vertical bars and the strings will likely be truncated. 32-bit .smr data files do not have the capability to store long strings for units or titles; Spike2 truncates long channel titles written to a 32-bit file.

See also:

ChanComment$(), ChanUnits$(), XYKey()

ChanUnits$()

This gets or sets the units for waveform, RealWave, WaveMark or RealMark channels and result or XY views. From 5.03 you can read back and set the displayed units of time view event-based channels. Changes to event-based channels are lost when the display mode changes. From version 9.02 you can set individual units for RealMark items in 64-bit smrx files. Channel units are limited in length when they are written to data files (to 5 8-bit characters for a 32-bit smr file and to 10 Unicode characters for a 64-bit smrx file). It usually makes sense to keep channel units short.

```
Func ChanUnits$(chan%{, new$});
```

chan% A channel in the time or result view. In an XY view the channel number is ignored. We suggest you use 0 to match ChanTitle$(). If you use this with a RealMark channel, the command will set or get the units for the current drawing mode of the channel unless the new$ string includes a vertical bar character, when it will replace the entire units.

new$ An optional string holding the new y axis units for the channel.

Returns It returns the old units of the designated channel. If the channel does not exist or is not of a suitable type, the function does nothing and returns an empty string.

RealMark channel units

RealMark channels hold a time stamp with a list of attached data values. In some cases, it can be convenient to store a range of values that do not share the same units. From Spike2 version [9.02] we allow you to store multiple sets of units (and titles) for RealMark channels in 64-bit smrx files. We do this by storing the units in the data file as:

```
"Default|units0|units1|units2|..."
```

That is, we store a list of units, separated by the vertical bar character; n vertical bars generate n+1 fields. The first field holds the standard units that are to be used for all items that do not have units explicitly set. This is followed by the fields for each item. An empty field means, 'use the default'. For example, if we had a RealMark channel holding 4 attached items, being O_2 saturation with units %, then systolic, diastolic and mean blood pressure, all with units of mmHg, we could set the units as:

```
"mmHg|%|||"
```

The default is mmHg as this applies to the majority of items. The first item has units of %, the second to fourth fields are all empty, so are defaulted to mmHg. Note that a field holding space characters does not count as a default, so if you want no units for an item, you can use a space character to achieve this. Fields that do not exist are also taken as the default, so we could also represent this data more compactly as:

```
"mmHg|%"
```

To handle RealMark channels, there is a command variant to manipulate the individual units for each item without the need to deal with formatting the string and managing the vertical bar characters. If you use the variant without the index% argument, this sets the entire channel units string.

```
Func ChanUnits$(chan%, index%{, new$});
```

chan% The channel number of a RealMark channel in a time view. If you use this command variant with any other channel type or view type, the index% is ignored and the command behaves as the first variant described above.

index% This is the 0-based index to the item we want to get or set the in the channel. You can also use the special item values:

- -3 Get or set the entire units string including any vertical bars.
- -2 Get or set the units string for the current item (as set by ChanIndex()).
- -1 Get or set the default units string.

new$ An optional string holding the new y axis units for the channel and index. If this string includes a vertical bar characters, the index% value is ignored and is treated as if it were -3.

Returns It returns the previous units string of the designated channel as selected by index%. If the channel does not exist or is not of a suitable type, or index% is out of range, the function does nothing and returns an empty string.

Backwards compatibility

If you set different units for RealMark channel items, this is not backwards compatible. Versions of Spike2 before 9.02 will see such saved units as the full string including any vertical bars and the strings will likely be truncated. 32-bit .smr data files do not have the capability to store long strings for units or titles; Spike2 truncates long file units written to a 32-bit file.

See also:

ChanScale(), ChanOffset(), ChanTitle$()

ChanValue()

In time views this returns the value of a channel at a time. For a waveform channel it returns the waveform value, for other channel types, it returns a value in the y axis units of the channel display mode. If the channel has no y axis or is drawn in raster mode, the value is the time of the next event on the channel.

In a result view, this returns the value of the result corresponding to an x axis value. You can use the [bin] notation to access result views by bin number.

```
Func ChanValue(chan%, pos{, &data%{, mode%{, binsz{, trig%|edge%{, as%}}}}});
```

chan% The channel number in the time or result view.

pos In a time view, the time for which the value is needed. In a result view, this is the x axis value for which a result in needed.

data% Returned as 1 if there is data at pos, 0 if not. For example, for a waveform if there was no data within Binsize(chan%) of pos, this would be set to 0.

mode% If present, this sets the display mode in which to read the value from a time view. If mode% is inappropriate or absent, the current display mode is used. This parameter is ignored in a result view. See DrawMode() for a full description of all drawing modes. The modes in a time view are:

- 0 The current mode for the channel. Any additional arguments are ignored.
- 1 Dots mode for events. Returns the event time at or after pos.
- 2 Lines mode, result is the same as mode 1.

3	Waveform mode.
4	WaveMark mode, returns waveform values for WaveMark channels.
5	Rate mode. The `binSz` argument sets the width of each bin.
6,11	Mean frequency mode, `binSz` sets the time period, 11 is rate per minute.
7,12	Instantaneous frequency, returns value at next event, 12 is per minute.
8	Raster mode, `trig%` sets the trigger channel, result as for mode 1.
13	Cubic spline for waveforms and WaveMark data.
16	Skyline mode for waveform, RealWave and RealMark channels.
17	Interval mode, returns value at next event. New at [10.05].

`binSz` This sets the width of the rate histogram bins and the smoothing period for mean frequency mode when specifying your own mode.

`trig%` The trigger channel for raster displays, we assume raster displays of level data are never required.

`edge%` For level data event channels. This sets which edges of the signal to use for mean frequency, instantaneous frequency and rate modes: 0=both edges, 1=rising edges, 2=falling edges. If `edge%` is omitted, both edges are used.

`as%` Used with instantaneous frequency mode to determine how the data is measured. 0=Default, 1=Dots, 2=Line, 3=Skyline.

Returns It returns the value or 0 if no data is found. For waveform data, if there is no data within `Binsize(chan%)` of the time, the value is zero.

If `data%` is omitted any error stops the script. Errors include: no current window, current window not a time or result view, no data at `pos`, and `pos` beyond range of x axis. If `data%` is present, errors cause it to be set to 0.

See also:
`DrawMode(), ChanData(), ChanMeasure(), Cursor Values, MinMax()`

ChanUndelete()

This can be used in a Time view to test if a disk-based channel is deleted and potentially recoverable and report the type of the deleted channel. It can recover a deleted channel from a 64-bit .smrx file. You cannot use this to recover deleted 32-bit .smr file channels; there is a script `unDelCh.s2s` in the `Scripts` folder of your Spike2 installation that will help you to do this. You can use this to detect that a 32-bit file has a deleted channel; currently it always reports that the channel is a Waveform (we actually do not know the type). `ChanUndelete()` will not recover Virtual, Duplicate or Memory channels. This command was added at Spike2 version [8.10].

```
Func ChanUndelete(chan%{, act%});
```

`chan%` The number of the deleted channel.

`act%` The action to take. If 0 or omitted the command reports the type of the deleted channel (see `ChanKind()` for channel type codes). Set this to 1 to recover the channel.

Returns A negative error code if `chan%` is not a disk channel number or if the request is not possible, for example to recover a channel that is not recoverable. Otherwise:

64-bit `.smrx` file If `act%` is 0 or omitted, the returned value is the channel type that would be recovered or 0 if this is not a recoverable channel. If `act%` is 1, the returned value is 0 if the channel was recovered or a negative error code.

32-bit `.smr` file If `act%` 0 or omitted, the return value is 1 if the channel is not in use and has deleted blocks (so is potentially recoverable), otherwise we return 0. If the return value is 1, the channel could be of any type. **Do not** assume that this is a waveform channel. The `UnDelCh.s2s` script employs various heuristics to guess the channel type and lets you set it yourself. Using this with `act%` set to 1 will return an error.

When you recover a channel it will be hidden, so will not appear immediately. Use `ChanShow()` to make it visible.

See also:
ChanDelete()

ChanVisible()

This returns the state of a channel in a time, result or XY view as 1 if the channel is visible and 0 if it is not. If you use a silly channel number, the result is 0 (not displayed).

```
Func ChanVisible(chan%);
```

chan% The channel to report on.

Returns 1 if the channel is displayed, 0 if it is not.

See also:
ChanShow(), ChanHide()

ChanWeight()

This function sets the relative vertical space to give a channel or a list of channels. The standard vertical space corresponds to a weight of 1. When Spike2 allocates vertical space, channels are of two types: channels with a y axis and channels without a y axis. Spike2 calculates how much space to give each channel type assuming all channels have a weight of 1. Then the actual space allocated is proportional to the standard space multiplied by the weight factor. This means that if you increase the weight of one channel, all other channels get less space in proportion to their original space.

```
Func ChanWeight(cSpc{, new});
```

cSpc The channel specifier for the list of channels to process.

new If present, a value between 0.001 and 1000.0 that sets the weight for all the channels in the list. Values outside this range are limited to the range.

Returns The command returns the channel weight of the first channel in the list.

See also:
Channel specifiers, ChanOrder(), ViewStandard()

ChanWriteWave()

This function writes real or integer data to a waveform (16-bit integer) or RealWave (32-bit floating point) channel. The time gap between array points is the Binsize() value of the channel. You can overwrite existing data and add data to the end of the channel. You cannot fill in gaps in wave channels; values written into gaps in previously written data are ignored.

```
Func ChanWriteWave(chan%, const arr[]|arr%[], sTime);
```

chan% The waveform or RealWave channel in the current time view to write data to. This can be a duplicate channel, a disk channel or a memory channel. If you write to a duplicated channel, the original channel data is changed.

arr A real or integer array to write to the channel. When writing real data to a waveform channel or integer data to a RealWave channel, the data is converted to match the channel format using the channel scale and offset. When writing to a waveform, output is limited to 16-bit integers in the range -32768 to 32767.

sTime The first array point time. When overwriting, if the time does not align with existing data it is reduced by less than one sample interval to align it.

Returns The number of points processed including points skipped due to gaps in existing channel data or a negative error code, for example if the file is read-only.

The function will cause a fatal script error if used on the wrong view type, the wrong channel type or if the system runs out of memory. See the `ChanNew()` command for an example of use.

See also:
`Binsize(), ChanData(), ChanNew(), ChanOffset(), ChanScale()`

ChanZoom()

This function sets and reports the channel zoomed state in a Time or Result view, equivalent to double-clicking a channel. This command was added at [10.08].

```
Func ChanZoom({chan%{, flags%}});
```

`chan%` Omitted or -1 to report the current zoom state. 0 to restore a currently zoomed view to the state before being zoomed. Otherwise a channel number in the view, being the channel to zoom. Requesting a zoom when the view is already zoomed has no effect. Note that unlike interactive use, you are allowed to zoom a channel with no y axis.

`flags%` A set of flags used when a channel is specified for zooming and taken as 0 if omitted. Currently only the value 1 is defined, meaning include duplicates of `chan%` in the zoom operation.

Returns 0 if the view was not zoomed at the time of the call. If the view was zoomed the return value is 1 unless `chan%` was > 0 and the view was already zoomed, when the return value is -1 to signify that no change was made.

See also:
`ChanShow(), ChanHide()`

Chr$()

This function converts a code to a character and returns it as a single character string.

```
Func Chr$(code%);
```

`code%` The code to convert. Codes that have no character representation will produce unpredictable results.

Returns A string representing the code.

Unicode

If you are using a Unicode build of Spike2, the `code%` can be any character in the range 1 to `0xd7ff` or `0xe000` to `0x10ffff`. Characters outside these ranges result in an empty string unless the code is outside 32-bit integer range when the script will stop with a fatal error. Codes in the range 0 to 127 are common to Unicode and ASCII.

See also:
`%c` format in `Print(), Asc(), DelStr$(), LCase$(), Left$(), Len(), Mid$(), Print$(), Replace$(), Reverse$(), Right$(), Str$(), UCase$(), Val()`

Clamp()

This limits a numeric value or a numeric array to a range of values.

```
Func Clamp(x|x[]{[]...}, lo, hi);
```

x A value or a real or integer array of 1 to 5 dimensions. If this is an array, all elements are limited to the range `lo` to `hi`.

`lo` The low value to limit the range of values to. If used with an integer array, this value is truncated (towards zero) to set the low integer value.

`hi` The high value to limit the range of values to. If used with an integer array, this value is truncated (towards zero) to set the upper integer value. The `hi` value must be greater than or equal to `lo`.

Returns When used with a non-array first argument, the returned value is the x input value limited to the range. If used with an array, the result is zero.

Note that when used with an integer array, not all possible integer values are available as limits because a 64-bit real value has less resolution (but much greater range) than a 64-bit integer. See Real data type for a description of why this happens are where (it is around $9.007 * 10^{15}$, so not usually a problem).

See also:
`Abs(), ATan(), Cosh(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan()`

Colour()

Deprecated. This is provided for backwards compatibility with old scripts. Use `ColourGet()` and `ColourSet()` in new scripts.

This function gets and/or sets the colours of items using the colour palette. XY channels are coloured using `XYColour()`. The colours set for time and result view drawing modes can be overridden by `ChanColour()`.

```
Func Colour(item% {,col%});
```

`item%` This selects the item to be coloured. The available items are Given in the table. Note that this table matches the `ColourSet()` table only as far as item 17.

0 Channel number	14 Result dots	28 Cursors
1 Time background	15 Result skyline	29 Controls (not used)
2 Waveform channel	16 Result histogram	30 Grid colour
3 Events as dots	17 Result histogram fill	31 X and Y Axes
4 Events as lines	18 Marker code 00	32 XY background
5 Level events	19 Marker code 01	33 Not saving to disk
6 Marker data	20 Marker code 02	34 Result SEM and SD
7 Mean frequency	21 Marker code 03	35 Fitted curves
8 Inst. Frequency	22 Marker code 04	36 Cluster background
9 Raster dots	23 Marker code 05	37 WaveMark background
10 Rate outline	24 Marker code 06	38 Vertical markers
11 Rate fill	25 Marker code 07	39 XY channels
12 Result background	26 Marker code 08	40 XY fill
13 Result lines	27 TextMark text	41 XY key labels

`col%` If present, this sets the index of the colour in the colour palette to be applied to the item. There are 40 colours in the palette, numbered 0 to 39.

Returns The index into the colour palette of the colour that is nearest to the colour of the item at the time of the call. **Beware**: if you set a colour index for an item and read it back you may not get the index you set if another index holds the same colour.

See also:
Colour dialog, `ChanColour(), PaletteGet(), PaletteSet(), ViewColour(), ViewUseColour(), XYColour()`

ColourGet()

Get the RGB colour for an item in the palette, the main colour table or the marker colour table. This can also be used to get the colour scales used for the sonogram drawing mode and Spike sorting cluster density plots.

Get system colour

This function was added to Spike2 at version 7.07 to encourage users to use RGB colours, rather than a colour palette (see the deprecated `Colour()` command).

```
Func ColourGet(table%{, item%, &r, &g, &b});
```

`table%` -1 to select the colour palette, 0 for the main colour table, 1 for the marker table.

`item%` The item number in the table selected by `table%`. See `ColourSet()` for a description of the item numbers.

`r g b` If present, returned as the red, green and blue colour values in the range 0.0 to 1.0.

Returns If only the table number is supplied, returns the length of the table, otherwise 0.

Get colour scale

This was added at version 9.01 to give script users access to the colour scales used in sonograms, clustering and by the `ArrMapImage()` script command. Colour scales can be viewed, created and deleted in the Sonogram Colours dialog. The colour scales are saved in the registry.

```
Func ColourGet(n%, &name${, map[][]});
```

`n%` The item number of the colour scale. Item numbers 0 through 10 are reserved for preset colour scales (of which items 0 through 7 are implemented). Items 10 through 20 are available for user-defined colour scales. Items -1 and -2 are also defined. Item -1 stores the colour map set for sonogram displays, item -2 stores the colour map set for density plots in clustering dialogs. The preset map numbers are:

Map#	Name	Description
0	Monochrome	Runs linearly from black to white.
1	Rainbow	An approximation to the rainbow.
2	Thermal	Black to red to yellow to white to blue.
3	Geography	Black to blue to green to brown to grey to white.
4	Brightness	Runs from dark blue through mid-red through yellow to white.
5	The Blues	Black through shades of blue and on to white.
6	The Greens	Black through shades of green to white.
7	The Reds	Black through shades of red to white.
8-9		Reserved for future use.

`name$` This is used to return the name associated with the colour scale.

`map` This optional argument is a two-dimensional array. The first dimension must be at least 4, the second should be large enough to collect all the control points (10 is sufficient for all current colour scales). If the second dimension is too short, the array is filled and the extra data is discarded. See *How colour scales are implemented* for a description of the array values that are returned.

Returns The number of positions in the colour scale or 0 if `n%` is valid but the slot is empty or -1 if you request an impossible value of `n%` or -2 if the `map` array is present and its dimensions are too small to be useful.

See also:

Colour dialog, `ChanColourGet()`, `ColourSet()`, `ViewColourGet()`

How colour scales are implemented

The Waveform channel Sonogram display mode and the clustering dialogs use colour scales to generate colour maps. The `Spline2D()` and `ArrMapImage()` script commands also use colour scales when generating bitmaps. The `ColourSet()` and `ColourGet()` script command create, delete and access colour scales.

Colour scales are implemented as a set of from 2 to 10 control points, each of which has a colour. These control points are positioned along a line that can be though of as running between 0 and 1. The first control point is at position 0, the last at position 1 and any others are positioned along the line. The colour at any position on the line is obtained by linearly interpolating the colours of the control points either side of the position.

In terms of the script language, this set of control points is implemented through a two dimensional array. The first dimension has size 4, the second dimension is the size of the number of control points. If we have `map[4][n%]`, representing `n%` control points, elements `map[0][]` hold the Red contribution to the colour as a value in the range 0.0 (no red) to 1.0 (as much red as possible), elements `map[1][]` hold the green contribution and elements `map[2][]` holds the blue contribution. Elements `map[3][]` hold the position of the element on the line. Element positions must increase with the element number. We store the positions such that map[3][0] always holds 0.0 and the last element position is always 1.0. We allow you to define a map in `ColourSet()` with arbitrary positions (as long as they increase with element index), but we convert them into the equivalent of 0 to 1 for storage.

We use floating point numbers for colours to avoid saying anything about how colours are actually implemented for future-proofing reasons.

A very simple array could be:

```
var monoChrome[4][2] := {{0,0,0,0}, {1,1,1,1}};
```

In this case the colours go from black at position 0 to white at position 1 with a grey scale in between. If you want to see all the colour scales defined on your machine you can run this script:

```
var map[4][10];     ' [r, g, b, pos][positions]
var name$;          ' the name (if set) of each colour scale
var n%, sz%;        ' index and number of positions
for n% := -2 to 20 do
    sz% := ColourGet(n%, name$, map);
    if (sz% > 0) then
        PrintLog("Colour scale: %d, name: %s\n%.4f\n\n", n%, name$, map[][:sz%]);
    endif
next
```

If you make a change to a colour scale with script recording enabled, Spike2 will generate the code to implement your change. For example, here is the result of creating the reverse of the Rainbow scale:

```
var v11%:= LogHandle();
FrontView(v11%);
SetColourScaleReverseRain(); 'Set colour scale

Proc SetColourScaleReverseRain()
const ReverseRain[4][5] :=
{
    {1.000, 0.000, 0.000, 0.000},
    {1.000, 1.000, 0.000, 0.247},
    {0.000, 1.000, 0.000, 0.498},
    {0.000, 0.000, 1.000, 0.757},
    {0.251, 0.000, 0.251, 1.000}
};
ColourSet(10, "ReverseRain", ReverseRain);
end
```

You could shorten this further if you do not mind having the `ReverseRain` array as a global by deleting the procedure call, the `Proc` line and the `end` statement.

ColourSet()

Set the RGB colour for an item in the palette, the Application colour table or the marker colour table. If you change the colour of any item that is visible, this will make the display invalid and the item will repaint at the next opportunity. You can also use this command to set the colour scale used for the sonogram and the spike shape cluster density display.

Set system colour

These functions were added to Spike2 at version 7.07 to encourage users to use RGB colours, rather than a colour palette (see the deprecated `Colour()` command).

```
Func ColourSet(table%, size%);
Func ColourSet(table%, item%, r, g, b);
```

`table%` -1 to select the colour palette, 0 for the Application colour table, 1 for the marker table.

`size%` Used to set the size of the marker colour table (`table%` = 1) in the range 3 to 256. The initial, default value is 9 for marker colours 0-8. If you increase the size, the new table elements are set to black. Setting sizes of 1 and 2 set the minimum size of 3. The following values have a special effect:

- 0 Reset the table to its default size, deleting any additional colours that were set. This only works for the marker colour table.
- -1 Reset the table to its default size and resets the colours to the defaults for a white background.
- -2 Reset the table to its default size and resets the colours to the defaults for a black background (Dark mode, new at Spike2 version [9.02]).

Setting the table size only works for the marker colour table.

`item%` The item number in the table selected by `table%`. This is an index from 0 to 39 for the colour palette. It is one of the following for the main colour table:

0 Channel numbers	13 Result lines	26 Fitted curves
1 Time background	14 Result dots	27 Cluster background
2 Waveform channel	15 Result skyline	28 WaveMark background
3 Events as dots	16 Result histogram	29 Vertical markers
4 Events as lines	17 Result histogram fill	30 XY channels
5 Level events	18 TextMark text	31 XY fill
6 Marker data and code	19 Cursors	32 XY key labels
7 Mean frequency	20 Controls (unused)	33 Info window text
8 Inst. Frequency	21 Data area grid	34 Info window background
9 Raster dots	22 X and Y Axes	35 Grid view background
10 Rate outline	23 XY background	36 Grid view text (18 before [10.05])
11 Rate fill	24 Not saving to disk	37 Graphical sequencer background [10.09b]
12 Result background	25 Result SEM and SD	38 Graphical sequencer foreground [10.09b]

For the marker colour table it is an index from 0 up to the table size -1.

`r g b` If present, sets the red, green and blue values in the range 0 (no colour) to 1.0 (maximum colour). Values outside this range are limited to the range 0.0 to 1.0.

Returns The return value for the variant that sets a table size is the table size before any change. Use `ColourGet()` to get the table size without changing it. The variant to set an item colour returns 0.

Hierarchy of colours

The colour of items in a channel is determined by the following sequence:

1. If there is a colour set for the channel (see `ChanColourSet()`), this is used unless this is a sample document and the data is not being saved, in which case the *Not saving to disk* (Application colour item 24) colour is used.
2. If there is a colour set for the view (see `ViewColourSet()`), this is used.
3. Otherwise the item is looked up in the Application colour table and that colour is used.

You can think of this as colours being inherited from a hierarchy of Application->View->Channel with each item able to override the colour it inherits.

Set colour scale or delete user-defined item

This was added at version 9.01 (and 8.16) to give script users access to the colour scales used in sonograms and clustering. Colour scales can be viewed, created and deleted in the Sonogram Colours dialog. The colour scales are saved in the registry. This command allows you to write the non-preset colour scales and to delete user-defined scales.

```
Func ColourSet(n%, name$[, const map[][]]);
```

`n%` The item number of the colour scale. Item numbers 0 through 10 are reserved for preset colour scales (of which items 0 through 7 are implemented). Items 10 through 20 are available for user-defined colour scales. Items -1 and -2 are also defined. Item -1 stores the colour map set for sonogram displays, item -2 stores the colour map set for density plots in clustering dialogs. From version 9.06, a change to the sonogram map causes sonograms to update. Currently, we do not force cluster view density plots to update (this may be done in a future version).

`name$` This sets the name associated with the colour scale. If this name is blank and this a user-defined colour scale, the user-defined entry is deleted. You cannot delete the preset entries or items with negative numbers.

`map` This argument is a two-dimensional array. The first dimension must be at least 4, the second should be of size from 2 to 10). See *How colour scales are implemented* for a description of the array values that you should fill in. Note that if you use `ColourSet()` followed by `ColourGet()`, you may not get back exactly the same values. The current system quantises the colour and position values to one part in 256. From version 9.06 you can omit this when `name$` is blank.

Returns 0 for success or -1 if you request an impossible value of `n%` or -2 if the `map` array is present and its dimensions are too small or too large to be usable, -3 if the position values in `map[3][]` are not in ascending order and -4 if there was any other problem. If you set Red (`map[0][]`), Green (`map[1][]`), or Blue (`map[2][]`) values that are outside the range 0 to 1, they are limited to this range. The position values do not need to be in the range 0.0 to 1.0 as these are scaled by the command so that the first position is 0.0 and the last is 1.0; the scaling is linear.

See also:
Colour dialog, `ChanColourSet()`, `ColourGet()`, `ViewColourSet()`, `ViewUseColour()`

Conditioner commands

The `Cond…` family of commands control signal conditioners. They support the CED 1902 programmable signal conditioner, the Power 1401 programmable gain options and the Axon Instruments CyberAmp and the Digitimer 360 and 440. Other conditioners may be added in the future.

These commands do not select which serial port (if any) the conditioner uses or the type of conditioner supported. You choose the conditioner type when you install Spike2. You set the serial port in the Edit menu Preferences option. All these commands require a `port%` argument. This is the physical waveform input port number that the conditioner is attached to. It is not a channel number in a time view.

You can access the built-in interactive support for the conditioner from the Sampling Configuration channel parameters dialog. This can be a useful short-cut to getting the lists of gains and signal sources available on your conditioner(s).

Conditioner link to TextMark channel during sampling

When you make changes to the conditioner settings from the interactive Conditioner dialog during sampling, in addition to applying your changes, the modifications are also recorded by writing to the TextMark channel (if it is enabled). If you make changes using the script commands, *this does not happen*; you can choose to record the changes in any TextMark channel using the `SampleText()` command, or in any other way that you choose.

See also:
`CondFeature()`, `CondFilter()`, `CondFilterList()`, `CondFilterType()`, `CondGain()`, `CondGainList()`, `CondGet()`, `CondOffset()`, `CondOffsetLimit()`, `CondRevision$()`, `CondSet()`, `CondSourceList()`, `CondType()`

CondFeature()

This command gets and sets special signal conditioner features that are not general enough to have dedicated commands to support them. See the `CondSet()` command for more details of conditioner operation. There are four command variants:

Get feature count

```
Func CondFeature(port%);
```

`port%` The waveform port number that the conditioner is connected to.

Returns The number of special features supported by the signal conditioner.

Get feature information

The command supports two types of features: those that have a set of discrete values such as ["None", "Rectify"], and those that support a continuous range of floating point values, such as 10.0 to 25.6, for example. To determine if the feature is continuous or discrete, call the function with only the first 3 or 4 arguments.

```
Func CondFeature(port%, feat%, &name${,&flags%{, list$[]}});
Func CondFeature(port%, feat%, &name${,&flags%{, &low{, &high}}});
```

`feat%` The feature number, from 1 to the number of features available

`name$` Returned set to the name of the feature.

`flags%` Returned set to the feature flags. Currently, none are defined, so this will be 0.

`list$` Returned set to an array of the possible settings (strings) for discrete type.

`low` Returned as the low limit for a feature with continuous values.

`high` Returned as the upper limit for a feature with continuous values

Returns The number of discrete feature values, or 0 if the feature supports continuous values over the range returned in `low` and `high`.

Set feature value

```
Func CondFeature(port%, feat%{, value});
```

`value` If present, it sets the value for the feature set by `feat%` and `port%`. If this is a continuous feature, `value` sets the new value. If the feature has `n` discrete setting, `value` should be 0 to `n`-1 to select the feature corresponding to the feature description in `list$[value]`. If `value` exceeds the allowed range, a continuous feature is set to the nearest allowed value and a discrete feature is unchanged.

Returns The feature value at the time of the call (before any change). This is an integer index for features with discrete values otherwise it is the feature value.

See also:

`CondFilterList(), CondGain(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision$(), CondSet(), CondSourceList(), CondType()`

CondFilter()

This sets or gets the frequency of the low-pass or high-pass filter of the signal conditioner. See the `CondSet()` command for more details of conditioner operation.

```
Func CondFilter(port%, high% {,freq{, type%}});
```

`port%` The waveform port number that the conditioner is connected to.

`high%` This selects which filter to set or get: 0 for low-pass, 1 for high-pass.

`freq` If present, this sets the desired corner frequency of the selected filter. See the `CondSet()` description for more information. Set 0 for no filtering. If omitted, the frequency is not changed. The high-pass frequency must be set lower than the frequency of the low-pass filter, if not the function returns a negative code.

`type%` Optional, taken as 1 if omitted. The filter type to use when setting the filter.

Returns The cut-off frequency of the selected filter at the time of call, or a negative error code. A return value of 0 means that there is no filtering of the type selected.

See also:
CondFilterList(), CondGain(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision$(), CondSet(), CondSourceList(), CondType()

CondFilterList()

This function gets a list of the possible filter frequencies of the conditioner. Conditioners that support continuous frequency ranges also supply a list of frequencies to match the list of frequencies shown in the conditioner control panel. See the CondSet() command for more details of conditioner operation.

```
Func CondFilterList(port%, high%, freq[]{, type%});
```

port% The waveform port number that the conditioner is connected to.

high% Selects which filter to get: 0 for low-pass, 1 for high-pass.

freq[] A real vector that is set to the cut-off frequencies of the selected filter. There is always a value of 0 meaning no filtering.

type% Optional, taken as 1 if omitted. The filter type in the range 1 to the number of types (as returned by CondFilterType()).

Returns The number of filter frequencies (including 0) or a negative error code.

See also:
CondFilter(), CondFilterType(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision$(), CondSet(), CondSourceList(), CondType()

CondFilterType()

This function returns information about the filter types supported by the conditioner for the low pass and high pass filters. For example, the CED 1902 mk IV supports a choice of filter types. There are three command variants:

Get number of filter types

```
Func CondFilterType(port%, high%);
```

port% The waveform port number that the conditioner is connected to.

high% Selects which filter to return information for: 0 for low-pass, 1 for high-pass.

Returns The number of filter types.

Get filter type in use

```
Func CondFilterType(port%, high%, 0);
```

Returns The currently selected filter type, from 1 to the number of filter types.

Get filter type information

```
Func CondFilterType(port%,high%,type%{,&name${,&lower{,&upper}}});
```

type% The filter number, from 1 to the number of available filters.

name$ If present, returned holding the name of the filter selected by type%.

lower If present, returned as the lowest filter frequency (excluding 0, meaning 'off').

upper If present, returned as the highest supported filter frequency.

Returns The number of frequency values the filter can be set to (including 0) or 0 if the filter corner frequency can be set to any value in the range lower to upper.

See also:
CondFilter(), CondGain(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision$(), CondSet(), CondSourceList(), CondType()

CondGain()

This sets and gets the gain of the signal passing through the signal conditioner See the CondSet() command for more details of conditioner operation.

```
Func CondGain(port% {,gain});
```

port% The waveform port number that the conditioner is connected to.

gain If present this sets the ratio of output signal to the input signal. If this argument is omitted, the current gain is returned. The conditioner will set the nearest gain it can to the requested value.

Returns The gain at the time of call, or a negative error code.

See also:
CondFilter(), CondFilterList(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision$(), CondSet(), CondSourceList(), CondType()

CondGainList()

This function gets a list of the possible gains of the conditioner for the selected signal source. See the CondSet() command for more details of conditioner operation.

```
Func CondGainList(port%, gain[]);
```

port% The waveform port number that the conditioner is connected to.

gain[] A real vector that is set to the conditioner gains for the selected signal source. If a conditioner (for example, 1902) has a fixed set of gains, this is the set of gain values. If the conditioner supports continuously variable gain, the first two elements of this array hold the minimum and the maximum values of the gain.

Returns The number of gain values if the conditioner has a fixed set of gains or 2 if the conditioner has continuously variable gain. In the case of an error, a negative error code is returned.

See also:
CondFilter(), CondFilterList(), CondGain(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision$(), CondSet(), CondSourceList(), CondType()

CondGet()

This function gets the input signal source of the signal conditioner, and the conditioner settings for gain, offset, filters and coupling. The settings are returned in arguments which must all be variables. See CondSet() for details of conditioner operation.

```
Func CondGet(port%, &in%, &gain, &offs, &low, &hi, &notch%, &ac%{, &typeL%
{, &typeH%}});
```

port% The waveform port number that the conditioner is connected to.

in% Returned as the zero-based index of the input signal source (see CondSet()).

gain Returned as the ratio of output to input signal amplitude (ignoring filtering).

offs A value added to the input waveform to move it into a more useful range. Offset is specified in user units and is only meaningful when DC coupling is used.

low Returned as the cut-off frequency of the low-pass filter. A value of 0 means that there is no low-pass filtering enabled on this channel.

`hi` Returned as the cut-off frequency of the high-pass filter. A value of 0 means that there is no high-pass filtering enabled on this channel.

`notch%` Returned as 0 if the mains notch filter is off, and 1 if it is on.

`ac%` Returned as 1 for AC or 0 for DC coupling.

`typeL%` Optional integer variable returned holding the low-pass filter type number as described for `CondFilterType()`.

`typeH%` Optional integer variable returned holding the high-pass filter type number.

Returns 0 if all well or a negative error code.

See also:
`CondFilter(), CondFilterType(), CondFilterList(), CondGain(), CondGainList(), CondOffset(), CondOffsetLimit(), CondRevision$(), CondSet(), CondSourceList(), CondType()`

CondOffset()

This sets or gets the offset added to the input signal of the signal conditioner. See the `CondSet()` command for more details of conditioner operation.

```
Func CondOffset(port%{, offs});
```

`port%` The waveform port number that the conditioner is connected to.

`offs` The value to add to the input waveform of the conditioner to move it into a more useful range. If this argument is omitted, the current offset is returned. The conditioner will set the nearest value it can to the requested value.

Returns The offset at the time of call, or a negative error code.

See also:
`CondFilter(), CondFilterList(), CondGain(), CondGainList(), CondGet(), CondOffsetLimit(), CondRevision$(), CondSet(), CondSourceList(), CondType()`

CondOffsetLimit()

This function gets the maximum and minimum values of the offset range of the conditioner for the currently selected signal source. See the `CondSet()` command for more details of conditioner operation.

```
Func CondOffsetLimit(port%, offs[]);
```

`port%` The waveform port number that the conditioner is connected to.

`offs[]` This is a real vector that is set to the minimum (`offs[0]`) and the maximum (`offs[1]`) values of the offset range of the conditioner for the currently selected signal source.

Returns 2 or a negative error code.

See also:
`CondFilter(), CondFilterList(), CondGain(), CondGainList(), CondGet(), CondOffset(), CondRevision$(), CondSet(), CondSourceList(), CondType()`

CondRevision$()

This function returns the name and version of the signal conditioner as a string or an empty string if there is no conditioner for the port.

```
Func CondRevision$(port%);
```

`port%` The waveform port number that the conditioner is connected to.

Returns A string describing the conditioner. Strings defined so far include: "1902ssh", where ss is the 1902 ROM software version number and h is the hardware revision level; and "CYBERAMP 3n0 REV x.y.z" where n is 2 or 8.

See also:
CondFeature(), CondFilter(), CondFilterList(), CondFilterType(), CondGain(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondSet(), CondSourceList(), CondType()

CondSet()

This command has two variants, one to reset the conditioner settings and the other to set the conditioner.

Reset the conditioner settings (new in Spike2 9.01)

This sets the stored conditioner settings back to an initialised state; you must read back the state to discover what this is as it will vary between conditioners, and may even be useless for sampling (it might select a grounded input, for example).

```
Func CondSet(port%);
```

port% The 1401 interface waveform port number that the conditioner is connected to. Currently, conditioners are possible on ports 0 to 127, but this may change.

Returns 0 if all well or a negative error code.

Set the conditioner to a desired state

This sets the input signal source, gain, offset, filters and coupling of the conditioner. All values are requests; the command sets the closest possible value to that requested. If it is important to know what has actually been set you should read back the values with CondGet() after setting them, or use the functions for reading specific values.

```
Func CondSet(port%, in%, gain, offs {,low, high, notch%, ac%{, typeL%{,
typeH%}}});
```

port% The 1401 interface waveform port number that the conditioner is connected to.

in% A conditioner has one or more signal sources. For example, the CED 1902 supports Grounded, Single ended, Normal Diff, Inverted Diff... Conditioners of the same type may have different sources. To select a source, set in% to its zero-based index in the list returned by CondSourceList().

gain This is the desired ratio of output to the input signal amplitude (ignoring the effect of any filtering). The actual gain depends on the capabilities of the signal conditioner, see CondGainList(). The gain range may be altered by the choice of signal source. For example, the 1902 Isolated Amp input has a build-in gain of 100. This command sets the nearest gain to the requested value.

offs This is the desired value in user units to add to the input waveform to move it into a more useful range. Offsets are only meaningful with DC coupling. Different conditioners have different offset ranges, and the offset range may be altered by the choice of signal source, see CondOffsetLimit(). The command will set the nearest offset it can to the desired value.

low If present and greater than 0, it is the desired corner frequency of the low-pass filter. Low-pass filters are used to reduce the high frequency content of the signal, both to satisfy the sampling requirement, and in case where it is known that no useful information is to be found in the signal above a certain frequency. If omitted, or 0, there is no low-pass filtering. The actual filter value set depends on the capabilities of the signal conditioner.

high If present and greater than 0, it is the high-pass filter corner frequency. High-pass filters reduce the low-frequency content of the signal. This must be set lower than the frequency of the low-pass filter; if not, the function returns a negative code. If omitted, or set to 0, there is no high-pass filtering.

Different signal conditioners have different ranges of frequency filtering. To find out the real filter frequency set, use CondFilter(). CondFilterList() returns the list of possible filter frequencies.

notch% Some signal conditioners have a mains-frequency notch filter (usually 50 Hz or 60 Hz) used to reduce the effect of mains interference on low level signals. This filter will remove the fundamental 50 Hz or 60 Hz signal; it will not remove higher harmonics (for example 150 Hz). If notch% is present with a value greater than 0, the notch filter is on. If omitted, or 0, the notch filter is off.

ac% The 1902 supports both AC and DC signal coupling. If you set AC coupling you should probably set the offset to zero. If ac% is greater than 0, the signal conditioner is AC coupled. If omitted or 0, the signal conditioner is DC coupled.

typeL% Optional value, taken as 1 if omitted, that sets the low-pass filter type as described for CondFilterType() in the range 1 to the number of filter types.

typeH% Optional value, taken as 1 if omitted, that sets the high-pass filter type.

Returns 0 if all well or a negative error code.

See also:
CondFilter(), CondFilterList(), CondFilterType(), CondGain(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision$(), CondSourceList(), CondType()

CondSourceList()

This function gets a list of the possible signal source names of the conditioner, or the specific signal source name with the given index number. See the CondSet() command for more details of conditioner operation.

```
Func CondSourceList(port%, src$[]|src$ {,in%});
```

port% The waveform port number that the conditioner is connected to.

src$ This is either a string variable or an array of strings that is returned holding the name(s) of signal sources. Only one name is returned per string.

in% This argument lets you select an individual source or all sources. If present and greater than or equal to 0, it is the zero-based index number of the signal source to return. In this case, only one source is returned, even if src$ is an array.

If omitted and src$ is a string, the first source is returned in src$. If src$[] is an array of strings, as many sources as will fit in the string array are returned.

Returns If in% is greater than or equal to 0, it returns 1 or a negative error code. If in% is omitted, it returns the number of signal sources or a negative error code.

See also:
CondFilter(), CondFilterList(), CondGain(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision$(), CondSet(), CondType()

CondType()

This function returns the type of the signal conditioner.

```
Func CondType(port%);
```

port% The waveform port number that the conditioner is connected to.

Returns A value indicating the connected conditioner or an error code. At the time of writing, the following codes are known:

6 Digitimer 360R
5 Digitimer 440
4 Digitimer 360
3 Power1401 with gain controls
2 Axon Instruments CyberAmp
1 CED 1902
0 No conditioner or it is not the type set when installing

-1 Unable to open the communication port assigned for this waveform port
-2 Error setting up the communication port
-3 Failure during conditioner initialisation
-4 No conditioner channels detected
-5 The DLL required is missing or does not have the required entry points (maybe the wrong DLL or out of date)
-6 There is no device on this channel

See also:
`CondFilter(), CondFilterList(), CondGain(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision$(), CondSet(), CondSourceList()`

Cos()

This calculates the cosine of one or an array of angles in radians.

```
Func Cos(x|x[]{[]...});
```

`x` The angle, expressed in radians, or a real array of angles. The best accuracy of the result is obtained when the angle is in the range -2π to 2π.

Returns When the argument is an array, the function replaces the array with the cosines of all the points and returns either a negative error code or 0 if all was well. When the argument is not an array the function returns the cosine of the angle.

See also:
`Abs(), ATan(), Cosh(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan()`

Cosh()

This calculates the hyperbolic cosine of one value or an array of values.

```
Func Cosh(x|x[]{[]...});
```

`x` The value, or a real array of values.

Returns When the argument is an array, the function replaces each value with its hyperbolic cosine and returns 0. When the argument is not an array the function returns the hyperbolic cosine of the argument.

See also:
`Abs(), ATan(), Cos(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sinh(), Sqrt(), Tanh(), Trunc()`

Count()

In a time view this counts events and forms the mean level of a waveform channel. It can also be used in a result view to sum bin contents between a start and end bin number. In a result view you can use `XToBin()` to convert an x axis co-ordinate to a bin number, or use `ChanMeasure(4,...)`.

```
Func Count(chan%, from, upto);
```

`chan%` The channel number in a time or result view. If the channel does not exist, -1 is returned.

`from` The start time in a time view, the bin number in a result view. If `from` is greater than or equal to `upto`, the result is 0.

`upto` The time or bin to run up to but not including.

Returns In a time view, for waveform channels it returns the mean waveform level in the time range (gaps in waveforms are ignored). If the time range falls entirely in a gap, the result is 0. For all other channels, it returns the number of events. In a result view it returns the sum of the bins in the range.

If you want to count the number of waveform points between two times (for example where a waveform has gaps), you can use `ChanData()` to fill an array using code such as:

```
Func CountWave%(chan%, tFrom, tTo)
if ViewKind() then return 0 endif;    'must be time view and wave
if ChanKind(chan%)<>1 and ChanKind(chan%)<>9 then return 0 endif;
const max% := 8000;
var buffer[max%], n% := 0, nGot%;
repeat
    nGot% := ChanData(chan%, buffer, tFrom, tTo, tFrom);
    if (nGot% <= 0) then return n% endif;
    n% += nGot%;
    tFrom += nGot% * BinSize(chan%);
until 0;
end;
```

Change at version [8.04]
Prior to Spike2 version 8.04, in a time view, the time range was inclusive of `upto` but in a result view it was non-inclusive. This means that there will be a difference if there is a data item at exactly the `upto` time in a time view. There is a option in the Edit menu Preferences to set the old behaviour for scripts that depend on it.

See also:
`ArrSum(), ChanData(), ChanMeasure()`

Cursor...()

Cursor()

This returns a cursor position and optionally sets a new position. If you move cursor 0 in a time view, all active cursors 1-9 will search, equivalent to `CursorSearch(1)`. You can find the position of the cursor before the move that took it to the current position with `CursorX()`.

```
Func Cursor(num%{, where});
```

`num%` The cursor number to use in the range 1 to 9 (0 to 9 in a time view).

`where` If present, and `num%` is positive, the new cursor position. If this exceeds the x axis range, it is limited to the x axis. In a time view the position is in seconds. In a result view it is a bin number, use `XToBin()` to convert an x axis value to a bin number. In an XY view the position is in x axis units.

Returns The cursor position at the time of the call (before any move caused by `where`) or -1 if the cursor doesn't exist.

Examples
```
Cursor(1,20);          'Set cursor 1 at position 20
where := Cursor(1);    'Get cursor position
```

Vertical cursors in spike sorting view
The template start and end markers in spike sorting views are not vertical cursors. They can be positioned using `SSTempSizeSet()`. Horizontal cursors in a spike sorting view can be set by the `HCursor()` command.

See also:
`BinToX(), XToBin(), CursorDelete(), CursorLabel(), CursorLabelPos(), CursorNew(), CursorRenumber(), CursorSet(), CursorVisible()`

CursorActive()

This function retrieves the current active cursor mode and optionally sets a new mode and search parameters. This is valid for a time view only. The function is equivalent to the vertical cursor mode dialog. Once you have set a cursor mode you can command it to seek with `CursorSearch()` and tell if the search succeeded with

CursorValid(). You can perform much the same searches without using active cursors with the ChanSearch() command.

```
Func CursorActive(num%);
Func CursorActive(num%, mode%{, ch%, str$|minSt, end$${, def$${, ampLev|n%{,
hyPer{, width{, ref$${, lev2}}}}}}});
Func CursorActive(num%, mode%{, &get$});
```

num% This is the cursor number, from 0 to 9.

mode% If this argument is present and positive, it sets the new cursor mode. Modes in *italics* cannot be used for cursor 0 and are converted to Static mode. See the documentation for the active cursor modes for details of each mode. Modes 20 and 21 were added at Spike2 [10.19].

0	Static	8	Falling threshold	16	Expression
1	*Maximum value*	9	*Steepest rising*	17	Turning point
2	*Minimum value*	10	*Steepest falling*	18	*Slope%*
3	*Max excursion*	11	*Steepest slope (+/-)*	19	Outside thresholds
4	Peak find	12	Slope peak	20	Inside thresholds
5	Trough find	13	Slope trough	21	Gap start
6	*Threshold*	14	Data points	22	Gap end
7	Rising threshold	15	*Repolarisation %*		

Negative values return active cursor settings as a value. Negative mode% values were introduced at Spike [8.03] to replace use of CursorArctiveGet(). get$ returns string values and can always be omitted. When get$ is available, the command return value is the result of evaluating the string or 0 if evaluation fails. Allowed mode% values are:

-1 The active cursor mode (mode%).

-2 The time view channel (ch%) or 0 if no channel is used.

-3 get$ is returned as the start time of the search (str$).

-4 get$ is returned as the end time of the search (end$).

-5 get$ is returned as the default time (def$).

-6 The data point count for mode 14 (n%).

-7 get$ is returned as the amplitude/threshold level as a string (ampLev).

-8 The hysteresis value for thresholds and the percent value for modes 15 and 18 (hyPer).

-9 The width for slope measurements, the reference level width in mode 15, the maximum width of peaks and troughs and the minimum time above threshold (width).

-10 The minimum cursor 0 step (minSt).

-11 get$ is returned as the reference (ref$) for mode 15.

-12 get$ is returned as the second threshold level (lev2).

get$ Used with negative mode% values to get string values.

ch% The time view channel to search. Use 0 for modes that do not require a channel.

str$ This is an expression that sets the start time for the search when num% is not 0. In expression mode (16) this is the expression to evaluate.

minSt This sets the minimum step for cursor 0 in all modes except 0 and 16.

end$ This string expression sets the end limit of the search. This is ignored for cursor 0 operations when it should be an empty string.

def$ Optional. If a search fails, and this string evaluates to a valid time, the cursor is positioned at the time and the position is valid. This is ignored for cursor 0 operations when it should be an empty string; cursor 0 does not move if a search fails. If you do not supply this or it is an empty string, cursors other than cursor 0 are positioned in the middle of the search time range (though this may be modified in future versions to give you more control).

`ampLev` A value or string that sets the amplitude for peaks, threshold level for threshold crossings and baseline level for maximum excursion. It is in the y axis units of the search channel (y axis units per second for slopes). If omitted, the value 0.0 is used. Set it to 0 or an empty string if `ampLev` is not required for the mode.

`n%` The data point count for mode 14. A value of 0 is treated as 1.

`hyPer` The hysteresis for threshold crossings and percent for modes 15 and 18. If omitted, 0 is used. Set it to 0 if `hyPer` is not used by the mode.

`width` This is the width in seconds for all slope measurements. It sets the reference level measurement width in mode 15. It sets the minimum time (Delay) that the data must be above/below a level for threshold measures. It sets the maximum allowed width of a peak or trough. It sets the minimum Gap width in modes 20 and 21. If omitted, 0 is used. Set it to 0 if `width` is not used in the mode or you do not want a time constraint.

`ref$` This string expression is used in mode 15 to set the time at which the 100% value is measured. The 0% value is measured at the start time.

`lev2` This sets the second threshold level in modes 19 and 20. If omitted, 0 is used. You can supply this as a value or as a string to evaluate.

Returns The active cursor mode at the time of the call unless `mode%` is negative when it returns the numeric value of the requested item If the item is a string that fails to evaluate, the return value is 0.

The arguments `str$`, `end$`, `def$` and `ref$` are strings holding expressions that evaluate to a time in seconds. They are typically of the form `"Cursor(0)+1.3"`. They can contain any expression that would be valid in the Cursor mode dialog.

See also:
Active cursors, Cursor mode dialog, `CursorActiveGet()`, `CursorNew()`, `CursorSearch()`, `CursorValid()`, `MeasureChan()`, `MeasureX()`

CursorActiveGet()

This gets active cursor parameters set by `CursorActive()` or the cursor mode dialog. From Spike2 version [8.03], `CursorActive()` can return all these values and the use of this command is deprecated.

```
Func CursorActiveGet(num%, item% {,&value$});
```

`num%` This is the cursor number from 0 to 9.

`item%` This specifies the parameter value as defined for `CursorActive()` to return:

1	`ch%`	3	`end$`	5	`ampLev|n%`	7	`width`	9	`ref$`
2	`str$`	4	`def$`	6	`hyPer`	8	`minSt`	10	`lev2`

`value$` This optional string variable is updated with the parameter value when `item%` is 2, 3, 4, 5, 9 or 10.

Returns The numerical value of items 1, 5, 6, 7, 8 and 10. Items 5 and 10 may return 0 if the item is a string and cannot be parsed. Otherwise it returns the active cursor mode.

See also:
`CursorActive()`, `CursorExists()`, `CursorNew()`, `CursorSearch()`, `CursorValid()`, `MeasureChan()`, `MeasureX()`

CursorDelete()

Deletes a cursor. It is not an error to delete an unknown cursor (which has no effect). You cannot delete cursor 0; use `CursorVisible(0,0)` to hide cursor 0.

```
Func CursorDelete({num%});
```

`num%` The cursor number to delete. If omitted, the highest numbered cursor is deleted. Use -1 to delete all cursors 1-9.

Returns The value of num% if any cursors were deleted or 0 if no cursor was deleted.

See also:
Cursor(), CursorLabel(), CursorLabelPos(), CursorNew(), CursorRenumber(), CursorSet(), CursorVisible()

CursorExists()

Use this function to determine if a vertical cursor exists.

```
Func CursorExists(num%);
```

num% The cursor number in the range 0-9. Cursor 0 always exists in a time view and never exists in any other view.

Returns 0 if the cursor does not exist, 1 if it does.

See also:
CursorDelete(), CursorNew(), CursorValid(), HCursorExists()

CursorFlags()

Since Spike2 version [10.10] cursors have a set of flags associated with them that control what a user is allowed to do interactively with each cursor. This command allows you to set and clear these flags.

```
Func CursorFlags(num%,{ set%{, mask%}});
```

num% The cursor number. Use -1 to set flags for all cursors.

set% The flags to set for the cursor(s) selected by num%. Omit or set -1 for no change. The value is the sum of:

- 1 Disable interactive cursor movement. The mouse pointer does not change when moved over the cursor and the interactive options to set the position and link the position to another cursor are disabled. Script and active cursor operations that change cursor positions are always allowed. If the cursor is locked to another cursor it will still move if the other cursor moves. A cursor with this flag has an F after the cursor number in the cursor label.
- 2 Disable the right-click context menu when the mouse is over the cursor.
- 4 Disable the right-click context menu options to Fix position and unFix position.
- 8 Disable the right-click context menu option to Hide cursor 0 and Delete the other cursors. If you set this flag, the only way to interactively get rid of the cursor is to use the View menu Standard Display command.

Flags you do not set are cleared unless you use the mask% argument to specify which flags to apply.

mask% Chooses which flag options the set% value applies to. If omitted, all options are included. If you supply a mask% value, only the flags with values in mask% are changed. The mask% argument allows you to set and clear specific flags while leaving others alone. For example, to clear the three context menu options without changing the cursor movement option you would use: `CursorFlags(num%,0,2+4+8);`

Returns -1 if the cursor does not exist or the previous cursor flags of the selected cursor or of the first found cursor if num% is -1.

Purpose of this command

The main use of this command is in script-driven analysis procedures where you may allow users to interact with the data and some of the cursors, but they must leave others alone. You already have the more draconian measure of disallowing all access to the Cursor menu commands with the allow% argument to Interact(), DlgAllow(), Toolbar() and Yield() commands. This command gives you more fine-grained control.

You can also use it to provide fixed (not moveable interactively) marker points in Result and XY views which lack the Vertical Markers available in Time views.

Note that setting cursor flags has no effect on script commands or on the results of active cursor movements.

See also:
CursorDelete(), CursorNew(), CursorSet(), CursorVisible()

CursorLabel()

This gets and optionally sets the cursor label style for the view or for a cursor. You can label cursors with a position and/or the cursor number, or with user-defined text. There are two variants: the first sets styles and labels; the second reads back user-defined labels.

```
Func CursorLabel({style%{, num%{, form$}}});
Func CursorLabel(&form$, num%);
```

style% Set 0=None, 1=Position, 2=Number, 3=Both, 4=User-defined. Omit for no change. Style 4 is used with a format string. Styles 0-3 set the styles of cursors selected by num% and the view style for new cursors if num% is -1 or omitted. Style 4 is applied to cursors set by num%; it does not set the view style.

num% A value of -1 or omitting the argument selects all cursors and sets the view style for new cursors, 0-9 selects one cursor.

form$ A user-defined label string with replaceable fields %p, %n and %v(chan) for position, number and channel value; chan is the channel number whose value you require. %w.dp and %w.dv(chan) formats are allowed where w and d are numbers that set the field width and number of decimal places.

Returns A cursor style before any change as 0-4 if a single cursor is selected, or the view cursor style as 0-3. If style% is omitted or not 0-3, the current view cursor style is not changed.

See also:
Cursor(), CursorDelete(), CursorLabelPos(), CursorNew(), CursorRenumber(), CursorSet()

CursorLabelPos()

This lets you set and read the position of the cursor label.

```
Func CursorLabelPos(num% {,pos});
```

num% The cursor number. Setting a silly number does nothing and returns -1.

pos If present, the desired position of the top of the label as the percentage of the distance from the top to the bottom of the cursor (regardless of the label size). 0 is always at the top. 100 would position the top of the label at the bottom of the cursor, and is not achievable as the entire label is always visible. Values less than 0 or that would put the label off the bottom set the label to the top or the bottom. See ChanPixel() for an example of setting the label position based on a channel y axis value.

Returns The cursor position before any change was made.

See also:
Cursor(), CursorDelete(), CursorLabel(), CursorNew(), CursorRenumber(), CursorSet()

CursorNew()

This command adds a new cursor to the view at the designated position. You cannot use this to create cursor 0 in a time view as this cursor always exists. To show cursor 0 use CursorVisible(0,1). A new cursor is created in Static mode (not active).

```
Func CursorNew({where{, num%}});
```

where The cursor position. In a time view it is a time in seconds, in a result view, it is the bin number. Use XToBin() to convert x axis units to bin numbers. In an XY view it is in x axis units. The position is limited to the x axis range. If the position is omitted, the cursor is placed at a position on the screen that is proportional to the cursor number.

num% If this is omitted, or set to -1, the lowest-numbered free cursor is used. If this is a cursor number, that cursor is created. This must be a legal cursor number or -1.

Returns It returns the cursor number as an integer, or 0 if all cursors are in use.

See also:
Cursor(), CursorActive(), CursorDelete(), CursorLabel(), CursorLabelPos(), CursorRenumber(), CursorSet(), CursorVisible(), XToBin()

CursorOpen()

This command reports on the open state or opens the cursor values and regions dialogs for the current Time or Result view. It is a fatal error to call this if the current view is not a Time or Result view. If you use this to open a dialog, the current view is set to the cursor dialog and ViewKind() will return 11 (Other types). Dialogs open in the last used position unless it is not usefully visible, when they open centred on the application window. If the dialog is already open you can use this command to get the dialog handle and change settings. Use FileClose() to close an opened dialog.

```
Func CursorOpen({opt%{, mode%{, xZero%{, yZero%|type%}}}});
```

opt% Set 0 to open the values dialog and 1 to open the regions dialog. Omit or set -1 as the only argument to report on the open state of the cursor dialogs.

mode% Set 1 to open the dialog and show it, 0 or omitted to open and hide it. You may wish to open a window invisible so that you can position it before display. If the dialog was already open, this argument just sets the visible state.

xZero% This sets the state of the Time Zero or Zero Region check boxes and the associated cursor column selection. Set -2 for unchecked, -1 or omit for no change or set 0 to 9 for the values or 0 to 8 for the regions dialog to check the box and select a column. 0 selects the first column, 1 the second, and so on.

yZero% This sets the state of the Y Zero check box and associated column selection for the value dialog. Set -2 for unchecked, -1 or omit for no change and 0-9 to check the box and select a column. 0 selects the first column, 1 the second, and so on.

type% This sets the measurement type for the regions dialog. Set 1-19 to set a measurement as for ChanMeasure(). Omit or set -1 or 0 for no change. See the documentation of the Cursor Regions window for details of these measurements.

1 Area	5 Area (scaled)	9 Minimum	13 Abs max.	17 Mean in X
2 Mean	6 Curve area	10 Peak to Peak	14 Peak	18 SD in X
3 Slope	7 Modulus	11 RMS Amplitude	15 Trough	19 Mean of abs
4 Sum	8 Maximum	12 Standard deviation	16 RMS error	

Returns If opt% is -1 or omitted the return value is the sum of 1 if the values dialog is open and 2 if the regions dialog is open. Otherwise the return value is the handle of the dialog, or 0 if the dialog failed to open for some reason.

See also:
ChanMeasure(), ChanValue(), FileClose(), Window()

CursorRenumber()

This command renumbers the cursors from left to right in the view. It has no effect on cursor 0. The visible state of a cursor has no effect on renumbering. Active cursor and label information stays with the cursors, not the number.

```
Func CursorRenumber();
```

Returns The number of cursors that were renumbered (this is the number of vertical cursors that exist, not including cursor 0).

See also:
Cursor(), CursorActive(), CursorDelete(), CursorLabel(), CursorLabelPos(), CursorNew(), CursorSet()

CursorSearch()

This function causes active cursors in a time view to search according to the current cursor mode. You can cause all cursors to search, or a restricted range of cursor numbers. Moving cursor 0 with Cursor(0, newPosition) also causes all cursors 1-9 to search. If you have active horizontal cursors, searching will also cause these cursors to move. From Spike2 8.09 you can find the positions the cursors moved from with the CursorX() and HCursorX() commands.

```
Func CursorSearch(num% {,stop%});
```

num% This is the first cursor number to run the search defined by the active cursor mode. Set this to 0 to cause cursor 0 to search forwards and to -1 for cursor 0 to search backwards. CursorSearch(0) and CursorSearch(-1) are equivalent to the Ctrl+Shift+Right and Ctrl+Shift+Left key combinations.

stop% This optional argument is the number of the last cursor to try to reposition. If you omit this argument, all cursors from num% upward will search according to their active mode. To reposition a single cursor set stop% the same as num%.

Returns The time that cursor num% moved to or -1 if the search failed. Use CursorValid() to test if searches of other cursors succeeded.

If a search fails, cursor 0 is not repositioned. Other active cursors are set to the middle of the search range (though we reserve the right to modify this in the future, so do not depend on this).

See also:
Cursor(), CursorActiveGet(), CursorNew(), CursorValid(), MeasureChan(), MeasureX()

CursorSet()

This sets the number of vertical cursors in addition to cursor 0. It deletes cursors 1-9, then positions num% cursors equally spaced in the view, numbered in order from left to right. Finally, if any cursors positions are given, they are applied. The cursor labelling style is not changed. This destroys all active cursor information for the deleted cursors.

```
Proc CursorSet(num% {,where1 {,where2 {,where3 {,where4...}}}});
```

num% The number of cursors to display in the range 0 to 9. It is a run-time error to ask for more than 9 or less than 0 cursors. If num% is 0, cursor 0 is hidden.

whereN Optional positions of cursor N (1 to 9). Positions that are out of range are set to the nearest valid position. In a time view positions are in seconds. In a result view, they are the bin number; use XToBin() to convert x axis units to bin numbers. In an XY view the position is in x axis units. You cannot change the position of cursor 0 in a time view with this command.

Examples:

```
CursorSet(0);       'Delete 1-9, hide cursor 0
CursorSet(2,20,30); 'Delete 1-9, cursor 1 at 20, cursor 2 at 30
```

See also:
BinToX(), Cursor(), CursorDelete(), CursorLabel(), CursorLabelPos(), CursorNew(), CursorRenumber(), CursorVisible(), XToBin()

CursorValid()

Use this function to test if the last search of a cursor in a time view succeeded. Cursor positions are valid if a search succeeds or if the cursor is positioned manually or by a script command. The position of a newly created cursor is valid.

```
Func CursorValid(num%);
```

`num%` The cursor number to test for a valid search result in the range 0-9.

Returns The result is 1 if the position of the nominated cursor is valid or 0 if it is invalid or the cursor does not exist.

See also:
`CursorActiveGet(), CursorNew(), CursorSearch(), CursorVisible(), MeasureChan(), MeasureX()`

CursorVisible()

Vertical cursors can be hidden without deleting them. Interactively you can hide cursor 0, but from a script you can show and hide any vertical cursor. Cursors are always made visible by the `Ctrl+n` key combination.

```
Func CursorVisible(num% {,show%});
```

`num%` The cursor number in the range 0-9 or -1 for all vertical cursors.

`show%` If present set this to 0 to hide the cursor and non-zero to show it.

Returns The state of the cursor at the time of the call (0=hidden or does not exist, 1=visible). If `num%` is -1, the result is the number of vertical cursors.

See also:
`CursorExists(), CursorNew(), CursorSearch(), CursorValid()`

CursorX()

This command gets the position of a cursor before the last move. It is particularly useful with Active cursors. The same command is available as a Dialog Expression. This command was added at Spike2 version [8.09].

```
Func CursorX(num%);
```

`num%` The cursor number to use in the range 1 to 9 (0 to 9 in a time view).

Returns The cursor position before the last move (however it was caused) or -1 if the cursor doesn't exist. Use `Cursor(num%)` to get the current position.

See also:
`BinToX(), XToBin(), Cursor(), CursorDelete(), CursorLabel(), CursorLabelPos(), CursorNew(), CursorRenumber(), CursorSet(), CursorVisible()`

D

```
   DlgEnable()
   DlgFont()
   DlgGetPos()
   DlgGroup()
   DlgImage()
   DlgInteger()
   DlgLabel()
   DlgList()
   DlgMouse()
   DlgProgress()
   DlgReal()
   DlgShow()
   DlgSlider()
   DlgString()
   DlgText()
   DlgValue() and DlgValue$()
   DlgVisible()
   DlgXValue()
Draw()
DrawAll()
DrawMode()
DrawModeCopy()
Dup()
DupChan()
```

Date$()

This function returns a string holding the date. Use `TimeDate()` to get the date as numbers. For this description, we assume that today's date is Wednesday 1 April 1998, the system language is English and the system date separator is "`/`". Default argument values are shown **bold**.

```
Func Date$({dayF%{, monF%{, yearF%{, order%{, sep$}}}}});
```

`dayF%` This sets the format of the day field in the date. This can be written as a day of the week or the day number in the month, or both. The options are:

- 1 Show day of week: "`Wednesday`".
- **2** Show the number of the day in the month with leading zeros: "`01`".
- 4 Show the day without leading zeros: "`1`". This overrides option 2.
- 8 Show abbreviated day of week: "`Wed`".
- 16 Show weekday name first, regardless of the `order%` field.

Use 0 for no day field. Add the numbers for multiple options. For example, to return "`Wed 01`", use `11` (1+2+8) as the `dayF%` argument.

If you add 8 or 16, 1 is added automatically. If you request both the weekday name and the number of the day, the name appears before the number.

`monF%` The format of the month field. This can be returned as either a name or a number. If this argument is omitted, the value 3 is used. The options are:

- 0 No month field.
- 1 Show name of the month: "`April`".
- 2 Show number of month: "`04`"
- **3** Show an abbreviated name of month: "`Apr`"
- 4 Show number of month with no leading zeros: "`4`"

`yearF%` The format of the year field. This can be returned as a two or four digit year.

- 0 No year is shown
- **1** Year is shown in two digits: "`98`".
- 2 Year is shown in two digits with an apostrophe before it: "`'98`".
- 3 Year is shown in four digits: "`1998`".

`order%` The order that the day, month and year appear in the string.

- **0** Operating system settings
- 1 month/day/year

2 day/month/year
3 year/month/day

`sep$` This string appears between the day, month and year fields as a separator. If this string is empty or omitted, Spike2 supplies a separator based on system settings.

For example, `Date$(20, 1, 2, 1, " ")` returns "`Wednesday April 1 '98`". As 20 is 16+4, we have the day first, even through the `order%` argument places the day in between the month and the year. `Date$()` returns "`01/Apr/98`".

See also:
`Seconds(), FileDate$(), TimeDate(), Time$()`

Debug()

This command either opens the debug window so you can step through your script, set breakpoints and display and edit variables or it can be used to stop the user entering the debugger with the `Esc` key.

```
Proc Debug({msg$ | Esc%});
```

`msg$` When the command is used with no arguments, or with a string argument, the script stops as though the `Esc` key had been pressed and enters the debugger. If the debugging toolbar was hidden, it becomes visible. If the `msg$` string is present, the string is displayed in the bar at the top of the script window.

`Esc%` When the command is used with an integer argument, it enables and disables the ability of the user to break out of a running script. If `Esc%` is 0, the user cannot break out of a script into the debugger with the `Esc` key and must wait for it to finish. If `Esc%` is 1, the user can break out. Spike2 enables the `Esc` key each time a script starts, so make this the very first instruction of your script if you want to be certain that the user cannot break out.

This command was included for use in situations such as student use, where it is important that the user cannot break out of a script by accident. It is advisable to test your script carefully before using this option. Once set, you cannot stop a looping script except by forcing a fatal error. Make sure you save your script before setting this option.

See also:
`App(-4), Eval(), DebugHeap(), DebugList(), DebugOpts()`

DebugHeap()

This script command is provided for use by CED engineers to help to debug system problems. It was added at version 6.08. It reports on the state of the application heap, used to dynamically allocate memory. The heap is a list of memory sections. Each section is described by its start address, its size, and if it is in use by the application or is free (available for use). When the application wants more memory, it asks the heap for it. If there is no suitable memory in the heap, the heap requests more memory from the system and uses this to create more heap sections.

The command has the following variants:

Test heap integrity

When called with no arguments, the command tests the integrity of the heap (all command variants do this first). The return values indicate problems in the heap. The error return values apply to all calls to the `Heap()` function.

```
Func DebugHeap();
```

Returns 0 = heap OK, -1 = `_HEAPBADBEGIN` initial header information is bad or cannot be found, -2 = `_HEAPBADNODE` bad node or the heap is damaged, -3= `_HEAPBADPOINTER` a pointer into the heap is invalid, -4 = `_HEAPEMPTY` the heap has not been initialised, -5 = unknown error.

Release unused memory

When called with a single argument of -1 (or any value less than 0), the heap will return unused memory back to the operating system. However, I have never seen this make any difference to the data held by the heap.

```
Func DebugHeap(-1);
```

Returns The same values as the call with no arguments.

Set unused heap memory to known value

When called with a single argument that is a positive number, all unused memory that is owned by the heap is set to this value.

```
Func DebugHeap(fill%);
```

`fill%` All unused bytes in the heap are set to the low byte of this value. This can sometimes be useful when you suspect that unused memory is being used by the program.

Returns The same values as the call with no arguments.

Get heap information

The heap is a list of used an unused sections of memory; this call gets information on the number of used an unused sections and the total size of the used sections and the total heap size.

```
Func DebugHeap(info%[]{, stats%[]});
```

`info%` An integer array of at least 4 elements. The first four array elements are returned holding:

- 0 The total number of memory sections in the heap.
- 1 The number of used memory sections in the heap.
- 2 The total size of memory controlled by the heap.
- 3 The total size of memory that is in used and controlled by the heap.

`stats%` This optional argument is an integer array of at least 32 elements. The elements are returns with heap information. The n^{th} element is returned holding the number of heap sections of a size between 2^n and 2^{n+1}-1 bytes. The very first element, which should hold the count of sections that are 1 byte in size actually holds the count of sections that are 0 or 1 byte long.

Returns The same values as the call with no arguments.

Get heap entries

The last call variant returns the entire heap information, and can optionally return the first few bytes of data held in the heap.

```
Func DebugHeap(walk%[][]);
```

`walk%` This is an integer matrix with at least 3 columns. Ideally it would have at least as many rows as there are memory sections (both used and free) in the heap. If there are more than 3 columns, the additional columns are returned holding the heap data. Each row of the matrix returns information for one section. The columns hold:

- 0 0 if the section is unused, 1 if it is used.
- 1 The start address of the section.
- 2 The size of the section, in bytes.
- 3 If present, returns the first 4 bytes of data in this section.
- 4 If present, returns the next 4 bytes of data in this section.
- n Returns further data (if it exists) in the section.

Returns If there is no error, the function returns the number of sections in the heap. This can be more than the number of rows in the matrix. If there is an error, the function returns a negative value as described for the call with no arguments.

Example

This example prints out the number of system handles in use and a synopsis of the heap usage. If you suspected that your script was causing a memory leak you could insert this procedure, then call it periodically. If there is a steadily increasing use of handles or heap space, check that you are closing all the views you have created.

```
proc ReportHeap()
var info%[4], stats%[32], i%;
DebugHeap(info%, stats%);
PrintLog("Handles=%d, Heap used=%d kB in %d fragments, %d Kb free in %d fragments\n",
 App(-4), info%[3]/1000.0, info%[1], (info%[2]-info%[3])/1000.0, info%[0]-info%[1]);
for i% := 0 to 15 do PrintLog("%6d", pow(2, i%)); next;
PrintLog("\n");
for i% := 0 to 15 do PrintLog("%6d", stats%[i%]); next;
PrintLog("\n");
for i% := 16 to 19 do PrintLog("%4dkB", pow(2, i%)/1024); next;
for i% := 20 to 29 do PrintLog("%4dMB", pow(2, i%)/(1024*1024)); next;
for i% := 30 to 31 do PrintLog("%4dGB", pow(2, i%)/(1024*1024*1024)); next;
PrintLog("\n");
for i% := 16 to 31 do PrintLog("%6d", stats%[i%]); next;
PrintLog("\n");
end;
```

See also:
`App(-4), Debug(), Eval(), DebugList(), DebugOpts()`

DebugList()

This command is used for debugging problems in the system. It writes information to the Log view about the internal list of "objects" used to implement the script language. Low numbered objects are the integers 0 to 20, higher-numbered items are the instructions that the script compiles to followed by objects that represent the built-in script commands.

```
Proc DebugList(list% {, opt%});
```

`list%` This determines what to list and also controls the accumulation of timing information.

- -3 Disable the accumulation of timing information (the normal state) for calls to built-in functions. Timing is a relatively low-cost operation, but not free.
- -2 Enable the accumulation of timing information. This is normally used as a diagnostic of slow script performance when we need to figure out where the time is being spent.
- -1 Reset accumulated times and call counts to 0.
- 0 List a summary of the `DebugList()` options in the Log view.
- 1 List the names of fixed objects (constants and operators). This is usually only of interest to CED programmers.
- 2 List the names of permanent objects (constants, operators and built-in commands).
- 3 List built-in commands (things like `NextTime()`).
- >3 List information for the object with the index list%.

`opt%` This optional argument (default value 0) sets the additional object information to list and is the sum of:

- 1 List the object index number.
- 2 List the object type.
- 4 List timing information for the built-in script commands. The timing information is three numbers: the number of times the command was called, the total time in seconds used and the time per call in microseconds.
- 8 Only list timing information for built-in script commands that were called at least once (added at version 8.03).
- 16 Only list (timing information for) built-in script command that were not called. We use this to check our test program coverage.

To use the timing information:

```
DebugList(-3);   'Disable timing
DebugList(-1);   'Reset call counts and times
DebugList(-2);   'Enable timing
... your code to be timed
DebugList(-3);   'stop accumulating times
```

```
PrintLog("Command name     Total calls      Seconds      us/call Index\n");
DebugList(3, 1+4+8); 'list indices and times for used built-in commands
```

Here is some typical output. The timing is from the point where the command arguments have been prepared up to the point where the command returns control to the script language.

```
Command name     Total calls      Seconds      us/call Index
...
Asc                      537     0.000013        0.023 562
Chr$                     668     0.000092        0.137 563
DelStr$                   13     0.000003        0.203 564
Trim                       2     0.000001        0.733 565
TrimLeft                   2     0.000001        0.293 566
TrimRight                  2     0.000000        0.147 567
InStr                   1081     0.000184        0.170 568
InStrRE                   13     0.010982      844.759 569
LCase$                   130     0.000030        0.232 570
Left$                    339     0.000069        0.202 571
Mid$                    3765     0.000447        0.119 572
Right$                    79     0.000014        0.178 573
...
```

The timing information is normally used by CED programmers to check that functions are working with a reasonable efficiency and when attempting to find out where all the time is going in a script. Any time not accounted for by built-in commands is general script execution time (fetching instruction, running them, allocating variables and so on). If you should discover that a particular built-in function is using a lot of time, you may be able to optimise your use of the function to improve matters. If you think that a particular function is slow, let us know; we may be able to improve it.

See also:
`App(-4), Eval(), Debug(), DebugHeap(), DebugOpts()`

DebugOpts()

This command is used for debugging problems in the system. It controls internal options used for debugging at the system level.

```
Func DebugOpts(opt% {,value%});
```

`opt%` This selects the option to return (and optionally to change). A value of 0 prints a synopsis of available options to the Log view and the current value of each option. Values greater than 0 return the value of that option, and print the option information to the Log view. At the time of writing, only option 1, dump compiled script to the file `default.cod` is implemented.

`value%` If present, this sets the new value of the option.

See also:
`App(-4), Eval(), Debug(), DebugHeap(), DebugList()`

DelStr$()

This function removes a sub-string from a string.

```
Func DelStr$(text$, index%, count%);
```

`text$` The string to remove characters from. This string is not changed.

`index%` The start point for the deletion. The first character is index 1. If this is greater than the length of the string, no characters are deleted.

`count%` The number of characters to remove. If this would extend beyond the end of the string, the remainder of the string is removed.

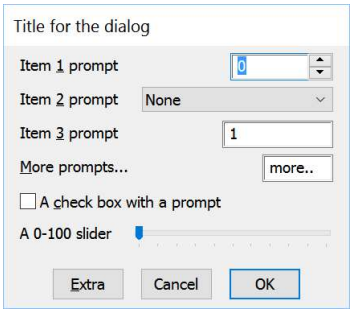
Returns The original string with the indicated section deleted.

Unicode
In a Unicode build of Spike2 there is the possibility that you might attempt to delete half an extended character. The rule we apply is that if the start or end of any string section falls between the lead and trail codes of a Unicode surrogate pair, we move the start or end on by one position, which must put it at the start of a character.

See also:
`Asc(), Chr$(), InStr(), LCase$(), Left$(), Len(), Mid$(), Print$(), ReadStr(), Replace$(), Reverse$(), Right$(), Str$(), UCase$(), Val()`

Dialogs

You can define your own dialogs to get information from the user. You can define dialogs in a simple way, where each item of information has a prompt, and the dialog is laid out automatically, or you can build a dialog by specifying the position of every item. A simple dialog has the structure shown in the diagram:



The dialog is arranged in terms of items. Unless you specifically request otherwise, the dialog items are stacked vertically above each other with buttons arranged at the bottom.

The dialog has a title of your choosing at the top and OK and Cancel plus user-defined buttons at the bottom. When the dialog is used, pressing the `Enter` key is equivalent to clicking on OK.

This form of dialog is very easy to program; there is no need to specify any position or size information, the system works it all out. Some users require more complicated dialogs, with control over the field positions. This is also possible, but harder to program. You are allowed up to 1000 fields in a dialog.

In more complex cases, you specify the position (and usually the width) of the box used for user input. This allows you to arrange data items in the dialog in any way you choose. It requires more work as you must calculate the positions of all the items.

Steps to program a dialog
Generating and using a dialog is a three step process:

1. Use `DlgCreate()` to start designing a dialog. The dialog starts with an `OK` and a `Cancel` button, but you can change that in step 2.
2. Add fields to the dialog with commands like `DlgInteger()`, `DlgChan()` and so on. You can add and modify buttons with `DlgButton()`. You can also configure how the dialog behaves with `DlgAllow()` and set mouse interactions with `DlgMouse()`.
3. Use `DlgShow()` with a list of arguments, one per dialog field, to get and set the editable field contents. As there is a limit on the number of arguments allowed in the script language, you can also use arrays as arguments; an array with `n` elements matches `n` fields in the dialog.

Version 5 extensions
From version 5 onwards you can add and manipulate buttons, collect a time or x axis value and add a group box. You can define script functions that are called in response to button presses and user changes to the dialog and an idle-time function that is called repeatedly whilst the dialog is waiting for user actions. All these functions can enable and disable, hide and show and modify dialog items. `DlgChan()` can now get a channel from an XY view. You can add tool tips for all the prompts in a dialog to give users a more detailed explanation of a field. Finally, there are extensions to the integer, real and string fields that allow you to define a drop-down list of selectable items to copy into the fields and you can add a spin control to both integer and real numeric fields. All dialogs created in previous versions of Spike2 should work without any change.

Version 6 extension
We added `DlgGetPos()` at version [6.06] to get the dialog position. Most users like to choose where a dialog is positioned; this lets you find out where it has been placed and restore it the next time.

Version 7 extension

The `DlgMouse()` function was extended at [7.01] to link mouse movements and button clicks to user-defined functions while your dialog is active in exactly the same way as for the `ToolbarMouse()` command.

The `DlgSlider()` function was added at [7.07] plus the ability for the x positions of dialog items to be set negative, meaning position relative to the right-hand edge of the dialog.

Dialog units

Positions within a dialog are set in dialog units. In the x (horizontal) direction, these are in multiples of the maximum width of the characters '0' to '9'. In the y (vertical) direction, these are in multiples of the line spacing used for simple dialogs. The maximum y position is 40. You are allowed to use negative x positions, meaning place the right hand edge of the item this many dialog units in from the dialog right edge. There is a 2 dialog unit border to left and right of the dialog, so the standard x position (if x is omitted or 0) is 2. and the standard indent from the right is -2. Unless you intend to produce complex dialogs with user-defined positions, you need not be concerned with dialog units at all.

Default vertical positions

When an item is added to the dialog, it gets a default position (before any position you set is applied). In the vertical direction, this is the item number. Item number 1 goes at the top of the dialog. If you supply your own vertical position, we expect it to be at least 1. If you set a vertical position that is less than or equal to 0 it will be assumed that you want to keep the default position, which is often not be what you intended. For future compatibility, if you want the default position, use 0. Items without an explicit item number, for example `DlgText()`, `DlgImage()` and `DlgGroup()`, can have unexpected default positions.

Simple dialog example

The simple example dialog shown above can be created by this code:

```
var ok%, item1%, item2%, item3, item4$:= "more...", item5, item6;
DlgCreate ("Title for the dialog");        'start new dialog
DlgInteger(1,"Item &1 prompt",0,10,0,0,1); 'range 0-10,spinner
DlgChan   (2, "Item &2 prompt|Tip", 1);    'Waveform channel list
DlgReal   (3, "Item &3 prompt", 1.0, 5.0); 'real, range 1.0-5.0
DlgString (4, "&More prompts...|Tip", 6);  'string, any characters
DlgCheck  (5, "A &check box with a prompt");'a check box item
DlgButton (2, "&Extra||Tooltip");          'button 2, tooltip
DlgSlider (6, "A 0-100 slider", 0, 100, 10, 5);'a slider with ticks

ok% := DlgShow(item1%, item2%, item3, item4$, item5, item6); 'show dialog
```

Default button and the `Enter` key

In an active dialog, one field or button has the input focus. If this field accepts user input, it holds a flashing cursor, otherwise the field indicates it has the focus by a change of colour or displaying with a dotted outline. In addition to the input focus, one of the dialog buttons will usually be the *default* button. This is the button that is activated by the `Enter` key. You can temporarily change the default button by using the `Tab` key to give another button the input focus (and become the default), but as soon as any non-button field has the input focus, the original default button will become the default. Currently, button 1 (normally labelled OK), is always the default button. You can set button text and create buttons with the `DlgButton()` command.

Prompts, & and tooltips

In the functions that set an item with a prompt, if you precede a character in the prompt with an ampersand (&), the following character is underlined and is used by Windows as a short-cut key to move to the field or activate the button. All static dialog items except a group box allow you define a tool tip by appending a vertical bar followed by the tool tip text to the `text$` argument. For example:

```
DlgReal(3, "Rate|Enter the sample rate in Hz", 100, 500);
```

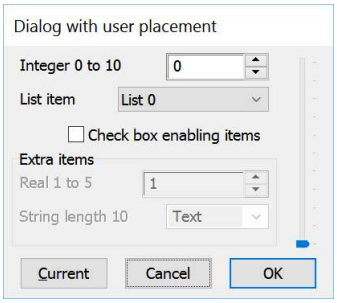
From [10.14] you can include a vertical bar in the prompt by inserting two vertical bars:

```
DlgReal(3, "This || is not a tip|A tooltip", 100, 500);
```

Buttons allow you to specify an additional activation key and an optional tool tip by adding a vertical bar followed by the key code and then another vertical bar followed by the tool tip. See the `label$` argument of `ToolbarSet()` for details of key codes. You cannot have a vertical bar in the button text due to the ambiguity

More complex example

This example shows how to respond to user actions within a dialog. We use a check box to enable and disable a group of items and a button that displays the current values of dialog items. The numbered fields are:



1 An integer, range 0-10 with a spinner
2 A drop list of 4 items
3 A check box, used to enable items 4 and 5
4 A real number with a spinner
5 A string with a drop down list of items
6 A vertical slider positioned on the right

We have added button 2 (buttons 0 and 1 are Cancel and OK) and a group box around items 4 and 5. To make room for the group box, the y positions of items 4 and 5 are set explicitly. To make room for the slider, we have defined a negative x offset and applied it to all the non-button fields.

With `DlgAllow()` we have set `Func Change%(item%)` to be called whenever the user changes a selection or check box or when an editable field loses the input focus. The `item%` argument is set to the item number that changed or to 0 if the dialog is appearing for the first time. We are interested in item 3, the check box, and we use the state to enable or disable the group box and the items inside it.

`Func Current%()` is linked to the `"Current"` button and can be activated with the `C` and `F1` keys and has a tool tip. The function displays a message box that lists the current values of items in the dialog.

```
var ok%, item1%, item2%, item3%, item4, item5$:= "Text", item6, gp%;
var xo := -8;  'Space from right edge for slider
DlgCreate("Dialog with user placement",0,0,40,7.5);
DlgInteger(1,"Integer 0 to 10",0,10,xo,1,1);   'Int with spinner
DlgList(2,"List item","List 0|List 1|List 2|List 3", 4, xo, 2);
DlgCheck(3, "Check box enabling items",xo,3);   'check box item
DlgReal(4, "Real 1 to 5",1.0,5.0,xo,4.5,0.5);  'Real with spinner
DlgString(5, "String length 10",10,"",xo,5.5,  'String item with
            "String 1|String 2|String 3");     'drop-down list
DlgSlider(6, -5.8, 0, 100, 10, 1, -2, 1);       'rhs vert slider
DlgButton(2,"&Current|Ox70|Tooltip", Current%);'button F1+function
DlgAllow(0x3ff, 0, Change%);  'Allow all, no idle, change function
gp% := DlgGroup("Extra items",1,3.8,xo+1,2.9); 'Group box
DlgMouse(-1, -1, 1, 1, MouseDown%, MouseUp%, MouseMove%);
ok% := DlgShow(item1%,item2%,item3%,item4,item5$,item6);
Halt;

Func Change%(item%)
var v%;
docase
  case ((item% = 3) or (item% = 0)) then '0 is initial setup
    v% := DlgValue(3);         'get check box state
    DlgEnable(v%, 4, 5);       'enable items 4, 5
endcase;
return 1;                      'Return 1 to keep dialog running
end;

Func Current%()
var v1%, v2%, v3%, v4, v5$, v6;
v1% := DlgValue(1); v2% := DlgValue(2); 'Retrieve values
v3% := DlgValue(3); v4  := DlgValue(4); v5$ := DlgValue$(5);
v6 := DlgValue(6);
Message("Values are %d, %d, %d, %g, %s and %g",v1%,v2%,v3%,v4,v5$,v6);
return 1;                      'Return 1 to keep the dialog running
end;

func MouseDown%(vh%, chan%, x, y, flags%)
PrintLog("Down: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 1024; 'measurement cursor  + a line linking start to end
```

```
end;

func MouseMove%(vh%, chan%, x, y, flags%)
PrintLog("Move: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 0;       'keep same cursor as for the mouse down
end;

func MouseUp%(vh%, chan%, x, y, flags%)
PrintLog("Up: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 1;       'do not close the toolbar
end;
```

DlgAllow()

Call this function after `DlgCreate()` and before `DlgShow()` ends to enable dialog idle time processing, advanced call-back features and dynamic access to the dialog fields. There are no restrictions on what call-back functions can do. However, it is not sensible to place time-consuming code in an idle call-back function or to do anything other than check dialog fields and possible display a warning message in a dialog-item-change function. Call-back functions use `DlgValue()`, `DlgEnable()` and `DlgVisible()` to manipulate the dialog fields.

`DlgAllow()` cannot be used during `DlgShow()` before version [6.11]. If you open a dialog within a dialog, this commands applies to the currently open dialog until you call `DlgCreate()`, after which it applied to the new dialog until you return from `DlgShow()`.

```
Proc DlgAllow(allow% {,func id%(){, func ch%()}});
```

`allow%` A number that specifies the actions that the user can and cannot take while interacting with Spike2. See `Interact()` for a full description.

`id%()` This is an integer function with no arguments, the Idle function. Use the name with no brackets, for example `DlgAllow(0,Idle%);` where `Func Idle%()` is a script function. When `DlgShow()` executes, the function is called repeatedly in system idle time, as for the `ToolbarSet()` idle function unless a change function, a button function or a Mouse function is active.

If the function return value is greater than 0, the dialog remains open. A zero or negative return value closes the dialog and `DlgShow()` returns the same value.

If this argument is omitted or 0, there is no idle time function.

`ch%()` This is an integer function with one integer argument, for example `Func Changed%(item%)`. You would use `DlgAllow(0,0,Changed%);` to link this function to a dialog. Each time the user changes a dialog item, Spike2 calls the function with the argument set to the changed item number. There is an initial call with the argument set to 0 when the dialog is about to be displayed.

A field is deemed to change when the user clicks a check box or changes a selection in a list or moves the focus from an editable item after changing the text. For real and integer values, the new value must be in range.

If the change function returns greater than 0, the change is accepted. If the return value is zero, the change is resisted and the focus set back to the changed item. If the return value is negative, the dialog closes and `DlgShow()` returns this value and the arguments are not updated.

You can find an example above (more complex example).

DlgButton()

Dialogs created by `DlgCreate()` have Cancel and OK buttons; this function adds, deletes and changes buttons. You can link a script function to a button and use the function return value to decide if the dialog should close. Use this function after `DlgCreate()` and before `DlgShow()`. To change a button label after you call `DlgShow()`, use `DlgValue$()` from a dialog call-back function. There are two command variants:

Create a button and optionally associate it with a script function

```
Func DlgButton(but%, text${, func ff% {, x, y}});
```

`but%` The button number from 0 to 199. Button 0 is the Cancel button, 1 is the OK button and is always the default button. Button numbers higher than 1 create new buttons.

`text$` This sets the button label. Set an empty string to delete a button. You cannot delete button 1; the label is set back to OK if you try. The label text can be followed by an optional key code and an optional tool tip separated by vertical bars. See the `label$` argument of `ToolbarSet()` for details of the format. You cannot include a vertical bar in the text of the button label.

You can delete the Cancel button. You can change the text of the Cancel button, but if you do, it still has the same effect (of closing the dialog) unless you associate a function with the button and return a positive value.

If you set a key code, the button can be activated even when the dialog does not have the input focus as long as it is the topmost user dialog and you have not created a toolbar or interact bar from a function linked to the dialog. This allows you to drag cursors in a window, then use the key code without the need to click in the dialog to activate it first.

`ff%()` This is an integer function with no arguments that is called when the button is used. You must supply the function name only; do not include the left and right round brackets after it or the function will evaluated during the `DlgButton()` call and no function will be linked to the button. Set the argument to 0 (other integer values are reserved for future use) or omit it if you don't want a button function, in which case clicking the button closes the dialog, `DlgShow()` returns the button number and the `DlgShow()` arguments are updated for all buttons except 0.

If you supply a function, it is called each time the button is used and the function return value determines what happens next:

<0 The button acts as Cancel. The dialog closes, `DlgShow()` returns this value and its arguments are not updated.

0 The button acts as OK. The dialog closes and the `DlgShow()` return value is the button number and its arguments are updated.

>0 The dialog continues to display.

The button function can use `DlgEnable()`, `DlgValue()` and `DlgVisible()` and can also create and show subsidiary dialogs.

`x,y` Set the button position in dialog units, both or neither of these must be supplied. If the button position is not supplied it will be positioned at the bottom of the dialog.

Returns 0.

Report last button pressed [7.01]

This allows you to service multiple buttons with the same routine (in place of writing a separate function for each button). You must still call the setup version of the command to link each button with a script function, but you give all the shared buttons the same function name. Within the named function you call `DlgButton()` to get the button number to service.

```
Func DlgButton();
```

Returns The number of the last button in the dialog that was pressed.

Example

```
func DoButton%()
Message("Button %d", DlgButton()); 'report button number
return 1; ' leave dialog open
end;

DlgCreate("Example of button replacement");
DlgText("This dialog demonstrates using a button", 0, 1);
DlgButton(0, "");   ' remove the Cancel button
DlgButton(2, "My &Button||Press to say hello", DoButton%);
DlgShow();  ' no arguments as no variable fields defined
```

There is a simple example here. You can find a more complex example here.

DlgChan()

This function defines a dialog entry that lists channels that meet a specification for time, result and XY views. For simple dialogs, the `wide`, `x` and `y` arguments are not used. Channel lists are checked or created when the `DlgShow()` function runs. If the current view is not a time, result or XY view, the list is empty.

```
Proc DlgChan(item%, text$|wide, mask%|const list%[]{, x{, y}});
```

`item%` This sets the item number in the dialog in the range 1 to the number of items.

`text$` The prompt to display, optionally followed by a vertical bar and tool tip text.

`wide` This is an alternative to the prompt. It sets the width in dialog units of the box used to select a channel. If `wide` is omitted the number entry box has a default width of the longest channel name in the list or 12, whichever is the smaller.

`mask%` This determines the channels to display. Select channel types by adding the following codes, which are given as decimal and hexadecimal.

1	0x1	Waveform or result view channel
2	0x2	Event+ and Event- channels
4	0x4	Event +- channels (level data)
8	0x8	Marker channels
16	0x10	WaveMark data
32	0x20	TextMark data
64	0x40	RealMark data
128	0x80	Unused/deleted disk channels
256	0x100	Deleted channels on disk
512	0x200	Real wave channel

If none of the above values are used or this is an XY view, the list includes all channels. Add the following codes to exclude channels from the list:

1024	0x400	Exclude visible channels
2048	0x800	Exclude hidden channels
4096	0x1000	Exclude time view disk channels but not their duplicates
8192	0x2000	Exclude memory channels but not their duplicates
16384	0x4000	Exclude duplicated channels
32768	0x8000	Exclude selected channels
65536	0x10000	Exclude non-selected channels
2097152	0x200000	Exclude virtual channels but not their duplicates

Finally, the following add special entries to the list and control the display:

131072	0x20000	Add None as an entry in the list, returns 0
262144	0x40000	Add All channels as an entry in the list, returns -1
524288	0x80000	Add All visible channels as an entry, returns -2
1048576	0x100000	Add Selected as an entry in the list, returns -3. This entry displays items in selection order. `ChanList(list%[], 65536)` returns them in channel number order.
4194304	0x400000	Do not display channel type or shorten long XY channel titles

`list%` As an alternative to a mask, you can pass in a channel list (as constructed by `ChanList()`). This must be an array of channel numbers, with the first element of the array holding the number of channels in the list. From version 6.03 you can add the special entries into the first element. For example, if you have 5 channels following and you want to add `None` as an option, use `list%[0]:=0x20000+5`.

`x` If omitted or zero, the selection box is right justified in the dialog box, otherwise positive values set the position of the left end of the channel selection box in dialog units and negative values set the position of the right hand end.

y If omitted or zero, this takes the value of item%. It is the position of the channel selection box in dialog units.

The variable passed to DlgShow() for this field should be an integer. If the variable passed in holds a channel number in the list, the field shows that channel, otherwise it shows the first channel in the list or None if 0x20000 is in the mask. The result from this field in DlgShow() is a channel number, or 0 if None is selected, -1 if All channels is selected, -2 if All visible channels is selected or -3 if Selected is chosen.

There is a simple example here.

DlgCheck()

This defines a dialog item that is a check box (on the left) with a text string to its right. For simple dialogs, the x and y arguments are not used.

```
Proc DlgCheck(item%, text${, x{, y}});
```

item% This sets the item number in the dialog in the range 1 to the number of items.

text$ The prompt to display, optionally followed by a vertical bar and tool tip text.

x,y The position of the bottom left hand corner of the check box in dialog units. If omitted, x is set to 2 and y to item%. When used without these fields, this behaves exactly like the simple dialog functions, and can be mixed with them.

The associated DlgShow() variable should be an integer. It sets the initial state (0 for unchecked, not 0 for checked) and returns the result as 0 (unchecked) or 1 (checked). This item does not have a prompt. If you use DlgValue(), a string value refers to the text and a numeric value refers to the check box state.

There is a simple example here.

DlgCreate()

This function starts the definition of a dialog and clears any DlgAllow() settings. It also kills off any previous dialog that might be partially defined. The scr% and rel% arguments were added at version 6.03.

```
Func DlgCreate(title${, x{, y{, wide{, high{, help{, scr%{, rel%}}}}}}});
```

title$ A string holding the title for the dialog.

x,y Optional, taken as 0 if omitted. The position of the top left hand corner of the dialog as a percentage of the rectangle defined by scr% and rel%. The value 0 means centre the dialog. Values out of the range 0 to 95 are limited to the range 0 to 95.

wide The width of the dialog in dialog units. If this is omitted, or set to 0, Spike2 works out the width for itself, based on the items in the dialog. If this is negative, -wide is the limit for the dialog width (before Spike2 version [8.00], negative widths were treated as 0). In any case, the dialog is limited to the main screen width.

high The height of the dialog in dialog units. If omitted, or set to 0, Spike2 works it out for itself, based on the dialog contents. If this is negative, -high is the limit for the dialog height (before [8.00], negative heights were treated as 0). In any case, the dialog is limited to the main screen height.

help A string or numeric identifier that identifies the help page to be displayed if the user requests help when the dialog is displayed. Use -1 for no defined help page. This is unlikely to be useful unless you create your own Help file and then use the Help() script command to change the default help file.

scr% Optional screen selector for views, dialogs and the application window. If omitted or -1, positions are relative to the application window. Otherwise, 0 selects the entire desktop rectangle and greater values select a particular screen rectangle (but see rel%). See System() for more screen information.

rel% Ignored unless scr% is 0 or greater. Set 0 or omit for positions relative to the intersection of the rectangle set by scr% and the application window, 1 for positions relative to the scr% rectangle. If there is no intersection, there is no position change.

Returns This function returns 0 if all was well, or a negative error code.

For simple use, only the first argument is needed. The remainder are for use with more complicated menus where precise control over menu items is required.

Maximum dialog size

From Spike2 version 8, the dialog size is limited to the size of the primary monitor attached to the system. Previously, the maximum size had an arbitrary limit of 180x40 in dialog units. If either dimension of the primary screen is less than the previous limits, we allow the old limits for reasons of backwards compatibility.

Use of & in prompts

In the functions that set an item with a prompt, if you precede a character in the prompt with an ampersand "`&`", the following character is used by Windows as a short-cut key to move to the field and the character is underlined. Ampersand characters are ignored on systems that do not use this mechanism.

DlgEnable()

Use this only from a dialog call-back function to enable or disable dialog items. With one argument, it returns the enabled state of an item; with two or more arguments it sets the enabled state of one or more items. Prompts and spin controls associated with the item are also enabled or disabled. When applied to a bitmap, disable hides the bitmap and enable shows it. `DlgEnable()` was new in version 5. Note that disabling a group box has no visible effect in modern versions of Windows.

```
Func DlgEnable(en%, item%|const item%[]{, item%|const item%[]...});
Func DlgEnable(item%);
```

`en%` Set 0 to disable items in the list, 1 to enable them and 2 to enable and give the first item the input focus. Input focus changes should be used sparingly to avoid user confusion; they can cause button clicks to be missed.

`item%` An item number or an array of item numbers of dialog elements. The item number is either the number you set, or the number returned by `DlgText()` or `DlgGroup()`, or `-button`, where `button` is the button number. You cannot access prompts separately from their items as this makes no sense.

Returns When called with a single argument it returns the enabled state of the item, otherwise it returns 0.

You can find an example here.

DlgFont()

Before Spike2 version 8, the font used in user-defined dialogs and for the `Input()`, `Input$()` and `Query()` commands was the system `ANSI_VAR_FONT`, which is usually MS Sans Serif 8pt. This is a proportionally spaced raster font, which does not look as nice as a TrueType or ClearType font as it does not take advantage of the ability to use fractional pixel positions on modern displays. However, by sticking with this font, user-defined dialogs look the same and layout the same across all windows platforms.

The `DlgFont()` script command gives you two more choices of font for user-defined dialogs, but with the penalty that you cannot be certain that complex user-defined dialogs will layout in the same way in different versions of Windows or with different user options. The font used by a user-defined dialog is set when the `DlgShow()` command is executed, so `DlgFont()` must be run before this. Each time a script starts the default font is reset, equivalent to `DlgFont(0)`. It is possible that in the future we will provide a Preferences option to change the default.

```
Func DlgFont({font%})
```

`font%` An optional argument to set the desired font policy. If omitted, no change is made. There are three options:

- 0 This selects the original `ANSI_VAR_FONT`. Before Spike2 version [8.03], this was set each time you ran a script.
- 1 This sets the same font that Spike2 uses for dialogs (such as the About Spike2 dialog). This is usually an 8 pt TrueType font (often Tahoma), so will look sharper than option 0 and will usually have a similar spacing. This is the default from [8.03] unless you set the Compatibility option to use the `ANSI_VAR_FONT`.

2 This sets the font that Windows is using for system dialogs.

Returns The font code that was in use at the time of the call (0, 1 or 2).

Setting option 1 will usually not cause too much difference in layout and the result should look better than option 0. Setting option 2 is more problematic as the default since Vista is Segoe UI 9 pt, and this is likely to cause layout incompatibilities, especially if you make complex dialogs.

See also:
Dialogs, `DlgShow(), Input(), Input$(), Query()`

DlgGetPos()

This can only be used from a dialog call-back function to get the position of the top left corner of the dialog. The call-back functions are set by `DlgAllow()` and `DlgButton()`. To get the final dialog position you need to have call-backs for any button press that would close the dialog, or in an idle routine. This function was added at Spike2 version [6.06]. The optional `wide` and `high` arguments were added at Spike2 version [10.0].

```
Func DlgGetPos(&x, &y{, scr%{, rel%{, wide{, high}}}});
```

`x,y` Returned holding the position of the top left hand corner of the dialog relative to the rectangle defined by `scr%` and `rel%`. The values are a percentage of the rectangle dimensions.

`scr%` Optional screen selector for views, dialogs and the application window. If omitted or -1, positions are relative to the application window. Otherwise, 0 selects the entire desktop rectangle and greater values select a particular screen rectangle (but see `rel%`). See `System()` for more screen information.

`rel%` Ignored unless `scr%` is 0 or greater. Set 0 or omit for positions relative to the intersection of the rectangle set by `scr%` and the application window, 1 for positions relative to the `scr%` rectangle. If there is no intersection, the return value is -1.

`wide` If present, returned as the width of the dialog in dialog units.

`high` If present, returned as the height of the dialog in dialog units.

Returns 1 if the position was returned, or -1 if the rectangle set by `scr%` and `rel%` is of zero size.

Note
We observe that top level windows (which includes dialogs) can have a semi-transparent border that is used to generate a shadow. The `x,y` position that is returned allows for any such border, so the returned values may appear a little smaller than you would expect. If they become 0 or negative, this can cause the dialog to be centred when used with the `DlgCreate()` command.

DlgGroup()

This routine creates a group box, which is a rectangular frame with a text label at the top left corner. You can use this between calls to `DlgCreate()` and `DlgShow()`. There is nothing for the user to edit in this item, so you do not supply an item number and there is no matching argument in `DlgShow()`. However, the returned number is an item number (above the values used to match items to `DlgShow()` arguments) that you can use in call-back functions to identify the group box. This function was added at version 5.

```
Func DlgGroup(text$, x, y, width, height);
```

`text$` The text to display at the top left of the group box. Any tool tip text is ignored.

`x,y` The position of the top left corner of the group box.

`width` If positive, the width of the group box in dialog units. If negative, this is the offset of the right hand side of the group box from the right hand edge of the dialog.

`height` The height of the group box in dialog units.

Returns The routine returns an item number so that you can refer to this in call-back functions to use `DlgVisible()` and `DlgEnable()`.

You can find an example here.

DlgImage()

Call this function after DlgCreate() and before DlgShow() to add an image to a dialog.

```
Func DlgImage(bmp$, x, y, width, height);
```

bmp$ The path to a suitable file holding an image. Images are read from .bmp, .jpg, .jpeg, .png, .tif or .tiff files on disk. The image is stretched to fit in the rectangle set by the x, y, width and height arguments.

x,y The position of the top left corner of the bitmap in dialog units. If the y value is less than 1, a default value is used, which depends on the number of items that have been defined in the dialog before this one. If you really want the default value, set a y value of 0 for future compatibility.

width If positive, the width of the image in dialog units. If negative, this is the offset of the right hand side of the image from the right hand edge of the dialog.

height The height of the bitmap in dialog units.

Returns The routine returns an item number so that you can refer to this in call-back functions to use DlgVisible() and DlgEnable().

DlgInteger()

This function defines a dialog entry that edits an integer with an optional spin control or drop down list of selectable items. The numbers you enter may not contain a decimal point. For simple dialogs, the wide, x, y, sp% and li$ arguments are not used. The sp and li$ arguments were new in version 5.

```
Proc DlgInteger(item%, text$|wide, lo%, hi%{, x{, y{, sp%|li$}}});
```

item% This sets the item number in the dialog in the range 1 to the number of items.

text$ The prompt to display, optionally followed by a vertical bar and tool tip text.

wide This is an alternative to the prompt. It sets the width in dialog units of the box in which the user types the integer. If the width is not given the number entry box has a default width of 11 digits (or width needed for the number range?).

lo% The start of the range of acceptable numbers.

hi% The end of the range of acceptable numbers.

x If omitted or zero, the number entry box is right justified in the dialog. Otherwise positive values set the position of the left side of the box relative to the left-hand side of the dialog in dialog units and negative values set the right side position relative to the right-hand side of the dialog.

y If omitted or zero, this takes the value of item%. It is the position of the bottom of the number entry box in dialog units.

sp% If present and non-zero, this adds a spin box with a click increment of sp%. You can change this value dynamically with DlgValue().

li$ If present, this argument is a list of items separated by vertical bars that can be selected into the integer field, for example "1|10|100".

The variable passed into DlgShow() as argument number item% should be an integer. The field starts with the value of the variable if it is in the range. Otherwise, it is limited to the nearer end of the range. You can find an example of DlgInteger() use here.

DlgLabel()

This function sets an item with no editable part that is used as a label. For simple dialogs, the x and y arguments are not used. You can add text to a dialog without using an item number with DlgText().

```
Proc DlgLabel(item%, text${, x{, y}});
```

item% This sets the item number in the dialog in the range 1 to the number of items.

text$ The prompt to display, optionally followed by a vertical bar and tool tip text.

x If omitted or zero, the text is left justified in the dialog. Otherwise positive values set the position of the left side of the text relative to the left-hand side of the dialog in dialog units and negative values set the right side position relative to the right-hand side of the dialog.

y If omitted or zero, this takes the value of item%. It is the position of the bottom of the text in the dialog in dialog units.

When you call DlgShow(), you must provide a dummy variable for this field. The variable is not changed and can be of any type, but must be present.

DlgList()

This defines a dialog item for a one of n selection. Each of the possible items to select is identified by a string. For simple dialogs, the wide, x and y arguments are not used.

```
Proc DlgList(item%, text$|wide, const list$[]|list${, n%{, x{, y}}});
```

item% This sets the item number in the dialog in the range 1 to the number of items.

text$ The prompt to display, optionally followed by a vertical bar and tool tip text. The list box width is the smaller of the longest string in the list or 20.

wide This is an alternative to the prompt. It sets the width of the box in which the user selects an item. If you need a prompt you can use DlgText().

list$ A string of list items separated by a "|" character or an array of strings, one per item. Long strings are truncated. The "|" method was added in version 3.

n% The number of entries to display. If this is omitted, or if it is larger than the number of entries provided, then all the entries are displayed.

x If omitted or zero, the number list box is right justified in the dialog. Otherwise positive values set the position of the left side of the box relative to the left-hand side of the dialog in dialog units and negative values set the right side position relative to the right-hand side of the dialog.

y The selection box vertical position in dialog units. If omitted, the value of item% is used.

The result obtained from this is the index into the list of the list element chosen. The first element is number 0. The variable passed to DlgShow() for this item should be an integer. If the value of the variable is in the range 0 to n-1, this sets the item to be displayed. Otherwise, the first item in the list is displayed.

The following example shows how to set a list, building the contents using an array of strings:

```
var ok%,which%:=0;                'string list, test for OK, result
DlgCreate("List example");        'Start the dialog
var list$[3];                     'these strings are the choices
list$[0] := "one"; list$[1] := "two"; list$[2] := "three";
DlgList(1,"Choose", list$[]);     'Add the list to the dialog
ok% := DlgShow(which%);           'Display dialog, wait for user
```

Alternatively, for a fixed list, you can replace the third, fourth and fifth lines with:

```
DlgList(1,"Choose","one|two|three"); 'version 3 onwards
```

DlgMouse()

Use after DlgCreate() and before DlgShow(). There are two variants of the command; the first sets the initial mouse pointer position when the dialog opens. If you do not use this command variant, the mouse pointer is not moved when the dialog opens. This command can be particularly useful if you are using multiple screens.

```
Proc DlgMouse(item%);
```

`item%` The item number of an element of the dialog at which to position the mouse pointer when the dialog opens. This is either the number you set for the dialog item, or the number returned by `DlgText()` or `DlgGroup()`, or `-button`, where `button` is the button number.

The second variant gives you access to the mouse positions and left button mouse clicks in Time, Result and XY views when the mouse is over a data channel while a dialog is active. This functions in exactly the same way as the `ToolbarMouse()` command, so see there for full details and a description of the mouse down, up and move functions. The command arguments are:

```
Proc DlgMouse(vh%, ch%, mask%, want%, Func Down%{, Func Up%{, Func Move
%}});
```

DlgProgress()

This routine creates a progress bar within a dialog; a rectangular area that is normally used to give a visual impression of what fraction of a task has been completed. You can use this between calls to `DlgCreate()` and `DlgShow()`. There is nothing for the user to edit in this item, so you do not supply an item number and there is no matching argument in `DlgShow()`.

The returned value is an item number (greater than the range of values used to match items to `DlgShow()` arguments) that you can use in call-back functions to identify the progress bar. The progress bar appearance is controlled by values set with the `DlgValue()` function. To make sensible use of a progress bar you will have an idle time function that makes calls to `DlgValue()` to indicate progress through the operation.

This command was added at Spike2 version [9.03].

```
Func DlgProgress(lb, rt, flags%, x, y, width{, height});
```

`lb` Sets the values corresponding to the left/bottom end of the progress bar. `lb` can be greater than `rt` but must not be the same. When you use `DlgValue()` to set the progress, values in the range `lb` to `rt` cause the progress bar to fill from left to right.

`rt` Sets the values corresponding to the right/top end of the progress bar. `lb` can be greater than `rt` but must not be the same.

`flags%` This sets display options and is the sum of:

- 1 Allow tooltip value readout of the bar position when the mouse hovers over the progress bar.
- 2 Use an old-style progress bar display where the bar is broken up into a series of blocks; this appears to have no effect in some versions of Windows.
- 4 Treat all progress bar values as integers, so ignore fractional parts of `lb`, `rt` and values passed with `DlgValue()`.
- 8 Set a progress bar that indicates progress vertically from bottom to top; without this it runs horizontally from left to right.

`x` This sets the horizontal position of the top left corner of the progress bar relative to the left-hand side of the dialog in dialog units.

`y` This sets the vertical position of the progress bar in dialog units.

`width` If positive, the width of the progress bar. If negative, this is the offset of the right hand side of the progress bar from the right hand edge of the dialog.

`height` The height of the progress bar; omit for a reasonable default value.

Returns The routine returns an item number so that you can refer to this in call-back functions in order to use `DlgVisible()` and `DlgValue()`.

Tooltips

If you enable the tooltip value readout option in `flags%`, the current position of the progress bar appears in any tip when the mouse is over the bar.

DlgReal()

This function defines a dialog entry that edits a real number. For simple dialogs, the `wide`, `x` and `y` arguments are not used. The `sp` and `li$` arguments were new in version 5. The item value can be adjusted dynamically with `DlgValue()`.

```
Proc DlgReal(item%, text$|wide, lo, hi{, x{, y{, sp|li${, pre%}}}});
```

`item%` This sets the item number in the dialog in the range 1 to the number of items.

`text$` The prompt to display, optionally followed by a vertical bar and tool tip text.

`wide` This is an alternative to the prompt. It sets the width in dialog units of the box in which the user types a real number. If `wide` is not given the box has a default width of 12 digits.

`lo,hi` The range of acceptable numbers.

`x` If omitted or zero, the number entry box is right justified in the dialog. Otherwise positive values set the position of the left side of the box relative to the left-hand side of the dialog in dialog units and negative values set the right side position relative to the right-hand side of the dialog.

`y` The number edit box vertical position in dialog units. If omitted, the value of `item%` is used.

`sp` If present and non-zero, this adds a spin box with a click increment of `sp`. You can change this value dynamically with `DlgValue()`.

`li$` If present, this argument is a list of items separated by vertical bars that can be selected into the editing field, for example `"1.0|10.0|100.0"`.

`pre%` If present, this sets the number of significant figures to use to represent the number in the range 6 (the default) to 15.

The variable passed into `DlgShow()` should be a real number. The field will start with the value of the variable if it is in the range, otherwise the value is limited to `lo` or `hi`.

There is a simple example here.

Significant figures and field width

When designing dialogs with real number fields you must be aware that you need enough field width to display the number. In the worst case, a small negative number with 6 (`pre%`) significant figures, for example `-1.23456e-13` will require 12 characters (`pre%`+6). If you add a spinner to the field, this uses around 3 characters of the available space. In the more usual case of the number not requiring an exponent, a positive number will usually need `pre%`+1 characters (`1.23456`) and a negative number `pre%`+2 (`-1.23456`). However, large and small numbers can use more space without switching to exponent notation: (`-1234560, -0.00123456`).

If you need to display large or small values in your dialogs, do not set the field width too small.

Spinner behaviour

If you set a spinner, the up and down buttons (or `Up` and `Down` arrow keys) will change the field value to the next multiple of the `sp` value in the direction implied by the key. If you hold the button/key down, the increment increases. From version [10.20a], if the increment is too small compared to the current value to be seen with `pre%` significant digits, the step size is increased to be large enough to be visible.

DlgShow()

This function displays the dialog you have built and returns values from the fields identified by item numbers, or makes no changes if the dialog is cancelled. While the dialog is displayed and waiting for user input the system is given idle time and if you have used `DlgAllow()` to create idle time call-backs to the script, these will be active.

Once the dialog has closed, all information about it is lost. You must create a new dialog before you can use this function again.

```
Func DlgShow(&item1|item1[], &item2|item2[], &item3|item3[] ...);
```

item For each dialog item with an item number, you must provide a variable of a suitable type to hold the result. It is an error to use the wrong variable type, except an integer field can have a real or an integer variable. Items created with `DlgLabel()` must have a variable too, even though it is not changed.

The variables also set the initial values. If an initial value is out of range, the value is changed to the nearest legal value. In the case of a string, illegal characters are deleted before display.

In addition to passing a simple variable, you can pass an array. An array with `n` elements matches `n` items in the dialog; this allows you to have many more items in a dialog than the argument limit for a function call. The array type must match the items.

Returns 0 if the user clicked on the Cancel button, or 1 if the user clicked on OK. You can also add your own buttons with the `DlgButton()` command and these define their own return values. If the dialog closes due to a user-defined mouse up function defined by `DlgMouse()` returning 0 or a negative number, the return value is the negative code or a value greater than any button number if the mouse up function returned 0.

If the user clicks on OK or a user-defined button that returns 0, all the variables are updated to their new values. If the user clicks on Cancel or a user-defined button that returns a negative value, the variables are not changed.

There is a simple example here.

DlgSlider()

This command adds a slider control that sets an integer value between two user-defined limits. The slider is horizontally orientated for simple use but can be vertically orientated. This command was new in Spike2 version [7.07].

```
Proc DlgSlider(item%, text$|width, lb, rt{, iTick{, flags%{, x{, y}}}});
```

`item%` This sets the item number in the dialog in the range 1 to the number of items.

`text$` A left-justified prompt to display, optionally followed by a vertical bar and tool tip text. If `text$` is used, the slider will be horizontal and fill the space from the end of the prompt to the right hand side of the dialog.

`wide` This is an alternative to the prompt. If positive, the slider is horizontal and the value sets the slider width in dialog units; a zero value uses the entire dialog width. A negative value sets the height of a vertical slider and you will need to provide the `x` and `y` values to complete the positioning.

`lb,rt` Sets the values corresponding to the left/bottom and right/top end of the slider. `lb` can be greater than `rt` but must not be the same.

`iTick` Optional, with a default value of 0. Enables ticks if not zero, values greater than 0 set the tick spacing and negative values set tick auto-scaling in a 1, 2, 5 sequence.

`flags%` This sets display options and is the sum of: 1=tool tip value readout during drag, 2=change notifications during drag operation, 4=round all slider values to integers.

`x` If omitted or zero, the slider is right justified in the dialog. Otherwise positive values set the position of the left side of the slider relative to the left-hand side of the dialog in dialog units and negative values set the right side position relative to the right-hand side of the dialog.

`y` If omitted or zero, this takes the value of `item%`. It is the vertical position of the number entry box in dialog units.

The variable passed into `DlgShow()` as argument number `item%` should be a real. The slider starts with the value of the variable if it is in the range. Otherwise, it is limited to the nearer end of the range. Note that the range of values that a slider can return is quantised by the pixel positions that the slider occupies but the returned variable only changes if the slider position is moved or if the original value is out of the slider range.

Tool tips
If you enable the tool tip value readout option in `flags%`, the position of the slider in the control appears in any tip when the mouse is over the slider. If you have set a tip with the `text$` argument, this tip appears when the mouse is over the prompt. If you add 4 to `flags%`, all displayed positions are integral.

Change function
If you use `DlgAllow()` to set a change function, the `flags%` value determines if it is called every time the slider position changes during a drag operation or only when the drag operation ends.

Keyboard control
The cursor left, right, up and down keys move the slider when it is has the keyboard focus.

DlgString()

This defines a dialog entry that edits a text string. You can limit the characters that you will accept in the string. For simple dialogs, the `wide`, `x` and `y` arguments are not used.

```
Proc DlgString(item%, text$|wide, max%{, legal${, x{,y{,sel$}}}});
```

`item%` This sets the item number in the dialog in the range 1 to the number of items.

`text$` The prompt to display, optionally followed by a vertical bar and tool tip text.

`wide` This is an alternative to the prompt. It sets the width in dialog units of the box in which the user types the string. If the width is not given the number entry box has a default width of `max%` or 60, whichever is the smaller.

`max%` The maximum number of characters allowed in the string.

`legal$` A list of acceptable characters. See `Input$()` for a full description. If this is omitted, or an empty string, all characters are allowed.

`x` If omitted or zero, the string entry box is right justified in the dialog. Otherwise positive values set the position of the left side of the box relative to the left-hand side of the dialog in dialog units and negative values set the right side position relative to the right-hand side of the dialog.

`y` If omitted or zero, this takes the value of `item%`. It is the vertical position of the string entry box in dialog units.

`sel$` If this string is present, it should hold a list of items separated by vertical bars, for example `"one|two|three"`. The field becomes an editable combo box with the items in the drop down list. This was added at version 5.

The result from this operation is a string of legal characters. The variable passed to `DlgShow()` should be a string. If the initial string set in `DlgShow()` contains illegal characters, they are deleted. If the initial string is too long, it is truncated.

There is a simple example here.

DlgText()

This places non-editable text in the dialog box. This is different from `DlgLabel()` as you do not supply an item number and it does not require a variable in the `DlgShow()` function. It returns an item number (higher than item numbers for matching arguments in `DlgShow()`) that you can use to identify this field in call-back functions, for example `DlgVisible()`. There was no returned value in Spike2 version 4.

```
Func DlgText(text$, x, y{, wide});
```

`text$` The prompt to display, optionally followed by a vertical bar and tool tip text.

`x,y` The position of the bottom left hand corner of the first character in the string, in dialog units. Set `x` to 0 for the default label position (the same as `DlgLabel()`). Otherwise positive x values set the position

of the left side of the text relative to the left-hand side of the dialog and negative values set the right side position relative to the right-hand side of the dialog.

`wide` Normally, the width of the field is set based on `text$`. This optional argument sets the width in dialog units. This allows you to replace the text with a longer string from a call-back function.

Returns An item number to identify this field for call-back functions.

DlgValue() and DlgValue$()

These functions can only be used from a dialog call-back function to get and optionally set the value of an item, spinner increment, progress bar, item prompt or button text. You can use this to change the allowed range of values for a numeric control (real or integer value). Call-back functions are established with `DlgAllow()`, `DlgMouse()` and `DlgButton()`.

Get/set the value of an item, spinner increment or progress bar

```
Func DlgValue(item%{, value});
```

`item%` This identifies the dialog item. For items with arguments in `DlgShow()`, use the `item%` value you set to create the field. To access the spinner increment for `DlgReal()` and `DlgInteger()` fields, add 2000 to the item number.

`value` This optional argument holds the new item value for numeric fields or to set a check box, spinner increment or a 0-based item number in a list.

Returns The returned value is the current value of the item before any change

Set the range of a numeric (Real or Integer) field [10.05] onwards

This works with `DlgReal()` and `DlgInteger()` fields and has no effect on other field types. If the current value exceeds the range, the current value changes to the nearer end of the range.

```
Func DlgValue(item%, low, high{, pre%});
```

`item%` The `DlgReal()` or `DlgInteger()` item number.

`low` Sets the low end of the allowed range.

`high` Sets the high end of the range. If `high` < `low`, no change is made. A field with `high = low` cannot be modified.

`pre%` Sets the display precision, as for the `DlgReal()` command. Ignored and should be omitted for integer fields.

Returns The returned value is the current numeric field value before any change (the value is limited to the range you set).

Get/set a text item, group text, button label, prompt or X Value as text

```
Func DlgValue$(item%{, value$});
```

`item%` This identifies the dialog item. For items with arguments in `DlgShow()`, use the `item%` value you set to create the field. To access the prompt for an item add 1000 to the item number. For items created with `DlgText()` and `DlgGroup()`, use the returned item number. For buttons use minus the button number.

`value$` This optional argument holds the new item value. This sets the text for a prompt, button label or the text of an editable control or to select the first matching item in a list box. From [10.07, 9.12, 8.20], in an X Value item, if it matches an item in the drop-down list, this is selected, otherwise the text is displayed if it is a valid x axis dialog expression. It is up to you to make sure the text is acceptable for editable items.

Returns The returned value is the current value of the item before any change.

If there is a problem running the command, for example if the item does not exist, or an argument type is not appropriate for an item, the result is an empty string or the value 0.

You can find an example here.

DlgVisible()

This can only be used from a dialog call-back function to show or hide dialog items. There are two versions of this command. The version with a single argument returns the visible state of an item; the version with two or more arguments sets the visible state of one or more dialog items. When you show or hide an item, any prompt or spin control associated with the item is also shown or hidden. This function was new in version 5.

```
Func DlgVisible(show%, item%|const item%[]{, item%|const item%[]...});
Func DlgVisible(item%);
```

`show%` Set this to 1 to show the items in the list and to 0 to hide them.

`item%` An item number of an element of the dialog or an integer array containing a list of item numbers. The item numbers are either the number you set, or the number returned by `DlgText()` or `DlgGroup()`, or `-button`, where `button` is the button number. You cannot access prompts separately from their items as this makes no sense.

Returns When called with a single argument it returns the visible state of the item, otherwise the return value is 0.

DlgXValue()

This creates an editable combo box to collect an x axis value for the current time, result or XY view. The combo box drop-down list is populated with cursor positions and other window values when `DlgShow()` runs. If the current view is not suitable, the list is empty. This control accepts expressions, for example: `(Cursor(1) +Cursor(2))/2`. The matching `DlgShow()` argument is a real number to hold a time in seconds for a time view, or an x axis value for other views. This command was added at version [5.00].

```
Proc DlgXValue(item%, text$|wide{, x{, y}});
```

`item%` This sets the item number in the dialog in the range 1 to the number of items.

`text$` The prompt to display, optionally followed by a vertical bar and tool tip text.

`wide` This is an alternative to the prompt. It sets the width in dialog units of the combo box. If the width is not given the combo box has a default width of 18 numbers.

`x` If omitted or zero, the string entry box is right justified in the dialog. Otherwise positive values set the position of the left side of the box relative to the left-hand side of the dialog in dialog units and negative values set the right side position relative to the right-hand side of the dialog.

`y` If omitted or zero, this takes the value of `item%`. It is the vertical position of the string entry box in dialog units.

Draw()

This optionally positions the current view and allows invalid regions to update. `Draw()` with no arguments on a view that is up-to-date should make no change. The view is not brought to the front. Use `Yield()` to get a text view up to date.

```
Proc Draw({from {, size}});
```

`from` The left hand edge of the window. The meaning of the argument depends on the view type:

Time	The start time in seconds. If the view is in Triggered mode, the displayed times on the x axis are relative to the trigger. From [11.00] you can set the from time as `ViewTrigger(4) + from` to begin the display at the displayed time `from`.
Result	The start position in bins.
XY	The start position in x axis units.

Text-based The line number of the top line to show in the window; the first line is 1. This also scrolls the view horizontally so that the first character position is at the left of the view.

Grid The column at the top left of the view. The first column is 0. Use `YRange()` to set the top row. Note that you can only scroll the view along while a column reaches the right-hand edge of the window so you may not get what you requested. Use `XLow()` to find the result.

For a time window, this is in seconds. For a result view, this is in bins. For an XY view, it is in x axis units. For a text-based view, this sets the top line to display in the view (or as close to the top line as is possible) and scrolls the view horizontally so that the first character position is at the left of the view.

`size` The width of the window in the same units as `from`. A negative `size` is ignored. This argument must be omitted in a text-based or grid view.

With two arguments, the width is set (unless it is unchanged) and then it is drawn. With one argument, the view scrolls by an integral number of pixels such that `from` is in the first pixel. For example, in a Time view, the following may not move the display if a pixel is more than a second wide:

```
Draw(XLow()+1.0); 'This may scroll by less than 1 second
```

Time views run from time 0 to the maximum time in the view. Result views have a fixed number of bins, set when they are created. XY view axes can be any positive length. The length of a text view is set by lines. You can convert the current position of the selection to lines and columns using `MoveBy()`.

See also:
`DrawAll(), XRange(), XLow(), XHigh(), Maxtime(), MoveBy()`

DrawAll()

This routine updates all views with invalid regions. Nowadays, calling `Yield()` is a better solution as this also allows the system time to clean up unused resources.

```
Proc DrawAll();
```

See also:
`Draw(), Yield()`

DrawMode()

This sets and reads the channel display mode in a time or result view. You can set the display mode for hidden channels. The first command variant gets and sets a single parameter, the second is for time views, the third is for result views and the fourth is for sonograms. You can also use `DrawModeCopy()` to copy the mode of one channel to another, `MarkShow()` to set the Marker drawing mode and `ChanDecorate()` to set RealMark decorations.

```
Func DrawMode(chan%{, item%{, value});
Func DrawMode(cSpc, mode%{, dotSz%|binSz|sFlag%{, trig%|edge%{, as%}}});
Func DrawMode(cSpc, mode%{, dotSz% {, opt%|err%{, sort}}});
Func DrawMode(cSpc, 9{, fftSz%, wnd%, top, range, xInc%, skip%});
```

`chan%` The channel number used to read back information with a negative `mode%`.

`cSpc` A channel specifier for the channels to apply the drawing mode to. Setting a draw mode for an inappropriate channel number has no effect.

`mode%` If `mode%` is omitted, or negative, see the description of `item%`. Positive values set the display mode. If an inappropriate mode is requested, no change is made. Some modes require additional parameters (for example a bin size). If they are omitted, the last known value is used.

The time view drawing modes are:

0 The standard drawing mode for the channel. This also sets any marker or marker-derived type channels to display marker code 0 in the standard mode for the channel type.

1 Dots mode for events or a waveform. `dotSz%` argument can be used.

2 Lines mode. Shows event data as vertical ticks on a horizontal line. If dotSz% is present and zero, the horizontal line is not drawn (however, using dotsz% to hide horizontal lines is deprecated from 8.06, see the item% = -18 explanation, below).

3 Waveform mode. Draws straight lines between waveform and WaveMark points. No WaveMark codes are displayed for WaveMark channels. Also see item% = -17.

4 WaveMark mode. Only use this is you wish to see WaveMark codes as it takes longer to draw than Waveform mode. Also see item% = -17.

5 Rate mode. binSz sets the width of each bin in seconds.

6 Mean frequency mode. binSz sets the time period.

7 Instantaneous frequency mode. dotSz% and as% can be used.

8 Raster mode. trig% sets the trigger channel, dotSz% is used.

9 Sonogram mode. In this mode the fftSz%, wnd%, top, range, xInc% and skip% arguments are allowed (or can be omitted from the right).

fftSz% The block size for the Fourier Transform used to generate the sonogram. Allowed values are powers of 2 from 16 to 32768. Intermediate values set the next lower allowed value.

wnd% The window applied to the data. 0 = no window, 1 = Hanning, 2 = Hamming. Values 3 to 9 set Kaiser windows with -30 dB to -90 dB side band ripple in steps of 10 dB.

top The signal dB value relative to 1 bit at the ADC input to show as the maximum density output. Signal values above top dB are shown in maximum intensity. Values 0.0 to 100.0 are allowed.

range The range in dB of output data below top that is mapped into the colour scale. Output below (top-range) is shown as minimum intensity. Values greater than 0.0 and up to 100.0 are allowed.

xInc% The number of pixels (1 to 100) to step across the screen before calculating the next set of sonogram values. This is normally 1.

skip% Normally 0. Set 1 to use one data block for each vertical strip of sonogram to reduce the calculation time for large files at the cost of a less representative result. Set 0 to process all data.

10 WaveMark overdraw mode.

11 Same as mode 6, but display rate per minute rather than per second.

12 Same as mode 7, but display rate per minute rather than per second.

13 Cubic spline mode for waveform, WaveMark and RealMark channels. See item% = -17 to draw a single trace of multi-trace WaveMark data.

14 Text mode for TextMark channels. dotSz% is used.

15 State mode for marker channels. sFlag% is used.

16 Skyline mode for waveform, RealWave and RealMark channels.

17 Interval mode for event-based channels. dotSz% and as% can be used. New at [10.05].

For a result view channel the modes are listed below.

0 The standard drawing mode for the result view.

1 Draw as a histogram. err% can be used.

2 Draw as a line. err% can be used.

3 Draw as dots. err% and dotSz% can be used.

4 Draw as a skyline. err% can be used.

8 Draw raster as lines, dotSz% sets the line length, opt% is used.

9 Draw raster as dots, dotSz% sets the dot size, opt% is used.

13 Cubic spline mode. err% can be used.

dotSz% Sets the dot or tick size to use in units of the pen width set for data in the Edit menu Preferences Display tab. 0 is the smallest size available. The maximum size is 10. Use -1 for no size change.

binSz Sets rate histogram bin width and the mean frequency smoothing period.

`sFlag%` For markers in State mode, this sets the additional information to display. 0=no extra information. Add 1 to draw the state number, 2 to display TextMark text.

`trig%` This is the trigger channel for time view raster displays. We assume that you do not want to display a level event channel in raster mode.

`edge%` Sets the edges of level event data to use for mean and instantaneous frequency and rate modes. 0 = both edges (default), 1 = rising edges and 2 = falling edges. This is ignored (and should be set to 0) for non-level event channels.

`as%` Used with instantaneous frequency mode and for RealMark data drawn as a waveform to determine how the data is drawn and measured. 0=Default, 1=Dots, 2=Line, 3=Skyline. From [10.05] it can also be used with WaveMark data drawn as a waveform or in OverdrawWM mode to set Dots mode (SkyLine and Default are treated as Line mode).

`opt%` Used in result view raster channels. The default is 0. Add 1 for horizontal line in raster line mode. Add 2 for y axis as time of sweep or variable value in raster modes. Add 16 to show symbols. For backwards compatibility, if `sort%` is omitted, add 4 to select sort value 1 and 8 to select sort value 2. These values (4 and 8) are not returned when `mode%` is -12, use -15 to get the sort value.

`err%` Result view error drawing style: 0=none (default), 1=1 SEM, 2=2 SEM, 3=SD.

`sort%` Result view raster sort mode. 0=time (default), 1-4 = use sort variables 1 to 4.

`item%` Calls that use the `item%` form of the command (where `item%` is omitted or is negative) are used to read back specific information and to set values if the `value` argument is present (except for `item%` of -1 which is read only). Some of the values you can set with `item%` can be set with the `mode%` command variant (the equivalent argument is in the `name` column). The allowed values of `item%` are:

item%	name	Function
-1	mode%	Or omit `item%` to return the current channel draw mode. You cannot use `value` to set the mode, use one of the `mode%` command variants.
-2	dotSz%	The dot or tick size to use in units of the pen width set for data in the Edit menu Preferences Display tab. 0 is the smallest size available. The maximum size is 10.
-3	binSz	The rate histogram bin width and the mean frequency smoothing period.
-4	trig%	The trigger channel for time view raster displays.
-5	edge%	The edges of level event data to use for mean and instantaneous frequency and rate modes. 0 = both edges (default), 1 = rising edges and 2 = falling edges.
-6	fftSz%	The block size for the Fourier Transform as a power of 2 in the range 16 to 32768. Intermediate values set the next lower allowed value.
-7	wnd%	The window applied to sonogram data. 0 = no window, 1 = Hanning, 2 = Hamming. Values 3 to 9 set Kaiser windows with -30 dB to -90 dB side band ripple in steps of 10 dB.
-8	top	The Sonogram signal dB value relative to 1 bit at the ADC input to show as the maximum density output. Signal values above `top` dB are shown in maximum intensity. Values 0.0 to 100.0 are allowed.
-9	range	The Sonogram range in dB of output data below `top` that is mapped into the colour scale. Output below (`top-range`) is shown as minimum intensity. Values greater than 0.0 and up to 100.0 are allowed.
-10	xInc%	The number of pixels (1 to 100) to step across the screen before calculating the next set of sonogram values. This is normally 1.
-11	skip%	Set 1 to use one data block for each vertical strip of sonogram to reduce the calculation time for large files (but only shows samples of the sonogram). Values 0 (default) and 1 are allowed.
-12	opt%	Return and or set the `opt%` argument used for result view raster channels as described above.
-13	err%	Return and or set the result view error drawing style: 0=none (default), 1=1 SEM, 2=2 SEM, 3=SD.

-14 `sort%` Return and or set the result view raster sort mode. 0=time (default), 1-4 = use sort variables 1 to 4.

-15 `sFlag%` Return and or set the additional information to display for markers in State mode. 0=no extra information. Add 1 to draw the state number, 2 to display TextMark text.

-16 `as%` Used with instantaneous frequency mode and for RealMark data drawn as a waveform to determine how the data is drawn and measured. 0=Default, 1=Dots, 2=Line, 3=Skyline.

-17 - This reports the state of the Single trace check box displayed in the Channel draw mode dialog when displaying a WaveMark channel with multiple traces. Returns 0=Display all traces, 1=Display a single trace. The state is set by the Marker Filter dialog or by `MarkTrace()`.

-18 - Reports whether events drawn in line mode draw the horizontal line. 0=no line, 1=draw it.

`value` Used with the `item%` variant to set a new value for one parameter, leaving all the remainder unchanged.

Returns If a single channel is set it returns the previous `mode%`. For multiple channels or an invalid call, it returns -1. Negative `index%` values return drawing parameters as described above.

See also:
Channel specifiers, `ChanDecorate(), ChanIndex(), Draw(), DrawModeCopy(), MarkShow(), SetEvtCrl(), SetPSTH(), ViewStandard(), XYDrawMode()`

DrawModeCopy()

Apply the drawing mode of a nominated channel to a list of other channels in the current Time or Result view. This was added at Spike2 version [10.05].

```
Func DrawModeCopy(cSpc, iSrc%{, vh%});
```

`cSpc` A channel specifier to set the target channels in the current view.

`iSrc%` The source channel number in the source view. This channel must exist.

`vh%` Optional. If present, the view handle of the source view. If omitted or 0, the current view is used. The view must have the same type as the current view (you cannot apply Result view drawing modes to a Time view channel, and vice versa).

Returns The number of channels that were modified or -1 if a view set by `vh%` is the wrong type or does not exist or -2 if the source channel is not found. If all the target channels already match the source channel draw modes, the result is 0.

See also:
Channel specifiers, `ChanDecorate(), Draw(), DrawMode(), ViewStandard(), XYDrawMode()`

Dup()

This gets the view handle of a duplicate of the current view or the number of duplicates. Duplicated views are numbered from 1 (1 is the original). If a duplicate is deleted, higher numbered duplicates are renumbered. See `WindowDuplicate()` for more information.

```
Func Dup({num%});
```

`num%` The number of the duplicate view to find, starting at 1. You can also pass 0 (or omit `num%`) as an argument, to return the number of duplicate windows (including the original).

Returns If `num%` is greater than 0, this returns the view handle of the duplicate, or 0 if the duplicate does not exist. If `num%` is 0 or omitted, this returns the number of duplicates. The following code illustrates the use of `Dup()`.

```
var maxDup%, i%, dvh%;      'declare variables
maxDup% := Dup(0);          'Get maximum numbered duplicate
for i% := 1 to maxDup% do   'loop round all possible duplicates
   dvh% := Dup(i%);         'get handle of this duplicate
   if (dvh% > 0) then       'does this duplicate exist?
      PrintLog(view(dvh%).WindowTitle$()+"\n");'print window title
   endif;
next;
```

See also:
`App(), View(), WindowDuplicate()`

DupChan()

This returns the number of duplicates of a channel in the current time or result view, or the channel number of the n^{th} duplicate, or it identifies the channel on which a duplicate channel is based, or it identifies which duplicate of an original channel this is. Duplicate channels occupy a range of channel numbers after all other channel types.

```
Func DupChan(chan%{, num%});
```

`chan%` A channel number in the current view. The channel need not exist. If this channel is a duplicate, the command behaves as though you passed the original channel on which the duplicates are based. From version 8.11 onwards, if you pass in 0, the result is the start of the range of channel numbers reserved for duplicates.

`num%` If greater than 0, this is the index of a duplicate channel in the range 1 to the number of duplicates. Use 0 to return the original channel that was duplicated. Use -1 as an argument (or omit `num%`), to return the number of duplicates of the channel. Use –2 as an argument to return which duplicate a channel is: 0 means not a duplicate, 1 upwards is the duplicate index.

Returns If `num%` is greater than 0, this returns the channel number of the duplicate, or 0 if the duplicate does not exist. If `num%` is 0, this returns the channel number that `chan%` was duplicated from or `chan%` if it is the original or `chan%` does not exist. If `num%` is –1 or omitted, this returns the number of duplicates. If `num%` is –2, this returns the duplicate index of `chan%` (0 if not a duplicate, 1 for duplicate a and so on). From Spike2 8.11 onwards, if `chan%` is 0, this returns the start of the range of channels reserved for duplicates in the current view.

See also:
`ChanDuplicate(), Chan()`

E

```
EditCopy()
EditCut()
EditFind()
EditPaste()
EditReplace()
EditSelectAll()
Error$()
Eval()
EventToWaveform()
Exp()
ExportChanFormat()
ExportChanList()
ExportRectFormat()
ExportTextFormat()
```

EditClear()

In a result view, this command zeros the histogram data and removes any raster information. In a text view, this command deletes any selected text. In a Grid view, this command clears selected cells.

In an XY view, this command deletes any data points, leaving the channels empty. It had no effect on XY views before Spike2 version [10.00].

```
Func EditClear();
```

Returns The function returns 0 if nothing was deleted, otherwise it returns the number of items deleted or 1 if the number is not known, or a negative error code.

See also:
`EditCut(), EditFind(), EditPaste()`

EditCopy()

This command copies data from the current view to the clipboard or copies a string to the clipboard or creates a bitmap on the clipboard from an array. The options to create a bitmap were added at Spike2 version 9.06. There are four command variants:

Copy current view

The type of the current view determines what is available for copying. Time views copy as a bitmap, as a scalable image and as text in the format set by `ExportRectFormat()`, `ExportTextFormat()`, `ExportChanFormat()` and `ExportChanList()`. Result and XY views can copy as a bitmap, text and as a scalable image. Multimedia views copy as a bitmap. Text windows copy as text. Info windows copy the current display as text. Grid views copy as text with cells in a row separated by the Tab character and rows separated by end of line (`"\n"`). Cursor range and values copy as text.

```
Func EditCopy({as%});
```

`as%` Sets how to copy data when several formats are possible. If omitted, all formats are used. It is the sum of: 1=Copy as a bitmap, 2=Copy as a scalable image (Windows metafile), 4=Copy as text

Returns It returns 0 if nothing was placed on the clipboard, or the sum of the format specifiers for each format used.

Copy script text

This variant copies script-defined text to the clipboard.

```
Func EditCopy(text$);
```

`text$` A text string to place on the clipboard. This can be useful when you want to move results to a different program.

Returns 0 if nothing was copied, else 4 (text).

Copy real array as a bitmap [9.06]

This takes a real array holding red, green and blue intensities, plus an optional Alpha (opacity) channel to the clipboard as a bitmap.

```
Func EditCopy(const arr[][][]{, flags%});
```

`arr` An array of size `[3][wide%][high%]` to set Red, Green and Blue values, or an array of size `[4][wide%][high%]` to set Red, Green, Blue and opacity. All values are in the range 0 to 1.0; values outside this range are limited to the range. The matrix `arr[0][][]` holds the Red values, `arr[1][][]` holds Green values and `arr[2][][]` holds Blue values. If `arr[3][][]` exists, it holds the opacity values (0 for transparent to 1 for opaque). The colour in `arr[][0]0]` correspond with the top left pixel in the bitmap unless `flags%` bit 0 is set, when it corresponds with the bottom left.

`flags%` A set of flags, taken as 0 if omitted. Currently only flag value 1 is defined which matches `arr[][0][0]` to the bottom left and the y direction runs upwards rather than downwards from the top.

Returns 0 if nothing was copied, else 1 (bitmap).

Copy integer array as a bitmap [9.06]
This copies an integer array holding data organised in bitmap format to the clipboard as a bitmap. Each integer value in the array holds colour values.

```
Func EditCopy(const arr%[][]{, flags%});
```

`arr%` An integer array of size `[wide%][high%]` holding the bitmap image data. Each array element defines one pixel. Bits 0-7 hold the blue value, 8-15 the green and 16-23 the red (all as 0-255). You can also specify the use of an Alpha channel in bits 24-31. Bits above 31 are not used.

`flags%` A set of flags, taken as 0 if omitted. 1 causes the bitmap to be created such that arr[][0][0] refers to the bottom left pixel. Without this flag bit set it refers to the top left pixel. Add 2 to use bits 24-31 of each array element as the Alpha (opacity) channel in the range 0 (transparent) to 255 (opaque).

Returns It returns 0 if nothing was placed on the clipboard, or the sum of the format specifiers for each format used.

See also:
`EditSelectAll()`, `ExportTextFormat()`, `ExportChanFormat()`, `ExportRectFormat()`, `ExportChanList()`, `EditCut()`, `EditImageSave()`, `EditPaste()`

EditCut()

This command cuts data from the current view to the clipboard (as for `EditCopy()`) and deletes the original (if this is possible).

```
Func EditCut({as%});
```

`as%` This optional argument sets how data is cut and copied to the clipboard. Currently only text may be cut. If omitted, all allowed formats are used.

- 1 Cut and copy as a bitmap, has no effect
- 2 Cut and copy as a scalable image (metafile), has no effect
- 4 Cut text to clipboard

Returns 0 if nothing was placed on the clipboard, or the sum of the format specifiers for each format that was used. The only possible values now are 0 and 4.

See also:
`EditCopy()`, `EditClear()`

EditFind()

In a text view, this command searches from the selection point for the next occurrence of the specified text and selects it if found. This is the same as the Find Text dialog.

```
Func EditFind(find${, dir%{, flags%}});
```

`find$` The text to search for. May include regular expressions if 4 or 8 is added to `flags%`.

`dir%` Optional. 0=search backwards, 1=search forwards (default), 2=wrap around.

`flags%` Optional, taken as 0 if omitted. The sum of: 1=case sensitive, 2=find complete words only, 4=simple regular expression search, 8=ECMAScript regular expressions. Before version 9.00, the ECMAScript option did not exist and if regular expressions were in use, backward searches were not allowed.

Returns 1 if found, 0 if not found.

See also:
Find text dialog, `EditReplace()`

EditImageLoad()

This script command, new at [10.11], loads an image from a file and saves it to the clipboard. You can access the image with `EditPaste()` and use it with `ChanImage()` and `InfoImage()` commands.

```
Func EditImageLoad(file$);
```

`file$` The file to load. The file must contain an image in BMP, JPEG, PNG, TIFF or GIFF format.

Returns 0 for success or -1 if the operation failed.

Example

Prompt the user to select a suitable file and copy it to the clipboard:

```
var image$;
var n% := FileList(image$, "Select image file",
          "Image files|*.bmp; *.png; *.gif; *.jpg; *.tif||All files (*.*)|*.*||", 1);
if n% > 0 then
    n% := EditImageLoad(image$);
    if n% then Message("Failed to load %s", image$); halt endif;
    var w%,h%;
    if EditPaste(1, w%, h%) then
        Message("Image is %d x %d pixels", w%, h%);
    endif
endif;
```

See also:

`ArrMapImage()`, `ChanImage()`, `EditImageSave()`, `EditPaste()`, `EditCopy()`, `InfoImage()`

EditImageSave()

This script command, new at Spike2 [9.06], saves a bitmap on the clipboard to a file in a user-defined format. Images may be placed on the clipboard using `EditCopy()` and `ArrMapImage()` and read from a file with `EditImageLoad()`.

```
Func EditImageSave(file${, type%});
```

`file$` The file to save to. We expect you to provide a suitable file extension. If you do not set a file extension, the file is saved as the type set by the `type%` field, and failing that, it is saved as a bitmap (and `.bmp` is appended to the file name).

`type%` Optional (zero if omitted). If present, it forces the type of the output file:

Value	Forced type
0	None. The file type is deduced from the file extension, else it is saved as a bitmap.
1	Standard Windows uncompressed BMP file.
2	JPEG image.
3	PNG image.
4	TIFF image.
5	GIF image.

Return -1 if we failed, 0 for success.

See also:

`ArrMapImage()`, `EditImageLoad()`, `EditPaste()`, `EditCopy()`

EditPaste()

This command can be used to obtain the current clipboard text contents as a text string, or to get a bitmap as an array, or to paste the clipboard into a Text or Grid view. Some of the command features were added at Spike2 version [9.02] and [9.06]. Alpha channel reporting was added at [10.06].

Target	Action
String	The text clipboard contents are copied to the string.
Text view	If the clipboard contains text the current text selection is replaced by the clipboard contents.
Grid view	If the clipboard contains text, the clipboard contents are pasted to the grid, starting at the current cell. The clipboard text can contain multiple cells and uses Tab characters to separate cells horizontally and end of line characters to separate cells vertically. Cells are pasted to the right and below the starting cell. After the paste operation, the rectangle holding the modified cells is selected.
Array	If the clipboard holds a bitmap, you can fill a 3d real array with the Red, Green, Blue components, and if the bitmap supports it, the opacity.

There are several command variants:

Paste text into Text or Grid view

This is the simplest version of the command and is equivalent to typing `Ctrl+V` in the current view.

```
Func EditPaste();
```

Returns The number of characters pasted into a text view or the number of cells pasted into a Grid view.

Get the current clipboard text

This copies any text on the clipboard into a script string.

```
Func EditPaste(&text$);
```

`text$` Any text in the clipboard in returned in this string.

Returns The return value is 1 if the clipboard holds text, 0 if it does not.

Get clipboard information [9.02]

This variant is used to find out what information is on the clipboard and to get the size of any bitmap. Note that returning 2 for Alpha channel was new at [10.06].

```
Func EditPaste(what%{, &wide%, &high%});
```

`what%` This determines the information you are seeking:

- 0 Returns flags for the information on the clipboard in the same format as the `as%` argument of `EditCopy()`. The returned value is the sum of: 1=bitmap, 2=scalable image (Windows metafile), 4=text.
- 1 1 if the clipboard holds a bitmap, 0 if it does not. If it holds a bitmap, `wide%` and `high%` are set to the width and height of the bitmap.

`wide%` Only used when `what%` is 1. It is returned holding the width of the bitmap.

`high%` Only used when `what%` is 1. It is returned holding the height of the bitmap.

Returns The information as described for `what%`.

Get clipboard bitmap image into an array

These variants copy a bitmap from the clipboard into arrays. The first variant with a real array was added at Spike2 version [9.02], the integer array variant and the use of the `flags%` argument was added at version [9.06]. The return of 2 to indicate Alpha channel presence was new at [10.06].

```
Func EditPaste(arr[][][]{, flags%});
Func EditPaste(arr%[][]{, flags%);
```

`arr` An array of size `[3][wide%][high%]` to collect Red, Green and Blue values, or an array of size `[4][wide%][high%]` to collect Red, Green, Blue and opacity. All returned values are in the range 0 to 1.0. The matrix `arr[0][][]` holds the Red values, `arr[1][][]` holds Green values and `arr[2][][]` holds Blue values. If the source bitmap has alpha (opacity) information and the array is large enough, `arr[3][][]` holds the opacity values (0.0 for transparent to 1.0 for opaque). If the source holds no opacity information, all the returned values will be 0. Only array elements that map to pixels in the bitmap are changed. The colours in `arr[][0]0]` correspond with the top left pixel in the bitmap unless `flags%` bit 0 is set, when they correspond with the bottom left.

`arr%` An integer array of size `[wide%][high%]` returned holding the bitmap image data. Each array element defines one pixel. Bits 0-7 hold the blue value, 8-15 the green and 16-23 the red (all as 0-255). Bits 24-31 hold the Alpha channel information (if any and requested), otherwise they are 0. Bits above 31 are not used.

`flags%` A set of flags, taken as 0 if omitted. Set 1 to cause incrementing y indices to run upwards from the bottom rather than downwards from the top. Set 2 with an integer array to collect opacity values, if present. In the real array case, setting the first dimension size of arr greater than 3 is taken as a request for any Alpha channel.

Returns 2 if we got the bitmap and alpha data was requested and present, 1 if we got the data, 0 if there was no suitable bitmap on the clipboard or -1 if the first array dimension size is not supported. Detection of alpha data is by checking for pixels with non-zero transparency information, so an image that it entirely transparent will appear to have no transparency information.

See also:
`EditCopy(), EditClear(), EditCut(), EditCopy(), EditImageSave(), ArrMapImage()`

EditReplace()

In a text view, this command checks if the selection matches a pattern and replaces it if it does, then it searches for the pattern again. This is the same as the Replace Text dialog.

```
Func EditReplace(find${, repl${, dir%{, flags%}}});
```

`find$` The text to search for. May include regular expressions if 4 or 8 is added to `flags%`.

`repl$` Optional replacement text, taken as an empty string if omitted.

`dir%` Optional. 0=search backwards, 1=search forwards (default), 2=wrap around.

`flags%` Optional, taken as 0 if omitted. The sum of: 1=case sensitive, 2=find complete words only, 4=simple regular expression search, 8=ECMAScript regular expression search.

Returns The sum of: 1 if found a new match, 2 if replaced the original selection.

See also:
Replace text dialog, `EditFind()`

EditSelectAll()

This function selects all items in the current view that can be copied to the clipboard. This is the same as the Edit menu Select All option.

```
Func EditSelectAll();
```

Returns It returns the number of selected items that could be copied to the clipboard.

See also:
`EditCopy(), EditClear(), EditCut(), MoveBy(), MoveTo()`

Error$()

This function converts a negative error code returned by a function into a text string.

```
Func Error$(code%);
```

`code%` A negative error code returned from a Spike2 function.

Returns It returns a string that describes the error.

See also:
`Debug(), Eval(), Print$(), PrintLog()`

Eval()

This evaluates the argument and converts the result into text. The text is displayed in the Script window or Evaluate window message area, as appropriate, when the script ends. This overrides any subsequent script runtime error message.

```
Proc Eval(arg);
```

`arg` A real or integer number or a string. If you set an empty string, this cancels the effect of a previous call to `Eval()`, allowing script runtime error messages to display.

If you use `Eval()` it will suppress any run-time error messages as it uses the same mechanism as the error system. A common use of `Eval()` in a script is to report an error condition during debugging, for example:

```
if value<0 then Eval("Negative value"); Halt; endif;
```

Another use of `Eval()` is in the Script menu Evaluate window to see the result returned by a function or expression, as in these examples for use with the Execute button:

```
Eval(FileDelete(myfile$)); ' display 1 or a negative error code
Eval(Error$(-1531));     ' give string for error code if known
```

If you use the Eval(...) button in the Evaluate window, this is implemented by inserting Eval(...) around the last expression on the line.

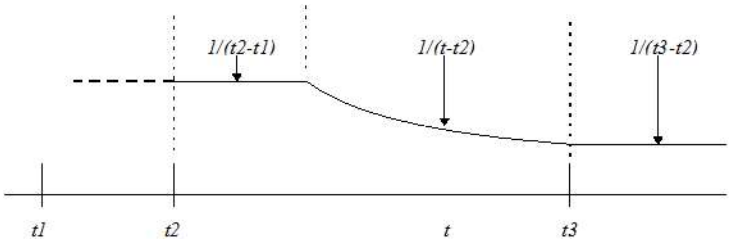
See also:
`Debug(), Error$(), Print(), PrintLog()`

EventToWaveform()

This function creates a waveform channel from an event channel. The waveform channel contents depend on the frequency of the events. You can convert based on instantaneous or smoothed frequency. `VirtualChan()` also converts events to a waveform and can be easier to use; in most cases you only need this function when you want to supply your own smoothing function.

Instantaneous frequency

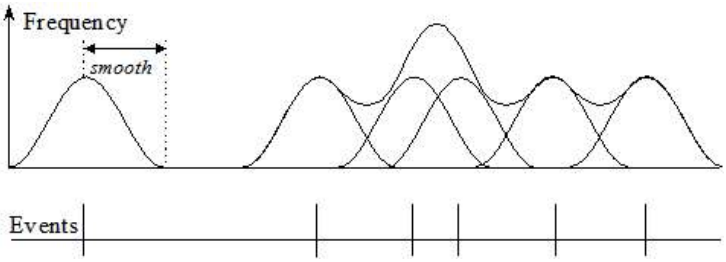
Spike2 calculates instantaneous frequency as the reciprocal of event intervals. Between events, the interval from the last event to the current time is compared with the interval between the last two events. If it is less, the reciprocal of the last interval is used. If it is more, the reciprocal of the interval from the last event is used.



The diagram shows the result for three events at times *t1*, *t2* and *t3*. From *t1* to *t2*, the result depends on events before *t1*. From *t2* to *t2*+(*t2*-*t1*), the result is set by the previous interval. From this point onwards to *t3*, the output reduces until at *t3* it becomes 1/(*t3*-*t2*).

Smoothed frequency

Spike2 calculates smoothed frequency by replacing (convoluting) each event with a waveform of unit area. The built-in waveforms are symmetrical about the event time. If the event is at time *t*, the waveforms extend from time *t-smooth* to *t+smooth*. You can also provide your own waveforms, and these need not be symmetric.



The diagram shows the result of using the built-in raised cosine waveform. The area under the curve for each spike is 1. You would normally use a smoothing period that covered several events to obtain a smooth output.

The command

```
Func EventToWaveform(smooth, eChan%, wChan%, wInt, sTime, eTime,
      maxF, meanF {,query% {,const wave[]|wave%{, nPre}}});
```

`smooth` The smoothing period in seconds. Set this 0 or negative for instantaneous frequency, otherwise an event at time `t` is spread over a time range `t-smooth` to `t+smooth` seconds for symmetric functions. If you omit the `wave` argument, the smoothing function is a raised cosine bell.

`eChan%` The channel number to use as a source of event times. If the channel is a marker, and a marker filter is set, only filtered events are visible.

`wChan%` The channel number to create as a waveform in the range 1 to the maximum allowed channels in the file.

`wInt` The desired sampling interval for the new channel in seconds. Values outside the range 0.000001 to 1000 are errors. This interval is rounded to the nearest multiple of the microseconds per time unit set for the file. Use `BinSize()` to return the actual sampling interval. Before version [4.03], `wInt` was rounded to the nearest multiple of the base ADC convert interval for the file. Older Spike2 versions will not open the file if you set an interval that they cannot achieve.

`sTime` The start time for output of the waveform data.

`eTime` The last time for waveform output will be less than or equal to this time. The command processes events from `sTime-smooth` to `eTime+smooth`.

`maxF` The output is stored as if it were a sampled signal. That is as a 16-bit integer in the range -32768 to 32767. `maxF` sets the frequency that corresponds to the 32767 value. It is the maximum frequency that can be represented in the result. Frequencies higher than this are limited to `maxF`.

`meanF` This sets the frequency that corresponds to 0 in the 16-bit waveform data. Most users set this to 0. The lowest frequency that can appear in the result is `maxF-2*(maxF-meanF)`. Frequencies lower than this are limited to this value.

If the result lies in the range `freq±delta`, you get the best resolution by setting `maxF` to `freq+delta` and `meanF` to `freq`.

`query%` Set this to 0 or omit it to query overwriting an existing channel. If this is present and non-zero, an existing channel is overwritten with no comment.

`wave%` This optional argument is used when `smooth` is greater than 0 and sets the smoothing function: 0=rectangular, 1=triangular, 2=raised cosine, 3=Gaussian (to 4 sigmas). If omitted, raised cosine smoothing is used. These functions are the same as are used for virtual channels.

`wave[]` If you provide an array, it is scaled to span `smooth` seconds. The sum of the waveform values must be in the range 1.2e-38 to 3.4e+38. If `nPre` is omitted or negative, the smoothing function is symmetrical and you supply the right-hand side only. You can set any array size; the built-in functions use 1000 points. This example matches the raised cosine generated with `wave%` set to 2.

```
var wave[1000], i%, nPre := -1.0;
const pi := 3.14159;
for i% := 0 to 999 do wave[i%]:=1 + Cos(i%*pi/1000) next;
```

`nPre` This sets the index in `wave[]` that matches the event time for asymmetric smoothing functions. The entire `wave[]` array is `smooth` seconds wide. If `nPre` is omitted or negative, a symmetric function is assumed. This example generates a typical asymmetric smoothing function in the `wv[]` array:

```
var wv[400],i%, nPre;
const a:=40, b:=80;  'The time constants in points
for i%:=0 to 399 do wv[i%]:=exp(-i%/a)*(1-exp(-i%/b)) next;
nPre := b*Ln(1+a/b); 'Calculate position of maximum
```

Returns The function returns 0 unless the user decided not to overwrite a channel, in which case the result is a negative error code. You can also get error -5799 if you run out of memory (unlikely) or disk space.

Usage

This command can be used to investigate frequency changes in an event train. If you have an event channel of heart beats at 70 beats per minute (1.17 Hz) and you are seeking a respiration-induced modulation at about 0.25 Hz, you can convert the heartbeats into a waveform with this command. The result would be a constant level of 1.17 Hz, plus a small oscillation around 0.25 Hz. The oscillation can be detected with `SetPower()`.

For 0.01 Hz resolution with `SetPower()` you need 100 seconds of data per power spectrum block. With a 1024 point transform size, a `wInt` of 0.1 Hz gives a block size of 102.4 seconds. The `SetPower()` result will have 512 bins, spanning 0 to 5.00 Hz in steps of 0.01 Hz. A `smooth` period of 5-10 seconds would be appropriate.

See also:

Virtual channel smoothing, `BinSize(), SetPower(), VirtualChan()`

Exp()

This function calculates the exponential function for a single value, or replaces a real array by its exponential. If a value is too large, overflow will occur, causing the script to stop for single values, and a negative error code for arrays.

```
Func Exp(x|x[]{[]...});
```

`x` The argument for the exponential function or an array of real values.

Returns With an array, the function returns 0 if all was well, or a negative error code if a problem was found (such as overflow). With an expression, it returns the exponential of the number.

See also:

`Abs(), ATan(), Cos(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan(), Trunc()`

ExportChanFormat()

This sets the channel text format used by `ExportTextFormat()`, `FileSaveAs()` and `EditCopy()`. It is equivalent to the Channels section of the Time view Copy As Text dialog. See `ExportTextFormat()` for the command to reset the format.

```
Func ExportChanFormat(type%, synop%, data%{, as%});
```

`type%` The channel type to set the format for. Types 2 and 3 share the same settings. Types 1 and 9 also share the same settings.

1 Waveform	4 Level (Evt+-)	7 RealMark
2 Event (Evt-)	5 Marker	8 TextMark
3 Event (Evt+)	6 WaveMark	9 RealWave

`synop%` Set this non-zero to enable synopsis output for this channel type.

`data%` Set this non-zero to enable data output for this channel type.

`as%` This sets the format for Markers and extended marker types (RealMark, TextMark and WaveMark). It is ignored and can be omitted for waveform and event channels. If omitted or set to 0, the channels are

treated as their own type. The codes are the same as for the `type%` field; the useful values are 2 and 3 to output as if the channel were an event, 4 to output as a Level, and 5 to dump the channel as if it were a marker. All other values are ignored.

Returns 0 if all was OK or a negative error code.

See also:
`FileSaveAs(), ExportChanList(), ExportTextFormat()`

ExportChanList()

This command sets a time range and a channel list to export for use by `FileSaveAs()` and `EditCopy()`. The current view must be a time view. There are two command variants. The first with one or no arguments clears the channel list. The second, with a channel specification, adds a set of channels and a time range to a list., suitable for use with `ExportTextFormat()`. When used with `ExportRectFormat()` or with an external exporter, only the first set of channels and time range is used. To write different segments of a file to separate output files, clear the list with the first command variant before setting each segment with the seconds variant, then output the data with `FileSaveAs()` or `EditCopy()`. See `ExportTextFormat()` and `ExportRectFormat()` for examples.

```
Proc ExportChanList({flags%});
Proc ExportChanList(sTime, eTime, cSpc {,cSpc...});
```

`sTime` The start time of the range of data to save.

`eTime` The end time of the range of data to save.

`cSpc` A channel specifier for the channels to export.

`flags%` This affects data written by `FileSaveAs()` with `type%` of 0 and is the sum of:

1 Time shift data so that the earliest `sTime` appears at time 0 in the output file.

2 Write RealWave channels as waveform data for backwards compatibility.

4 Display a progress bar with option of cancel (from [10.02]).

WARNING: Remember to clear the list before using the second command variant unless you really want to add a new time range and channel specification to the end of the existing list.

Example of exporting as a Spike2 data file

The following example saves channels 1, 2, 3 and 31 of the current file for the time ranges 1-2 seconds and 10-20 seconds to a new file with the type (smr or smrx) determined by the file extension set by the user:

```
'$Export|Example code to export sections of a file as a new Spike2 data file
if ViewKind() <> 0 then Message("Current view is not a time view"); halt endif

ExportChanList();       ' Clear the list, copy times as specified
ExportChanList(1.0, 4.0, "1,2,3,31");   'Set the first time range and selected channels
ExportChanList(10, 20.0, "1,2,3,31");   'Set the second time range and selected channels
var ok% := FileSaveAs("",0,8,"Set the output file name", 100); 'Set 100 channels in the file
if ok% then Message("Error %d when saving", ok%) endif;
```

See also:
Channel specifiers, `FileSaveAs(), ExportChanFormat(), ExportTextFormat(), ExportRectFormat(), EditCopy()`

ExportRectFormat()

This command sets the text export format for use by `FileSaveAs()` and `EditCopy()` in a time view. This is an alternative to using `ExportTextFormat()` and generates equally spaced output in time for all channels. If used with no parameters, this will reset the rectangular text output configuration to the default of 100 Hz output, linear interpolation, double-quotes delimiter and tab for the separator. Otherwise at least 2 arguments must be supplied.

```
Proc ExportRectFormat({freq, inter%{, delim${, sep${, flag%}}}})
```

`freq` This sets the number of output points per channel generated per second.

`inter%` This sets the interpolation method:

- 0 Use the data point nearest to the time.
- 1 Linear interpolation between the points either side.
- 2 Cubic spline interpolation (waveform data only).
- 3 Cubic spline interpolation if drawn splined, else linear (waveforms only).

`delim$` This sets the character used to delimit the start and end of text strings. If omitted, text strings are surrounded by quote marks, for example, `"Volts"`.

`sep$` The character to separate multiple data items on a line. If omitted, a Tab is used.

`flag%` Optional, taken as 1 if omitted. This is the sum of:

- 1 Export channel titles.
- 2 Export channel units.

The following example exports channels 1, 2 and 3 at 200 Hz from times 10 seconds to 30 seconds as a text file. The times are shifted so that the first data item is at time 0. Waveform channels are mapped to 200 Hz using cubic spline interpolation. The output has titles but no units.

```
ExportChanList(1);            'Clear export list, sets zero shift
ExportChanList(10,30,1,2,3);  'Choose channels 1, 2 and 3
ExportRectFormat(200, 2);     '200 Hz, cubic spline format
FileSaveAs("", 1, 0, "Set the output text file"); 'Export as text file, and...
EditCopy();                   '...also copy to the clipboard
```

See also:
`FileSaveAs(), ExportChanList(), ExportTextFormat(), EditCopy(),`

ExportTextFormat()

This command sets the text export format for use by `FileSaveAs()` and `EditCopy()` in a time view. It is equivalent to the top section of the File Dump Configuration dialog. The form with no arguments resets the dialog to enable all check boxes, sets the columns to 1, sets the string delimiter to a double quote mark, sets the separator to a tab character, and sets all channel types to be treated as themselves.

```
Proc ExportTextFormat(head%, sum%, cols%{, delim${, sep$}});
Proc ExportTextFormat();
```

`head%` If this is non-zero, Spike2 will output the header.

`sum%` If this is non-zero, Spike2 outputs the channel summary information.

`cols%` The number of columns to use when writing waveforms and event times.

`delim$` This sets the character used to delimit the start and end of text strings. If omitted, text strings are surrounded by quote marks, for example, `"Volts"`.

`sep$` The character to separate multiple data items on a line. If omitted, a `Tab` is used.

The following example exports channels 1 and 2 from 90.0 to 100.0 seconds and from 110 to 120 seconds as a text file.

```
ExportChanList(1);            'Clear export list, sets zero shift
ExportChanList(90, 100, 1, 2);'set channels 1 and 2
ExportChanList(110,120, 1, 2);'add 10 more seconds after 10s gap
ExportTextFormat();           'reset the file dump dialog
ExportChanFormat(1,0,1);      'turn off synopsis for waveform data
ExportTextFormat(0, 0, 1);    'No header or summary
FileSaveAs("", 1, 0, "Set the name of the text file");    'export to a new text file
EditCopy();                   '...also copy to the clipboard
```

See also:
`EditCopy(), ExportChanList(), ExportChanFormat(), ExportRectFormat(), FileSaveAs()`

F

File...()

FileApplyResource()

This applies a resource file to the current time, result or XY view. If a time view is duplicated, all other duplicates are deleted, then the resource file is applied, which may create duplicates. The current view handle is not changed. Handles of duplicate views are not preserved, even if the resource file creates the same number of duplicates. We expect you to apply the resources from a view of the same type, but we do not check for this. In most cases, applying the wrong view type will cause no change, The `view%` and `omit%` arguments were added at version 9.01; you cannot use them in previous versions of Spike2.

```
Func FileApplyResource(name${, omit%});
Func FileApplyResource(view%{, omit%});
```

`name$` The resource file to apply. You must include the file extension and required path. Remember that `\` in a string must be entered as `\\` or use `/`. If the name is `""` or contains a `"*"` or a `"?"`, the user is prompted for a file.

`view%` The view handle of a view to copy information from. This creates a temporary resource file, equivalent to using `FileSaveResource()` applying it, then deleting it. It saves time by never writing the data to disk.

`omit%` This is the sum of a set of flag values that are used to stop some attributes of the view being applied. It is very likely that additional flags will be added in the future.

Value	Meaning
1	Do not move the target.
2	Do not change the visibility of the target.
4	Do not change the x axis range of the target.
8	Do not change the y axis ranges of the target.

Returns The number of modified views (usually 1 unless the resource file generates duplicates), 0 if the resource file did not contain suitable information, -1 if the user cancelled the file dialog, -2 if the file could not be found or -3 if there is any problem reading or applying the resource, -4 if the `view%` handle does not refer to a Time, Result or XY view.

Applications

This is normally used where you have a set of very similar files to analyse and you want a quick way to get them all into the same state. Each time you load one you apply a specific resource file that you have saved for this purpose. This would use the first variant of the function using the `name$` argument.

Another use is where you have two or more similar files loaded into Spike2 that you want to keep in step so changes to the display mode of one are applied to the other. This would use the second variant of the function, using the `view%` argument.

See also:

`FileGlobalResource(), FileOpen(), FileSave(), FileSaveResource()`

FileClose()

This is used to close the current window or external file and release the system resources used by it. You can supply an argument to close all windows associated with the current time or result view or to close all the windows belonging to the application. You cannot close the application window with this, use `FileQuit()` instead. The `flags%` argument was added at version [10.20].

```
Func FileClose({all% {,query%{, flags%}});
```

`all%` This argument determines the scope of the file closing. Possible values are:

-1 Close all windows except loaded scripts and debug windows.

0 Close the current view. This is the same as omitting `all%`.

1 Close all windows associated with the current view.

2 Close all windows associated with the current view and save result views (new at [10.15]). This is equivalent to the File menu Close and Link command.

`query%` This determines what happens if a window holds unsaved data:

-1 Don't save the data or query the user.

0 Query the user about each window that needs saving. If the user chooses Cancel, the operation stops, leaving all unclosed windows behind. This is the same as omitting `query%`.

If the view is a Time View with a memory channel, this counts as unsaved data in the sense that the channel will not be saved and restored. What happens in this case depends on the Edit menu Preferences command, *Warn if file close would lose memory channel data* in combination with `query%`.

`flags%` Sum of flag values that control the file close process. Currently only 1 option is defined:

1 This prevents an unsaved Time view from being moved to the recycle bin (the default behaviour if it is longer than the Minimum Recycle time set in the Preferences).

Returns The number of views that have not been closed. This can occur if a view needs saving and the user requests Cancel, or if the view was handling a user-defined mouse function.

If the current view is a time view and you have been sampling data, you must call `SampleStop()` before using this command to set a file name. If you do not, this is equivalent to aborting sampling and your data file will not be saved. Use `FileSaveAs()` just before calling **`FileClose()`** to set a file name.

Do not use **`View(fh%).FileClose()`** if **`fh%`** is the current view; you will get an error when Spike2 tries to restore the current view. Use **`View(fh%);FileClose()`** instead.

See also:
`FileOpen(), FileSave(), FileSaveAs(), FileNew(), SampleStop()`

FileComment$()

This function accesses the file comments in the file associated with the current time view. Files have five comment strings. File comments are limited in length when they are written to data files (to 79 8-bit characters for a 32-bit smr file and to 2000 Unicode characters (was 100 before Spike2 8.09) for a 64-bit smrx file). If you set a comment of more than 100 characters and a user edits it with a version of Spike2 before 8.09, the comment will be truncated to 100 characters.

64-bit smrx files actually allow 8 comments, but we are currently reserving items 6-8.

```
Func FileComment$(n% {,new$));
```

`n%` The number of the file comment line in the range 1 to 5

`new$` If present, the command replaces the existing comment with `new$`.

Returns The comment at the time of the call or a blank string if `n%` is out of range.

See also:
`ChanComment$()`

FileConvert$()

This function converts a data file from a "foreign" format into a 64-bit `.smrx` Spike2 data file. If you want to create a 32-bit `.smr` file you must Export the result or use `FileSaveAs()`. The range of foreign formats supported depends on the number of import filters in the `import` folder (inside the folder where Spike2 was installed). The convert process follows the description for the interactive File menu Import command. However, there is no progress dialog, so you will have to wait patiently for the result.

```
Func FileConvert$(src${, dest${, flag%{, &err%{, cmd${, minCh%}}}}});
```

`src$` This is the name of the file to convert. The file extension (if present) is used to determine the file type, which will determine the importer to use (unless `flag%` bit 0 is set or the importer DLL is set using the

cmd$ argument). Known file extensions and configuration information can be found here. If you do not provide a full path, most importers will search for the file in the current folder.

Prior to version 10, if src$ contained * or ? characters or was empty, the user was given the opportunity to select a file; the initial name was blank. From version [10.00], if src$ contains a vertical bar character '|', the src$ string is taken as a file filter that will override any file filters set by the selected importer. This follows the pattern:

```
"Type 1 (*.fl1)|*.fl1||Type 2 (*.fl2;*.fl3)|*.fl2;*.fl3|"
```

This example produces two file types, one with two extensions. There is one vertical bar between the description and the template and after every template. For reasons of backwards compatibility double vertical bars are also accepted as separators.

If src$ does not contain a vertical bar, but does contain * or ? characters, it is used to set the initial file name and any extension sets the default extension, used when no extension is provided by the user. Setting a file name that ends with a dot character will stop the automatic addition of an extension.

dest$ If present, it sets the destination file. If this is not a full path name, the name is relative to the current directory. If you do not supply a file extension, Spike2 appends ".smrx". If you set any other file extension, Spike2 cannot open the file as a data file. If you omit this argument, the converted file is written to the same name and path as the source file with the file extension changed to .smrx.

flag% This argument is the sum of the flag values:

1	Ignore the file extension of the source file and try all possible file converters (could be slow).
2	Allow user interaction if required (otherwise sensible, non-destructive defaults are used for all decisions).
4	Display a progress dialog during the file import which also allows the user to cancel the operation (see err%). This was added in version [8.04].

err% Optional integer variable that is set to 0 if the file was converted, otherwise it is returned holding a negative error code or 1 if the user cancelled the operation.

cmd$ Optional string holding parameters that control the importer. The string is of the form: name1=value1;name2=value2;name3=value3 where the names are case insensitive. We append ;AppVer=Spike2,*N* to the end of the string; the *N* stands for the version of Spike2 multiplied by 100, so for version 10.13 *N* is 1013. It is entirely up to the importers what names they recognise. For more information about specific importers click here and then find the importer in the table. If the importer supports a command line, the **cmd$** column contains Yes. Click the Yes for details of supported keywords.

Before the string is passed to an importer it is scanned for parameters that apply generally. At the moment, only one such option exists:

Name	Description
dll	You can use this option to select a specific import DLL. If you do not supply this, Spike2 will use the first importer it finds that supports a file extension that matches the file set by src$. Most data formats have a unique extension, but some have a wide range of extensions. For example, to force the use of the CED binary importer you would set cmd$:= "dll=binary" as the argument. You can find the DLL name in the table of importers for the File menu Import command, or record your actions. If the nominated DLL cannot be found, the command will fail.

minCh% Optional (taken as 32 if omitted or less than 32). The minimum number of channels in the output data file. This allows you to specify extra channels beyond those needed to import the data for use as work space.

Returns The full path name of the new file or an empty string if there was no output.

See also:

FileOpen(), FilePath$(), FilePathSet(), FileList(), FileSaveAs(), Import command

FileCopy()

This function copies a source file to a destination file. File names can be specified in full or relative to the current directory. Unless you are certain of the current directory it is safer to use full paths. Wild card characters cannot be used.

```
Func FileCopy(src$, dest${, over%});
```

`src$` The source file to copy to the destination. This file is not changed.

`dest$` The destination file. If this file exists you must set `over%` to overwrite it.

`over%` If this optional argument is 0 or omitted, the copy will not overwrite an existing destination file. Set to 1 to overwrite.

Returns The routine returns 1 if the file was copied, 0 if it was not. Reasons for failure include: no source file, no destination path, insufficient disk space, destination exists and insufficient rights.

See also:
`BRead(), BWrite(), FileDelete(), FileOpen(), ProgRun()`

FileDate$()

In a time view this function returns a string holding the date when the data file was sampled with an optional time offset. The date was not stored in the data file before Spike2 version [4.03]. In other views it gets a date the operating system associates with the disk file attached to the view, if it exists. Use `FileTimeDate()` to get the date as numbers or both file and date as a string. If there is no date, the result is an empty string. Default argument values are shown **bold**.

```
Func FileDate$({dayF%{, monF%{, yearF%{, order%{, sep$${, offs%{, what
%}}}}}}});
```

`dayF%` This sets the format of the day field in the date. This can be written as a day of the week or the day number in the month, or both. The options are:

- 1 Show day of week: "`Wednesday`".
- **2** Show the number of the day in the month with leading zeros: "`01`".
- 4 Show the day without leading zeros: "`1`". This overrides option 2.
- 8 Show abbreviated day of week: "`Wed`".
- 16 Show weekday name first, regardless of the `order%` field.

Use 0 for no day field. Add the numbers for multiple options. For example, to return "`Wed 01`", use `11` (1+2+8) as the `dayF%` argument.

If you add 8 or 16, 1 is added automatically. If you request both the weekday name and the number of the day, the name appears before the number.

`monF%` The format of the month field. This can be returned as either a name or a number. If this argument is omitted, the value 3 is used. The options are:

- 0 No month field.
- 1 Show name of the month: "`April`".
- 2 Show number of month: "`04`"
- **3** Show an abbreviated name of month: "`Apr`"
- 4 Show number of month with no leading zeros: "`4`"

`yearF%` The format of the year field. This can be returned as a two or four digit year.

- 0 No year is shown
- **1** Year is shown in two digits: "`98`".
- 2 Year is shown in two digits with an apostrophe before it: "`'98`".
- 3 Year is shown in four digits: "`1998`".

`order%` The order that the day, month and year appear in the string.

0 Operating system settings
1 month/day/year
2 day/month/year
3 year/month/day

`sep$` This string appears between the day, month and year fields as a separator. If this string is empty or omitted, Spike2 supplies a separator based on system settings.

`offs%` This is a time offset from the start of the file, in seconds. It is treated as zero if omitted or if it is negative. You might use the offset if you wanted to write out the date that something happened in a long term recording.

`what%` Optional, with default value 0 and ignored for a time view. This chooses which of the times associated with the file to get:

0 The last modified time.
1 The file created time.
2 The last file access time.

Returns The date as a string.

Changes at Spike2 9.04

Prior to version 9.04, the `what%` argument did not exist and the command could only be used with a time view.

See also:
`Date$(), FileTimeDate(), FileTime$()`

FileDelete()

This deletes one or more files. File names can be specified in full or relative to the current directory. Windows file names look like `x:\folder1\folder2\foldern\file.ext` or `\\machine\folder1\folder2\foldern\file.ext` across a network. If a name does not start with a `\` or with `x:\` (where `x` is a drive letter), the path is relative to the current directory. Beware that `\` must be written `\\` in a literal string passed to the script compiler.

```
Func FileDelete(const name$[]|name${, opt%{, log%}});
```

`name$` This is either a string variable or an array of strings that holds the names of the files to delete. Only one name per string and no wild card characters are allowed. If the names do not include a path they refer to files in the current directory.

`opt%` If this is present and non-zero, the user is asked before each file in the list is deleted. You cannot delete protected or hidden or system files.

`log%` If present and non-zero, write the name of any file that fails to delete to the Log view, together with the reason the delete failed.

Returns The number of files deleted or a negative error code.

See also:
`FilePath$(), FilePathSet(), FileList(), FileStatus()`

FileGlobalResource()

This function is equivalent to the Global Resources dialog. The function has two forms: the first is for setting values, the second is to read them back:

```
Func FileGlobalResource(flags%, loc%, name${, path$});
Func FileGlobalResource(&loc%, &name$, &path$);
```

`flags%` This is the sum of the following values:

1 Enable the use of global resources. If unset, no global resources are used.
2 Only use global resources if a data file has no associated resource file.

4 Only use if the data file is within the path set by `path$`.

`loc%` The location of the global resource file. 0=the folder that Spike2 was run from, 1=search the data file folder, then the Spike2 folder, 2=the data file folder only.

`name$` The name of the global resource file excluding the path and file extension.

`path$` The file path within which to use the global resources. If you omit this argument when setting values, the current path does not change.

Returns When setting values, the return value is 0 or a negative error code. When reading back values, the return value is the `flags%` argument.

See also:
Global Resources dialog, `FileApplyResource(), FileOpen(), FileSave(), FileSaveResource()`

FileInfo()

This returns the file format version number of the Spike2 data file in the current time view. When used with any other view type it returns 0 (but may be extended in the future).

```
Func FileInfo()
```

Returns For a time view it returns the version number of the Spike2 data file format. 32-bit files have a format version number less than 256. 64-bit files have a version number starting at 256 and currently (Apr/2020) 257. Format number changes indicate new filing system features that are incompatible with software that was written before the version number changed. The version of Spike2 you are using can read files of all versions that are 256 or less.

FileList()

This command gets a list of files from one directory that are of a certain type or that match a file name mask. The search starts in the current directory as set by the `FilePathSet()` script command.

Get files relative to current directory

This function gets lists of files and sub-directories (folders) in the current directory and can also return the path to the parent directory of the current directory. This function can be used to process all files of a particular type in a particular directory.

```
Func FileList(name$[]|&name$, type%{, mask${, flags%}});
```

`name$` This is either a string variable or an array of strings that is returned holding the name(s) of files or directories. Only one name is returned per string.

`type%` The objects in the current directory to list. The Parent directory is returned with the full path, the others return the name in the current directory. Values are:

-3 Parent directory	2 Output sequence files (`*.pls`)	17 Grid view file (`*.s2gx`) - from [10.02, 9.10]
-2 Sub-directories	3 Spike2 script files (`*.s2s`)	
-1 All files	4 Spike2 result view files (`*.srf`)	
0 Spike2 data files (`*.smr`, `*.smrx`)	6 Spike2 configuration files (`*.s2cx`)	
1 Text files (`*.txt`)	1 XY view data file (`*.sxy`)	
	2	

`mask$` This optional string limits the names returned to those that match it; * and ? in the mask are wild card characters. ? matches any character and * matches any 0 or more characters. Matching is case insensitive and from left to right.

`flags%` This argument did not exist before version **[10.00]**. Optional, with value 0 if omitted. The sum of a set of flags to qualify the returned files:

1 The returned file(s) must exist. This is always the case with this command variant.

2 The returned file(s) cannot be Read Only.

Returns The number of names that met the specification or a negative error code. This can be used to get the size of the string array required to hold all the results.

Version 8 changes

If you select data or configuration files using the `type%` argument, you will get both old and new format files. This is done so that old scripts written for earlier versions of Spike2 and that use `type%` to specify data or configuration files will still work. You can use the `mask$` argument to be more specific. For example, if you only wanted old-format data files you would set `mask$` to `"*.smr"`.

If your script needs to specify part of the file name, but needs to list both old and new files you could specify the mask as: `"may*.smr?"` and this would list both `may123.smr` and `may124.smrx`, however you would also list `may999.smrz`.

Version 9 changes

Version 9 no longer supports the old-style `*.s2c` configuration files. You can convert these files by loading them using Spike2 version 7 or 8, open a sampling file ready to sample, then save the configuration (which writes the new format).

User selects files in a directory [10.00]

This variant opens a standard File dialog in which the user may select one or more files or type in the names of one or more files.

```
Func FileList(names$[]|&name$, title${, mask${, flags%}});
```

`names$` An array of strings to hold the path to the folder and the selected files within it. If the return value is 1, `names$[0]` holds the full path to a single file. If the return value is > 1, `names$[0]` holds the path to the folder holding the selected files (always terminated by a `\` character). The remaining returned elements hold the selected file names within the folder. The size of the array limits the number of files that can be returned; if `names$[]` is 100 elements longs, the maximum number of returned files is 99 as the first element holds the path to the folder. If `names$` is 1 element long, the user may select only a single file.

`name$` A string variable to be returned with the full path to a single file.

`title$` The title of the File dialog; this can be used to prompt the user to select a file for a particular purpose.

`mask$` This sets the types of file, selected by file extension, to display in the dialog box. This is a convenience and does not limit the user to selecting only files with these extensions as they can type in any file name, or type their own file mask and hit the `Enter` key to display and select any files that they please.

The file types to display are described by pairs of strings as : `Description text|filter`. The description text is displayed to the user and the filter is the file name pattern (or patterns) to match. You can specify multiple, selectable filters by separating each filter pair by `||`. The list is terminated by two vertical bars as `||`. The standard Windows format for masks can be achieved with:

```
"Type 1 (*.fl1)|*.fl1||Type 2 (*.fl2;*.fl3)|*.fl2;*.fl3||All files (*.*)|*.*||"
```

This example produces three file types, the first with one extension, the second with two extensions and the third that accepts any file type. There is one vertical bar between the description and the template and two after every template to match the format used by `FileOpen()`, which in turn, matches a format used in the original version of Spike2. Spike2 is quite forgiving about the format, and will accept single vertical bars for all separators and will cope if you omit the final `||`.

If you do not want to specify file extensions, set an empty string, and all files are displayed.

`flags%` Optional, with value 0 if omitted. The sum of a set of flags to qualify the returned files:

1 The returned file(s) must exist.

2 The returned file(s) cannot be Read Only.

Returns The number of strings set in the array or -1 if the user hit the File dialog Cancel button. If the user did not select a file, the result is 0. Note that a returned value of 2 should not occur.

See also:
FilePath$(), FilePathSet(), FileDelete(), FileName$()

FileName$()

This returns the name of the file associated with the current view (if any) and from [10.09a] can extract parts of a path from a string. You can recall the entire file name, or any part of it. If there is no file or the file has not been saved or the current view is a sampling file with no automatic name set, the result is an empty string. The name is considered to be made up of: Drive + Path + Name + Number + Extension. So if we had C:\Andre\Data\Fred0123.smrx

Drive The part of the file name that identifies the drive or volume, for example "C:".

Path The part of the file name, not including the drive and not including the file name, for example "\Andre\Data\".

Name The name of the file, not including any trailing digits, for example "Fred". If the file name is only composed of digits (0-9), then this is empty.

Number The trailing digits of the name, for example "0123". This is often empty.

Extension The end of the file name from the last period onwards, for example ".smrx".

```
Func FileName$({mode%{, fName$});
```

mode% If present, determines what to return, if omitted taken as 0. Options greater than 0 return individual file parts. The negative options were added at version 8.16 and 9.04. Before these versions, using a negative value returns the full file path.

- -5 Name + Number + Extension, for example: "Fred0123.smrx".
- -4 Name + Number, for example: "Fred0123".
- -3 Drive + Path + Name + Number, for example: "C:\Andre\Data\Fred0123".
- -2 Drive + Path + Name, for example: "C:\Andre\Data\Fred".
- -1 Drive + Path, for example: "C:\Andre\Data\".
- 0 Or omitted, returns the full file name including the path.
- 1 Drive. The disk drive/volume name.
- 2 Path. The path section, excluding the volume/drive and the name of the file.
- 3 Name. The file name up to but not including the last . excluding any trailing number.
- 4 Number. The trailing digits up to but not including the last .
- 5 Extension. The file name extension from the last dot.

fName$ If present, this holds the file path to be used as a source of the returned data. This is useful in cases where you need to extract parts of an arbitrary file path. New at [10.09a].

Returns A string holding the requested name, or a blank string if there is no file or name part.

Notes
Multimedia windows did not report the associated file name before versions [10.02, 9.10]. To find the name displayed at the top of a sampling data file, use the WindowTitle$() command.

See also:
FileList(), FilePath$(), FilePathSet(), SampleConfig$(), SampleSequencer$()

FileNew()

This is equivalent to the File menu New command. It creates a new window and returns the handle. You can create visible or invisible windows. Creating an invisible window lets you set the window position and properties before you draw it. The new window is the current view and if visible, the front view. Please remember that there are limits on the number of open windows imposed by the operating system.

Use `FileSaveAs()` to name created files. Use `FileClose()` to close them. There are three command variants:

```
Func FileNew(type%{, mode%});
Func FileNew(7{, mode%{, upt, tpa%, maxT{, nChan%{, big%}}}});
Func FileNew(17, mode%{, cols%{, rows%}});
```

`type%` The type of file to create:

- 0 A Spike2 data file based on the sampling configuration, ready for sampling. This may open several windows, including the floating command window and the sequencer control panel. Use `SampleStart()` to begin sampling and `SampleStop()` to stop sampling before calling `FileSaveAs()` to give the new file a name. During sampling, type 0 data files are saved in the folder set by the Edit menu preferences or the `FilePathSet()` command. Use `FileSaveAs()` command after `SampleClose()` to move the data file to its final position on disk.
- 1 A text file in a window
- 2 An output sequence file in a window
- 3 A Spike2 script file
- 7 An empty Spike2 data file (not for sampling). `upt%`, `tpa%` and `maxT` must be supplied. You can use `ChanNew()`, `ChanWriteWave()`, `ChanSave()`, `MemSave()` or to add data. Use `FileSaveAs(name$,-1)` to name the new file.
- 12 An XY view with one (empty) data channel. Use `XYAddData()` to add more data and `XYSetChan()` to create new channels.
- 17 An empty Grid view with `rows%` rows and `cols%` columns.

`mode%` This optional argument determines how the new window is opened. The value is the sum of these flags. If the argument is omitted, its value is 0. The flags are:

- 1 Make the new window(s) visible immediately. If this flag is not set the window is created, but is invisible.
- 2 For data files, if the sampling configuration holds information for creating additional windows or channels, use it. If this flag is not set, data files extract enough information from the sampling configuration to set the sampling parameters for the data channels. You can use the `ViewLink()` command to identify additional views or channels.
- 4 Show the spike shape set up dialog if there are WaveMark channels in the sampling configuration for a data file. If the spike shape dialog appears, this function does not return until the user closes the dialog.

`upt` Only for `type%` = 7. The microseconds per unit time for the new file (the clock tick period) in the range 0.001 to 32767. If `upt` is a non-integral value, Spike2 versions prior to [4.02] will not open the file. This sets the time resolution of the new file and the maximum duration. A 32-bit .smr file has a maximum of 2^31-1 time units, so a 1 µs resolution limits a file to around 30 minutes. A 64-bit .smrx file allows up to 2^63-1 ticks; at 1 µs resolution the file can last thousands of years.

`tpa%` Only for `type%` = 7. This was used in the filing system versions 1 and 2 to set the time units per ADC conversion for the new file. You could set values in the range 1 to 32767. The available sampling intervals for waveform data in the file were `n * upt% * tpa%` microseconds where `n` is an integer. This value is ignored in Spike2 version 8 onwards and the effect is as if you set `tpa%` to 1. This value related to how data sampling was set up for a particular 1401; see `SampleTimePerAdc()`; it is no longer relevant to data files (which are independent of how the data was acquired).

`maxT` Only for `type%` = 7. This sets the initial file size in time, which sets the value returned by `MaxTime()` until data written to the file exceeds this time.

`nChan%` Only for `type%` = 7. This sets the number of channels in the file in the range 32 to 2000. If you omit `nChan%`, 32 channels are used. Spike2 versions 5-8 can read up to 400 channels, 4 can read up to 100.

Version 3 can only read files with 32. If the sampling configuration is set for a 32-bit .smr file, only up to 400 channels can be set.

big% Only for type% = 7. If present this sets the type of the new file. If omitted, a 64-bit .smrx file is generated.

- 0 Very old 32-bit .smr format with a size limit of 2 GB.
- 1 Write as a 32-bit .smr file with a size limit of 1 TB.
- 2 Write as a 64-bit .smrx file.

cols% Only for type% = 17. It sets the number of columns in the Grid. There is currently an arbitrary limit of 1000 columns. If omitted, a default number is set (currently 26). You can resize the grid with GrdSize().

rows% Only for type% = 17. It sets the number of rows in the Grid. There is currently an arbitrary limit of 10000 rows. If omitted, a default number is set (currently 100).

Returns It returns the view handle (or the handle of the lowest numbered duplicate for a data file with duplicate windows) or a negative error.

See also:
ChanNew(), ChanSave(), ChanWriteWave(), FileOpen(), FileSave(), FileSaveAs(), FileClose(), FilePathSet(), GrdSize(), MaxTime(), MemSave(), SampleStart(), SampleStop(), ViewLink(), XYAddData(), XYSetChan(), XY example

FileOpen()

This is the equivalent of the File Open menu command. It opens an existing Spike2 data, text or Grid file in a window, or an external text or binary file. If the file is already opened, a handle for the existing view is returned. The window becomes the new current view. You can create windows as visible or invisible. It is often more convenient to create an invisible window so you can position it before making it visible. You close files with the FileClose() command. There are limits on the number of open windows imposed by the operating system. If you want to get a list of files for a particular operation you could consider using the FileList() command.

```
Func FileOpen(name$, type% {,mode% {,text$}});
```

name$ The name of the file to open. This can include a path (but be careful using the backslash character). The file name is operating system dependent, see FileDelete(). If the name is blank or contains * or ? (Windows only), the file dialog opens for the user to select a file. For file types 8 and 9 only, name$ can be of the form:

```
"Type 1 (*.fl1)|*.fl1||Type 2 (*.fl2;*.fl3)|*.fl2;*.fl3|"
```

This example produces two file types, one with two extensions. There is one vertical bar between the description and the template and after every template. For reasons of backwards compatibility double vertical bars are also accepted as separators.

type% The type of the file to open. The types currently defined are:

- 0 Open a Spike2 data file in a window
- 1 Open a text file in a window
- 2 Open an output sequence file in a window
- 3 Open a Spike2 script file in a window
- 4 Open a result view file in a window
- 6 Load configuration file or read configuration from data file (see here for information on file extensions)
- 8 An external text file without a window for use by Read() or Print()
- 9 An external binary file without a window for use by BRead(), BWrite(), BSeek() and other binary routines.
- 12 Open an XY view file in a window.
- 17 Open a Grid view file in a window.

mode% This optional argument determines how the window or file opens. If the argument is omitted, its value is 0. For file types 0 to 4, 12 and 17 the value is the sum of:

1 Make the new window(s) visible immediately. If this flag is not set the window is created, but is invisible.

2 Read resource information associated with the file. This may create more than one window, depending on the file type. For data files, it restores the file to the state as it was closed. If the flag is unset, resources are ignored.

4 Return an error if the file is already open in Spike2. If this flag is not set and the file is already in use, it is brought to the front and its handle is returned.

8 Determines the start folder if the system File Open dialog is used and `name$` does not specify a path. Add 8 to follow the operating system rules, usually opening the last File Open dialog folder. Otherwise open the current folder as set by `FilePathSet()`.

16 Used when opening an external text file (`type%` 8) to read, pay attention to any BOM or failing that, read as Unicode (wide UTF-16LE) characters. For writing external text files, use wide (UTF-16LE characters.

When used with file types 8 and 9 the following values of `mode%` are used. The file pointer (which sets the next output or input operation position) is set to the start of the file in modes 0 and 1 and to the end in modes 2 and 3. You can also add 8, as above, to set where any File Open dialog starts or 16 to use wide characters.

0 Open an existing file for reading only

1 Open a new file (or replace an existing file) for writing (and reading)

2 Open an existing file for writing (and reading)

3 Open a file for writing (and reading). If the file doesn't exist, create it.

When used with file type 6 (configuration files), the `mode%` argument is ignored and should be zero.

`text$` An optional prompt displayed as part of the file dialog.

Returns If a file opens without any problem, the return value is the view handle for the file (if multiple views open, it is the handle for the first time view created). If the file could not be opened, or the user pressed Cancel in the file open dialog, the returned value is a negative error code.

For configuration files (`type%` of 6), the return value is 0 if no error occurs or a negative error code.

If multiple windows are created for a data file, you can get a list of the associated view handles using `ViewList(list%[],64)`.

If a file fails to open because it is missing, there will be no message boxes warning of this (as there would be for interactive file open). For text-based files with a window, if the file fails to open due to being corrupt, or for some other reason, there may be a message box giving the reason in addition to the return value.

See also:
`FileDelete(), FileNew(), FileSave(), FileSaveAs(), FileClose(), FileList(), BRead(), BReadSize(), BSeek(), BWrite(), BWriteSize(), Read(), ViewList()`

FilePath$()

This function gets the current directory. This is the place on disk where file open and file save dialogs start. It can also get the path for created data files, the path to the Spike2 application and auto-saving. `<N>` stands for the current Spike2 version, so `Spike<N>` is currently `Spike11`.

```
Func FilePath$({opt%});
```

`opt%` If omitted this is taken as zero. This determines which directory/folder to get:

0 The current directory. This is the directory that incomplete path names (names that do not start with a drive specification or a backslash) are relative to. It is usually *not a good idea* to rely on the current folder pointing anywhere useful unless you have just set it (`FilePathSet()` with `opt%=0`)).

1 The path for Spike2 data files created by `FileNew()` as set by Edit menu Preferences in the Sampling tab or by `FilePathSet()` with `opt%=1`.

2 The path to the Spike2 application. You should treat this as a read-only location; Spike2 is normally installed within the protected `Program Files` directory tree where non-privileged

writes will fail. If you need a location to store data consider using one of the negative opt% values described below. The returned path ends with a "\" character.

3 The automatic file naming path as set by the Sampling Configuration Automation tab or by FilePathSet() with opt%=3.

We also support some negative values that return the path to special folders:

-1 The Desktop folder, for example: C:\Users\username\Desktop\

-2 The users documents folder (commonly called My Documents), for example: C:\Users\username\Documents\. This folder should always exist and be writeable.

-3 The system folder for Spike2 user and application data, for example: C:\Users\username\AppData\Local\CED\Spike<N>\ You can use this location to save files that are specific to the current logged on user and Spike2. This is where Spike2 saves the default filter bank files and the graphical sequencer .PLS file. You could also consider using the registry with the Profile() command to save small quantities of information.

-4 The Spike<N> folder inside the users documents folder (commonly called My Documents). It is recommended that you use this location for your data and files if possible. Available from [7.11].

-5 The Spike<N>Shared folder inside the documents folder for all users, for example C:\Users\Public\Documents\SpikeNShared\ Available from [7.11].

-6 The spike2 application program data folder, for example: C:\ProgramData\CED\Spike10\. This was added at [10.02, 9.10].

Calls with opt% in the range -3 to -6 will attempt to create the folder if it does not exist.

Returns A string holding the path or an empty string if an error is detected.

See also:
FileList(), FileName$(), FilePathSet(), Profile()

FilePathSet()

This function sets the current directory/folder, and where Spike2 data files created by FileNew() are stored until they are sent to their final resting place by FileSaveAs(). There are two version of the command. The first sets or optionally creates a directory based on a passed in path, the second prompts the user to choose and optionally create a directory.

```
Func FilePathSet(path${, opt%{, make%}});
Func FilePathSet(path$, opt%, prmpt${, make%});
```

path$ A string holding the new path to the directory. The path must conform to the rules for path names on the host system and be less than 255 characters long. If the path is empty or a prompt is set, a dialog opens for the user to select an existing directory/folder. If you supply the path as a literal string using backslash as a separator, for example "C:\\Folder\\SubFolder" you must remember to double the backslashes. Alternatively, use the slash character: "C:/Folder/SubFolder".

opt% If omitted this is taken as zero. This determines which directory/folder to set:

0 The current directory. This is the directory that incomplete path names (names that do not start with a drive specification or a backslash) are relative to.

1 The path for Spike2 data files created by FileNew() as set by Edit menu Preferences in the Sampling tab.

2 The path to the application. This is fixed, so an attempt to change it is an error.

3 The automatic file naming path as set by the Sampling Configuration Automation tab.

make% From [10.06] this is used only in the non-dialog version of the command. If this is zero or omitted, all elements of the path must already exist. If set to 1 and path$ is not empty, the command will create the directory/folder if all elements of the path exist except the last.

When the command displays a dialog this is ignored and the user can always create a new folder. Prior to [10.06], if this was zero the user can only select an existing path. If set to 1, the user is allowed to create a new directory/folder.

If the users sets a path you can find out what was set with `FilePath$()`.

`prmpt$` Optional prompt for use with the dialog. If you supply a prompt, a user dialog will appear using the current value of `path$` as the starting point. If `path$` is empty, the current path set by `opt%` is the starting point.

Returns Zero if the path was set, or a negative error code.

Do not set a path used for creating new files to be a folder that required administrative privilege to access unless you understand the implications of doing this, especially if you intend to allow others to access the file.

See also:
`FileList(), FileName$(), FilePath$()`

FilePrint()

This function is equivalent to the File menu Print command. It prints some or all of the current view to the printer that is currently set for Spike2. If no printer has been set, the current system printer is used. In a time or result view, it prints a range of data with the x axis scaling set by the display. In a text or log view, it prints a range of text lines. There is currently no script mechanism to choose a printer; you must do it interactively.

```
Func FilePrint({from{, to{, flags%}}});
```

`from` The start point of the print. This is in seconds in a time view, in bins in a result view and in lines in a text view. If omitted, this is taken as the start of the view.

`to` The end point in the same units as `from`. If omitted, the end of the view is used.

`flags%` 0=portrait, 1=landscape, 2=current setting. If omitted, the current value is used.

Returns The function returns 0 if all went well; otherwise it returns a negative error.

The format of the printed output is based on the screen format of the current view. Beware that for time and result views the output could be many (very many) pages long.

All the `FilePrintXXXX()` routines allow you to choose the orientation of the printed output. If you do not set the orientation, the last print orientation you used in the current Spike2 session is set. If you have not printed, the orientation depends on the current orientation set for the system printer.

See also:
`FilePrintScreen(), FilePrintVisible()`

FilePrintScreen()

This function is equivalent to the File menu Print Screen command. It prints all visible time, result, XY and text-based views to the current printer on one page. The page positions are proportional to the view positions in the Spike2 application window.

```
Func FilePrintScreen({head${, vTtl%{, box%{, scTxt%{, flags%{,
foot$}}}}}});
```

`head$` The page header. If omitted or an empty string, there is no page header.

`vTtl%` Set 1 or higher to print a title above each view, omitted or 0 for no title.

`box%` Set 1 or higher for a box around each view. If omitted, or 0, no box is drawn.

`scTxt%` Set 1 or higher to scale text differently in the x and y directions to match the original. If omitted or 0 scale both directions by the same scale factor.

`flags%` 0=portrait, 1=landscape, 2=current setting. If omitted, the current value is used.

`foot$` The page footer. If omitted or an empty string, there is no page footer.

Returns The function returns 0 if it all went well, or a negative error code.

See also:
`FilePrint(), FilePrintVisible()`

FilePrintVisible()

This function prints the current view as it appears on the computer screen to the current printer. In a text window, this prints the lines in the current selection. If there is no selection, it prints the line containing the cursor.

```
Func FilePrintVisible({flags%});
```

`flags%` 0=portrait, 1=landscape, 2=current setting. If omitted, the current value is used.

Returns The function returns 0 if all went well; otherwise it returns a negative error.

See also:
`FilePrint(), FilePrintScreen()`

FileQuit()

This is equivalent to the File menu Exit command. You are asked if you wish to save any unsaved data before the application closes. If the user cancels the operation (because there were files that needed saving), the script terminates, but the Spike2 application is left running. Use `FileClose(-1, -1)` before `FileQuit()` to guarantee to exit.

```
Proc FileQuit();
```

See also:
`FileClose(), Halt`

FileSave()

This function is equivalent to the File menu Save command and saves the current Result, XY, Grid, Text, Script, Output sequencer or newly sampled and unsaved time view as a file on disk. If the view has not been saved previously the File menu Save As dialog opens to request a file name. It cannot be used with external text or binary files. It cannot be used with a time view unless it has been sampled and has not been saved. If you want to save the view and provide a name as part of the script, use the `FileSaveAs()` command.

```
Func FileSave();
```

Returns The function returns 0 if the operation was a success, or a negative error code if the view type is not supported or an error during the save.

Use with a time view
You cannot use this command for a time view unless it has just been sampled; use `FileSaveAs()` instead. When used with a time view that has just been sampled, if an automatic file name is set, there is no prompt for a file name and the next automatic name is used. However, if automatic filing is enabled, the file is saved as soon as sampling stops, so this function is not needed.

See also:
`FileOpen(), FileSaveAs(), FileSaveResource(), FileClose(), SampleAutoFile(), SampleAutoName$()`

FileSaveAs()

This function saves the current view in its native format, as text, as an external format or as a picture. It is equivalent to the File menu Save As and Export As commands. This cannot be used for external text or binary files. This command can also name data files created with `FileNew(7,...)` and after sampling, but read the remarks about time views below. It can also be used with multimedia windows that display video to save the current frame.

```
Func FileSaveAs(name${,type%{,flags%{,text${,nCh%|expt%{,exp$|big%}}}});
```

`name$` The output file name including the file extension. This can be a path, but be careful using the backslash character. If the string is empty or it holds wild card characters * or ? or `text$` is not empty, then the File menu Save As dialog opens and `name$` sets the initial list of files. From version [8.00], if the name is not blank and there is no file extension (the name does not contain a '.' character), Spike2 will add one.

`type%` The type to save the file as (if omitted, type -1 is used):

-2 Valid for XY and result views only. Opens a File Save As dialog and allows the user to choose the file type to save as.

-1 Save in the native format for the view. You can also use this to save and name a Spike2 data file immediately after it has been sampled. This will also save any multimedia data associated with the file. The correct sequence is:

1. Use `FileNew(0, mode%)` to create a data file
2. Use `SampleStart()` to start data capture
3. When capture is over use `SampleStop()` to stop capture, or wait for the `SampleStatus()` to indicate that sampling has finished.
4. Use `FileSaveAs(name$,-1)` to set the file name
5. Use `FileClose()` to close the view

This command can also name a file opened with `FileNew(7,...)`. Once a data file has been named you must use type 0 to save a selection of channels to a new file; you cannot use type –1 again in a time view. Use of this command causes Spike2 to check the file length. Only channels that have been saved to disk are taken into account, so a file that contains only virtual and memory channels will be displayed as if it had zero length after using `FileSaveAs(name$, -1)`. You can use `MaxTime(-1)` to reassess file length after deleting disk-based channels.

0 Save sections set by `ExportChanList()` to a new Spike2 data file. This will not save associated multimedia files.

1 Save the contents of the current view as a text file. If the current view is a time window, Spike2 saves the channels set by `ExportChanList()` in the text format set by `ExportRectFormat()` or `ExportTextFormat()` and `ExportChanFormat()`. See `flag%` for result and XY views.

2 Save a text view as an output sequence file.

3 Save a text view as a Script file.

4 Save as a result view (result views only).

5 Save the screen image of a time, result or XY view as a metafile. The file extension sets the format; use `WMF` for a Windows metafile and `EMF` for an Enhanced metafile. We strongly recommend `EMF` (which support many more graphical features) unless you are forced to use `WMF`.

6 Save sampling configuration in a configuration file. See here for information on file extensions, a possible problem and how to convert old format files.

12 Save as XY view (XY views only).

13 Save the time, result, XY or multimedia (video only) view as a bitmap. Video frames are saved at the recorded resolution.

14 As 13, but saving in Joint Photographic Expert Group (JPEG) format.

15 As 13, but saving in Portable Network Graphics (PNG) format.

16 As 13, but saving in Tagged Image File Format (TIFF).

17 Save as Grid view (Grid views only).

18 As 13, but saving in Graphics Interchange Format (GIF) format. Note: in Signal (a CED application with a similar script language), the code for GIF format is 19.

100 Types 100 upwards identify external exporters installed in the `export` folder (where you will find additional documentation). Time views export the channels and time range set by `ExportChanList()`. Result and XY views export the data set by `flag%`. Codes defined so far:

Code	Exporter
100	MatLab follow this link to see the string format used for the `exp$` argument (below).

`flags%` In Spike2 version 7 this argument was called `yes%`. From version 8 it is the sum of the following flags:

1 If set, do not ask the user if they want to overwrite an existing file with the same name. Old scripts that set `yes%` to 0 or 1 will see no change in behaviour.

8 Determines the start folder if the system File Open dialog is used and `name$` does not specify a path. Add 8 to follow the operating system rules, usually opening the last File Open dialog folder. Otherwise open the current folder as set by `FilePathSet()`. The value 8 is used to match `FileOpen()`.

`text$` An optional prompt displayed as part of the file dialog to prompt the user. The File Save As dialog will appear if `text$` is not empty.

`nCh%` Used when `type%` is 0 to set the number of channels in the exported file in the range 32-2000. If omitted, the file has the same number of channels as the source. Spike2 version 4 could only read files with up to 100 channels, versions 5-8 read up to 400.

`expt%` In Spike2 version 7 this argument was called `flags%`. It is used when saving result and XY views as text or to an external exporter to override the normal behaviour, which is to save all channels and all data points. This optional argument has a default value of 0 and is the sum of:

1 Output only visible channels

2 Output only selected channels, or visible channels if no channel is selected

4 Output only data that is in the visible range (use 5 for visible data)

`exp$` When this is used to export data, this string sets additional export parameters. The string format depends entirely on the exporter (set by the `type%` argument), so see the exporter documentation for details.

`big%` Used when `type%` is 0 to determine what type of Spike2 data file is created. To force a type set: 0=very old format (`.smr`) data file limited to 2 GB is size, 1=.old format (`.smr`) with a maximum size of up to 1 TB (for Spike2 version 7), 2=modern 64-bit data file (`.smrx`). You can also omit this parameter, or set -1 in which case if the file name set ends with `.smr` we will behave as if 1 were set, otherwise we behave as if 2 were set. If a File Save dialog opens, this field determines which file extensions you can choose.

Returns The function returns 0 if the operation was a success, or a negative error code. Possible errors include:

-1512 Attempt to save a data file while it is sampling (wait until sampling has stopped)

-1513 The destination file exists and is read only

-1517 The operation was cancelled by the user (user decided not to overwrite the destination)

-1520 Attempt to save as an impossible type for example, XY view as a result view

-1523 The file name is already open in a window in Spike2 (often writing to the file that is being read)

-1542 Some elements of the path do not exist or there are illegal characters in the path

-1548 An error was detected during the file save operation

Use with a newly sampled time view

This command can save a time view that has just been sampled as described above for `type%` set to -1. However, if automatic filing is set with `SampleAutoFile()` or by the sampling configuration, the file is saved automatically as soon as sampling finishes and using a `type%` of -1 will generate an error.

When automatic file naming is enabled with `SampleAutoName$()` or with the sampling configuration, you can use this command to override the next sequential name, as long as the file has not already been saved and has finished sampling.

Clean up an existing data file or add channel space

If you have a file that has internal space for deleted channels, or you want to increase the number of channels you can store in the file you can use the following script to generate a new file just holding the channel data. This tests the current view to check it is a time view, then creates a copy of the file in the same folder and opens it.

```
if (ViewKind()) then Message("Not a time view"); halt endif;
const fName$ := FileName$(-1) + "copy.smrx";
ExportChanList(0); 'Initialise the channel list
ExportChanList(0, Maxtime(), -1); 'All channels
var ok% := FileSaveAs(fName$, 0); 'Export
if ok% = 0 then
    FileOpen(fName$, 0, 1);
    WindowVisible(1);
else
    Message(Error$(ok%));
endif
```

If you want to change the number of channels in the file, for example to 64 channels, change the **`FileSaveAs()`** line:

```
var ok% := FileSaveAs(fName$, 0, 0, "", 64); 'Export and set 64 channels
```

See also:
`EditCopy()`, `ExportChanFormat()`, `ExportChanList()`, `ExportTextFormat()`, `ExportRectFormat()`, `FileClose()`, `FileNew()`, `SampleAutoFile()`, `SampleAutoName$()`

FileSaveResource()

This saves a resource file for the current time, resource or XY view. If a time view is duplicated, resources for all the duplicates are saved.

```
Func FileSaveResource({name$|glob%})
```

`name$` The resource file to save to. If the name is `""` or contains a `"*"` or a `"?"`, the user is prompted for a file. Any extension in the file name is ignored and the extension is set to `.s2rx`. Remember that `\` in a string must be entered as `\\` or use `/`. If the name is `""` or contains a `"*"` or a `"?"`, the user is prompted for a file.

`glob%` An alternative to `name$`. 0=prompt for a file name, 1=save resources to the resource file associated with the view (this may be a global resource file), 2=save to the global resource file only (if enabled). If both `name$` and `glob%` are omitted, the effect is the same as setting `glob%` to 1.

Returns 0 if all was OK, -1 if the user cancelled the File Save dialog, -2 if the file could not be saved for any other reason.

This command will add information to an existing resource file, or create a new one.

See also:
`FileGlobalResource()`, `FileApplyResource()`, `FileOpen()`, `FileSave()`

FileSize()

This function returns the size of the data file associated with the current time view. During sampling this allows for data that is buffered, but not yet written. This function did not exist before Spike2 version [7.00].

```
Func FileSize();
```

Returns The size of the file, in bytes. This returns an integer (as integers are now 64-bits in size). In version 7 it returned a real number (as the value returned could exceed integer range). If you want your script to work in both versions 7 and 8 you should assign the result to a real number.

See also:
Big files, File size limit

FileStatus()

Function to report file information for a file system object defined by a path. This can be used to detect if a file exists, or if it is a folder or read-only. This was added at Spike2 version [10.05]. To get generate file attributes such as date and size of arbitrary file types, open them as a binary read only file and use the external file routines.

```
Func FileStatus(path$)
```

`path$` The path to the file system object to report on.

Returns An integer value with bits set to represent the properties of the file system object identified by `path$` or -1 if the object was not found. The result is the binary OR of the following values:

1	0x001	Read only. Applications can read the file but cannot modify or delete it.
2	0x002	Hidden. Not included in ordinary directory listings.
4	0x004	System. A file that belongs to the operating system.
8	0x008	Unused. This flag is reserved for future use.
16	0x010	Directory (folder)
32	0x020	Archive file or directory. Typically used to mark files for backup or removal.
64	0x040	Device. Reserved for system use.
128	0x080	Normal. Marks a file with no other attributes set, only valid when used alone.
256	0x100	Temporary file. This may never be buffered in cache memory and never written to disk.

See also:
`FilePath$(), FilePathSet(), FileList(), FileDelete()`

FileTime$()

In a time view, this function returns a string holding the time at which sampling started. In other views it gets a time the operating system associates with the disk file attached to the view, if it exists. Use `FileTimeDate()` to get the date as numbers. Default argument values are shown **bold**. If there is no time stored, the result is an empty string.

```
Func FileTime$({tBase%{, show%{, amPm%{, sep$${, offs{, what%}}}}}});
```

`tBase%` Specifies the time base to show the time in. You can choose between 24 hour or 12 hour clock mode. If this argument is omitted, 0 is used.

0 Operating system settings 2 12 hour format

1 24 hour format

`show%` Specifies the time fields to show. Add the values of the required options together and use that as the argument. If this argument is omitted or a value of 0 is used, 7 (1+2+4) is used for 24 hour format and 15 (1+2+4+8) for 12 hour format.

1	Show hours	4	Show seconds	16	Show milliseconds (new at [10.0])
2	Show minutes	8	Remove leading zeros from hours		

`amPm%` This sets the position of the "AM" or "PM" string in 12 hour format and has no effect in 24 hour format. If omitted, a value of zero is used. The string that gets printed ("AM" or "PM") is specified by the operating system.

0 Operating system settings 2 Show to the left of the time
1 Show to the right of the time 3 Hide the "`AM`" or "`PM`" string

`sep$` This string appears between adjacent time fields. If `sep$` = "`:`" then the time will appear as `12:04:45`. If an empty string is entered or `sep$` is omitted, the operating system settings are used.

`offs` If present, this is a time offset, in seconds, to add to the file start time. If omitted or negative it is treated as zero. You can use this to print the time of day at which something happened in the file. Before version [10.0] this was an integer argument; to print a time with fractional seconds, you had to arrange this yourself by adding the fractions to the formatted time.

`what%` Optional, with default value 0 and ignored for a time view. This chooses which of the times associated with the file to get:

0 The last modified time.
1 The file created time.
2 The last file access time.

Return The time as a string, or an empty string if we failed to get a useful time.

Changes at Spike2 9.04

Prior to version [9.04], the `what%` argument did not exist and the command could only be used with a time view.

See also:

`Time$(), FileDate$(), FileTimeDate()`

FileTimeBase()

This function gets and optionally sets the microseconds per time unit value for the data file associated with the current time view (the clock tick period). Changing the clock tick stretches or shrinks the time axis for all data items in the file. Only the data file header is changed by this operations and you must save the file when you close it or any changes will be lost.

When you change the time base we attempt to keep the displayed data looking sensible. The same data is displayed and we also (from version 8.06a onwards) scale the positions of vertical cursors. *Setting this value will very likely make nonsense of any derived views or dialog boxes related to the view and Spike2 makes no attempt to sort this out*.

```
Func FileTimeBase({new});
```

new If present, and in the range 0.001 to 32767, sets the number of microseconds per clock tick in the file. If the file is sampling, rerunning or write protected, this change has no effect.

Returns The microseconds per clock tick at the time of the call.

An example

Consider a situation where both Spike2 and another system sample data from the same experiment, but the two systems are not synchronised. Given that modern electronics are pretty accurate, we would expect that the two systems would both log data at an accurate clock tick, but that the actual durations of a second as measured on both systems would probably differ, perhaps by as much as one part in 50,000. This is a small difference over 1 second (20 microseconds), but over 30 minutes this would be 36 milliseconds, which might be significant. Let us further assume that this difference stays more or less constant with time, that we can import the data from the other system into a Spike2 data file, and that we have a common marker recorded at the start on both systems and at the end. We want to generate a single file holding the data from both files with all the times synchronised as closely as possible. We do this in the following steps:

1. There is likely to be a time offset between the start markers in both files. This can be eliminated by exporting the data in the files so that the start marker in each file is at time 0. See the `flags%` argument in `ExportChanList()` to do this from a script.

2. Next, work out the time scaling between the two files by looking at the ratio of the end time marker minus the start time marker between the two files. Suppose the time interval is `t1` for file 1 and `t2` for file 2. Unless you are very lucky, `t1` and `t2` will not be the same. To modify file 2 to have the same time interval as file 1 we would make file 2 current and then use:

```
FileTimeBase(FileTimeBase()*t1/t2); 'change the time base
```

3. If you want to amalgamate the two files into one and there are sufficient free channels in one of the files, use `ChanSave()` to move all the channels of one file into free channels in the other. `ChanSave()` adjusts event times and interpolates waveforms to compensate for time base differences between files. If you need to create a file with more free channels, you can do this by exporting one of the files and setting the desired number of channels.

See also:
`FileNew(), ChanSave(), ExportChanList()`

FileTimeDate()

This command gets a time and date associated with the file in the current view. In a Time view, it returns the time and date at which sampling started as numbers. Use `FileTime$()` and `FileDate$()` to get the result as strings. In other views, if there is a data file on disk, this attempts to get one of the time and date stamps stored by the operating system as set by the `what%` argument. This does not work for multimedia windows.

See the `TimeDate()` command to get the current system time in the same format. If there is no information, the returned values are all zero.

```
Func FileTimeDate(&s%|&s{, &m%{, &h%{, &d%{, &mon%{, &y%{, &wDay%{, what
%}}}}}});
Func FileTimeDate(td%[]{, what%})
```

`s{%}` If this is the only argument, it is set to the number of seconds since midnight. Otherwise it is set to the number of seconds since the start of the current minute. From version [10.0] onwards, this can be a real variable, which will return fractional seconds, when available.

`m%` If this is the last argument, it is set to the number of minutes since midnight. Otherwise it is set to the number of minutes since the start of the hour.

`h%` If present, the number of hours since Midnight is returned in this variable.

`d%` If present, the day of the month is returned as an integer in the range 1 to 31.

`mon%` If present, the month number is returned as an integer in the range 1 to 12.

`y%` If present, the year number is returned here. It will be an integer such as 2002.

`wDay%` If present, the day of the week will be returned here as 0=Monday to 6=Sunday.

`td%[]` If an array is the first and only argument, the first seven elements are filled with time and date data. The array can be less than seven elements long. Element 0 is set to the seconds, 1 to the minutes, 2 to the hours, and so on. From version [10.0], if the array is eight elements long, `td%[7]` is set to the number of milliseconds, when available.

`what%` Optional, with default value 0 and ignored for a time view. This chooses which of the times associated with the file to get:

- 0 The last modified time.
- 1 The file created time.
- 2 The last file access time.

If you want to find the operating system times for a time view, close the view and re-open the file as an external binary file in read only mode, then collect the time and close it. If there is demand to make this work for a time view we will consider extending the command.

Returns 0 if we failed to get a time, 1 for success.

Changes at Spike2 9.04

Prior to version [9.04], this was a `Proc` (so no return value), the `what%` argument did not exist and the command could only be used with a time view.

Multimedia files

To get date and time information for a multimedia file you can open it as a read-only external binary file, get the time and date, then close it. This works before, during or after using it as a multimedia file.

See also:
`Date$(), FileTimeDateSet(), MaxTime(), Seconds(), Time$(), TimeDate()`

FileTimeDateSet()

This command sets the sampling start time for the file associated with the current time view. This time has been stored in all Spike2 data files since version 4.03. If you create a file using the script language and do not use this command, the sampling start time is set to the time of creation of the file.

```
Func FileTimeDateSet(s, m%, h%, d%, mon%, y%);
Func FileTimeDateSet(const td%[])
```

`s` Sets the seconds in the range 0-59. From version [10.0], fractional seconds can be set (currently stored to an accuracy of 10 milliseconds).

`m%` Sets the minutes in the range 0-59.

`h%` Sets the hours in the range 0-23.

`d%` Sets the day of the month in the range 1 to 31.

`mon%` Sets the month of the year in the range 1 to 12.

`y%` Sets the year in the range 1980 to 2100. This was relaxed to 1970-2200 at version [10.18]. Spike2 did not exist before 1980, but you might import some older data. The Windows/Unix systems have a time stamp that is the count of seconds since Jan 1, 1970, so allowing 1970 seems reasonable. The high limit is just to prevent bad values getting past; I doubt that this software will still be in use in 2100, but you never know.

`td%[]` If an array is the first and only argument, the first size elements set the time and date data. Element 0 sets the seconds, 1 sets the minutes, 2 sets the hours, 3 sets the days, 4 sets the month and 5 sets the years. The values should be in the same range as for the `s%` to `y%` arguments.

From Spike2 version [10.0], `td%[6]`, if present, sets the milliseconds. Time views store the start time to a resolution of 10 milliseconds.

Returns If all the values are in the correct ranges and the file is not read only, the new data is set and the return value is 0. If there is a problem, the return value is a negative error code.

See also:
`Date$(), FileTimeDate(), MaxTime(), Seconds(), Time$(), TimeDate()`

FiltApply()

Applies a set of FIR filter coefficients or a filter in the FIR filter bank to a source waveform or RealWave channel in the current time view and places the result in a destination memory buffer or a disk-based channel. The output can be written as either a waveform or as a RealWave channel.

Each output point is generated from the same number of input points as there are filter coefficients. Half these points are before the output point, and half are after. Where more data is needed than exists in the source file (for example at the start and end of a file and where there are gaps), extra points are made by duplicating the nearest valid point.

```
Func FiltApply(n%|const coef[], dest%, srce%, sTime, eTime{, flags%});
```

`n%` Index of the filter in the filter bank to apply in the range -1 to 11. If you use -1 for the temporary filter you must have created it first (see `FiltCreate()`).

`coef[]` An array holding a set of FIR filter coefficients to apply to the waveform. The size of the passed array sets the number of coefficients.

`dest%` The channel to hold the filtered waveform: either an unused disk channel, a memory channel with the same sampling frequency as `srce%` or 0 to create a compatible memory channel and place the filtered waveform in it. When a new channel is created, the channel settings are copied from the old channel.

`srce%` The source waveform or RealWave channel. There must be at least half the number of sampling coefficients worth of data points before `sTime` if the output is to start at `sTime`. Similarly, the channel must extend for the same number of data points beyond `eTime` if the output is to extend to `eTime`.

`sTime` Time to start the output of filtered data. There is no output for areas where there is no input data. If the filter has an even number of coefficients, the output is shifted by half a sample relative to the input.

`eTime` The end of the time range for filtered data.

`flags%` Optional, default value 0. The sum of the following flag values:

- 1 Optimises the destination channel scale and offset values to give the best possible representation of the output as 16-bit integers. For Waveform output channels, this doubles the processing time and existing data in the output channel that is not overwritten is also rescaled. If you do not re-scale, the channel's scale and offset are unchanged. However, you run the risk of the output being clipped to the 16-bit range allowed for a waveform channel
- 2 Create a RealWave destination channel in place of a Waveform channel.
- 4 Display a progress dialog (allowing the user to cancel the filter) if the filter takes more than a second or so. This option was added at [10.03]. You can set this value in previous versions of Spike2 without error, however it has no effect. It records as set (the interactive dialog allows a progress bar).

Returns The channel number that the output was written to or a negative error code. A negative error code is also returned if the user clicks Cancel from the progress bar if `flags%` allows it or if `dest%` is a disk channel that is in use. Delete an existing channel with `ChanDelete(dest%)`.

See also:
Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `ChanDelete(), FiltAtten(), FiltCalc(), FiltComment$(), FiltCreate(), FiltInfo(), FiltName$(), FiltRange()`

FiltAtten()

This set the desired attenuation for a filter in the filter bank. When `FiltApply()` or `FiltCalc()` is used, the number of coefficients needed to achieve this attenuation will be generated. A value of zero sets the attenuation back to the default (-65 dB).

```
Func FiltAtten(index%{, dB});
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`dB` If present and negative, this is the desired attenuation for stop bands in the filter.

Returns The desired attenuation for a filter at the time of the call.

See also:
Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply(), FiltCalc(), FiltComment$(), FiltCreate(), FiltInfo(), FiltName$(), FiltRange()`

FiltCalc()

The calculation of filter coefficients can take an appreciable time; this routine forces the calculation of a filter for a particular sampling frequency if it has not already been done. If you do not force the calculation, you can still use `FiltApply()` to apply a filter. However, the coefficient calculation will then be done at the time of filter application, which may not be desirable if the filtering operation is time critical.

```
Func FiltCalc(index%, sInt{, coeff[]{, &dBGot {, nCoef%}}});
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`sInt%` The sample interval of the waveform you are about to filter. This is the value returned by `BinSize()` for a waveform channel.

`coeff` An array to be filled with the coefficients used for filtering. If the array is too small, as many elements as will fit are set. The maximum size needed is 2047 (but most filters require far fewer coefficients).

`dBGot` If present, set to the attenuation in negative dB attained by the filter coefficients.

`nCoef%` If present, sets the number of coefficients used in the calculation (use an even number for a full differentiator and an odd number for all other filter types).

Returns The number of coefficients generated by the filter. If you are passing the result to `FiltApply()`, use `coef[:n]`, not `coef[]` to pass `n` coefficients.

An example

Suppose the first filter in the bank (index 0) is a low pass filter with the pass band edge at 50 Hz. If we know that we will need to filter a channel 4 (sampled at 200 Hz) with this filter, we may want to calculate the coefficients needed in advance:

```
FiltCalc(0, BinSize(4));
```

This will calculate a filter corresponding to the specification of filter 0 for a sampling frequency of 200 Hz with an attenuation in the stop band of at least the current desired attenuation value for this filter.

Constraints on filters

The calculation of coefficients is a complex process and can produce silly results due to floating point rounding errors in some situations. To ensure that you will always get a useful result there is a limit to how small and how big a transition gap can be, relative to the sampling frequency. There is a similar limit on the width of a pass or stop band:

- The transition gap and the width of a pass or stop band cannot be smaller than 0.0025 of the sampling frequency.
- The transition gap cannot be larger than 0.12 of the sampling frequency.

This function always calculates a set of coefficients, but may alter the filter specification in order to do it (these changes are temporary, see later). This can happen in two cases:

1. If the sampling frequency is such that to produce the filter, the transition gap and/or pass and stop band widths are outside their limits, then the widths are set to the limits before calculating the filter. In our 50 Hz low pass filter example, if we calculate it with respect to a 12 kHz sampling frequency, the minimum pass band width is 12000*0.0025 = 30 Hz. So, the filter would be changed to a 60 Hz low pass filter.
2. If half the sampling frequency (the Nyquist frequency) is less than an edge of a pass or stop band, certain attributes of the filter are lost. In our 50 Hz low pass filter example, if we tried to calculate with a sampling frequency of 80 Hz, we would see that the Nyquist frequency is 40 Hz. No frequency above 40 Hz can be represented in a waveform sampled at 80 Hz, so a 50 Hz low pass filter is equivalent to an “All pass” filter. The filter specification will be altered to reflect this before calculating.

Any changes made to a filter specification to accommodate a particular calculation are made with reference to the original specification, not the specification that was last used for a calculation.

See also:

Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltComment$()`, `FiltCreate()`, `FiltInfo()`, `FiltName$()`, `FiltRange()`

FiltComment$()

This function gets and sets the comment associated with a filter in the filter bank.

```
Func FiltComment$(index% {, new$});
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`new$` If present, sets the new comment.

Returns The previous comment for the filter at the index.

See also:
Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltCalc()`, `FiltCreate()`, `FiltInfo()`, `FiltName$()`, `FiltRange()`

FiltCreate()

This function creates a FIR filter in the filter bank to the supplied specification and gives it a standard name and comment.

```
Func FiltCreate(index%, type%{, trW{, edge1{, edge2{, ...}}}});
```

`index%` Index of the new filter in the filter bank in the range -1 to 11. This action replaces any existing filter at this index.

`type%` The type of the filter desired (see table).

`trW` The transition width of the filter. This is the frequency interval between the edge of a stop band and the edge of the adjacent pass band.

`edgeN` This is a list of edges of pass bands in Hz. See the table.

Returns 0 if there was no problem or a negative error code if the filter was not created.

This table shows the relationship between different filter types and the meaning of the corresponding arguments. The numbers in brackets indicate the nth pass band when there is more than 1. An empty space in the table means that the argument is not required.

type%		trW	edge1	edge2	edge3	edge4
0	All stop					
1	All pass					
2	Low pass	Yes	High			
3	High pass	Yes	Low			
4	Band pass	Yes	Low	High		
5	Band stop	Yes	High(1)	Low(2)		
6	Low pass differentiator	Yes	High			
7	Differentiator					
8	1.5 Band Low pass	Yes	High(1)	Low(2)	High(2)	
9	1.5 Band High pass	Yes	Low(1)	High(1)	Low(2)	
10	2 Band pass	Yes	Low(1)	High(1)	Low(2)	High(2)
11	2 Band stop	Yes	High(1)	Low(2)	High(2)	Low(3)

The values entered correspond to the text fields shown in the Filter edit dialog box.

See also:
Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltCalc()`, `FiltComment$()`, `FiltInfo()`, `FiltName$()`, `FiltRange()`

FiltInfo()

Retrieves information about a filter in the bank.

```
Func FiltInfo(index%{, what%});
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`what%` Which bit of information about the filter to return:

-2	Maximum `what%` number allowed
-1	Desired attenuation in negative dB
0	type (if you supply no value, 0 is assumed)
1	Transition width
2-5	`edge1`-`edge4` given in `FiltCreate()`

Returns The information requested as real number.

See also:
Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltCalc()`, `FiltComment$()`, `FiltCreate()`, `FiltName$()`, `FiltRange()`

FiltName$()

This function gets and/or sets the name of a filter in the filter bank. The name of a filter is reset each time `FiltCreate()` is called.

```
Func FiltName$(index%{, new$});
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`new$` If present, sets the new name.

Returns The previous name of the filter at that index.

See also:
Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltCalc()`, `FiltComment$()`, `FiltInfo()`, `FiltCreate()`, `FiltRange()`

FiltRange()

Retrieves the minimum and maximum sampling rates that this filter can be applied to without the specification being altered. See the `FiltCalc()` command, Constraints on filters for more information.

```
Proc FiltRange(index%, &minFr, &maxFr);
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`minFr` Returns the minimum sampling frequency you can calculate the filter with respect to, so that no transition width is greater than the maximum allowed and that no attributes of the filter are lost.

`maxFr` Returns the maximum sampling frequency you can calculate the filter with respect to without the transition (or band) widths being smaller than allowed.

It is possible to create a filter that cannot be applied to any sampling frequency without being changed. This will be apparent because `minFr` will be larger than `maxFr`.

See also:
Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltCalc()`, `FiltComment$()`, `FiltInfo()`, `FiltCreate()`, `FiltName$()`

FIRMake()

This function creates FIR filter coefficients and places them in an array ready for use by `ArrFilt()`. Unless you need precise control over all aspects of filter generation, you may find it easier to use `FiltCalc()` or `FIRQuick()`. You will need to read the detailed information about FIR filters in the description of the Digital Filter dialog to get the best results from this command.

```
Proc FIRMake(type%, param[][], coef[]{, nGrid{, extFr[]}});
```

`type%` The type of filter file to produce: 1=Multiband filter, 2=Differentiator, 3=Hilbert transformer, 4=Multiband pink noise (Multiband with 3 dB per octave roll-off).

`param` This is a 2-dimensional array. The size of the first dimension must be 4 or 5. The size of the second dimension (`n`) should be the number of bands in your filter. You pass in 4 values for each band (indices 0 to 3) to describe your filter:

Indices 0 and 1 are the start and end frequency of each band. All frequencies are given as fraction of a sampling frequency and so are in the range 0 to 0.5.

Index 2 is the function of the band. For all filter types except a differentiator, this is the gain of the filter in the band in the range 0 to 1; the most common values are 0 for a stop band and 1 for a pass band. For a differentiator, this is the slope of the filter in the band, normally not more than 2. The gain at any frequency *f* in the band is given by *f*function*.

Index 3 is the relative weight to give the band. The weight sets the relative importance of the band in multiband filters. The program divides each band into frequency points and optimises the filter such that the maximum ripple times the weight in each band is the same for all bands. The weight is independent of frequency, except in the case of the differentiator, where the weight used is weight/frequency.

If there is an index 4 (the size of the first dimension was 5), this index is filled in by the function with the ripple in the band in dB. Because of this, the array cannot be `const`. In a pass band, the ripple is the ratio of the variation in the signal to the size of the signal. In a stop band, the ripple is the stop band attenuation. If all bands have the same weight, the ripple will be the same in all bands, but it is usual to give stop bands more weight, resulting in more stop band attenuation.

`coef` An array into which the FIR filter coefficients are placed. The size of this array determines the number of filter coefficients that are calculated. It is important, therefore, to make sure this array is exactly the size that you need. The maximum number of coefficients is 2047.

`nGrid` The grid density for the calculation. If omitted or set to 0, the default density of 16 is used. This sets the density of test points in internal tables used to search for points of maximum deviation from the filter specification. The larger the value, the longer it takes to compute the filter. There is seldom any point changing this value unless you suspect that the program is missing the peak points.

`extFr` An optional array to return the list of extremal frequencies (the list of frequencies within the bands which have the largest deviation from the desired filter). If there are `n%` coefficients, there are `(n%+1)/2` extremal frequencies.

The parameters passed in must be correct or a fatal error results. Errors include: overlapping band edges, band edges outside the range 0 to 0.5, too many coefficients, differentiator slope less than 0. If the filter is not a differentiator the band function must lie between 0 and 1, the band weight must be greater than 0.

For example, to create a low pass filter with a pass band from 0 to 0.3 and a stop band from 0.35 to 0.5, and no return of the ripple, you would set up `param` as follows:

```
'                    From    To  Gain Weight
var param[4][2] := {{0.00, 0.30, 1.00,   1.0},
                    {0.35, 0.50, 0.00,  10.0}};
```

See also:

More about `FIRMake()` filter types, Interactive FIR filtering, Filter banks, Technical details of FIR filters, `ArrFilt()`, `FiltApply()`, `FiltCalc()`, `FIRQuick()`, `FIRResponse()`

FIRQuick()

This function creates a set of filter coefficients in the same way the `FIRMake()` does, but many of the parameters are optional, allowing the most common filters to be created with a minimal specification. In particular, you do not need to specify the number of coefficients.

```
Func FIRQuick(coef[], type%, freq {, width {, atten}});
```

`coef` An array into which the FIR filter coefficients are placed. The size of this array should be 2047 (was 255 in versions before 4.01). This is the maximum number of coefficients that can be created and this function reserves the right to return as many as it feels necessary up to that value to create a good filter.

`type%` This sets the type of filter to create. 0=Low pass, 1=High pass, 2=Band pass, 3=Band stop and 4=Differentiator.

`freq` This is a fraction of the sampling rate in the range 0 to 0.5 and means different things depending on the type of filter:

For Low pass, High pass and Differentiator types, this represents the cut-off frequency. This is the frequency of the higher edge of the first frequency band.

For Band pass and Band stop filters, this is the midpoint of the middle frequency band: the pass band in a Band pass filter, the stop band in a Band stop filter.

`width` For Low pass, High pass and Differentiator filters, this is the width of the transition gap between the stop band and the pass band. The default value is 0.02 and there is an upper limit of 0.1 on this argument.

For a Band pass, or Band stop filter, `width` is the width of the middle band. E.g. if you ask for a Pass band filter with the `freq` parameter to be 0.25 and the width to be 0.05, the middle pass band will be from 0.2 to 0.3. For these types of filter, you still need a positive transition width. This transition width is 0.02 and cannot be changed by the user.

`atten` The desired attenuation in the stop band in dB. The default is 50 dB. This is analogous to the desired attenuation in the `FiltAtten()` command.

Returns The number of coefficients calculated. If the array is not large enough the coefficient list is truncated (and the result is useless). If you are passing the result to `FiltApply()`, use `coef[:n]`, not `coef[]` to pass `n` coefficients.

Degenerate filters

If you request a combination of `freq` and `width` parameters that make no sense, the `FIRQuick` command will do the best it can to approximate your request.

If you set a low pass filter with `freq + 0.5*wide >= 0.5`, or a high pass filter with `freq - 0.5*width <= 0.0`, the filter degenerates to an all pass filter. If you set a high pass filter with `freq + 0.5*wide >= 0.5`, or a low pass filter with `freq - 0.5*width <= 0.0`, the filter degenerates to an all stop filter.

In a similar manner, a band pass filter can degenerate into a high pass, a low pass or an all pass filter and a band stop filter can degenerate into a low pass, a high pass or an all stop filter.

See also:

More about `FIRMake()` filter types, Interactive FIR filtering, Filter banks, Technical details of FIR filters, `ArrFilt(), FiltApply(), FiltCalc(), FIRMake(), FIRResponse()`

FIRResponse()

This function retrieves the frequency response of a FIR filter as amplitude or in dB given the filter coefficients. The amplitude response will always be positive unless you also supply a filter type, when it can be negative (meaning the output is inverted due to a phase shift).

```
Proc FIRResponse(resp[], const coef[]{, as%{, type%}});
```

`resp` The array to hold the frequency response. This array will be filled regardless of its size. The first element is the amplitude response at 0 Hz and the last is the amplitude response at the Nyquist frequency. The remaining elements are set to the response at a frequency proportional to the element position in the array.

`coef` The coefficient array calculated by `FIRMake()`, `FIRQuick()` or `FiltCalc()`.

`as%` If this is 0 or omitted, the response is in dB (0 dB is unchanged amplitude), otherwise as linear amplitude (1.0 is unchanged).

`type%` If present, sets the filter type. The types are the same as those supplied for `FIRQuick()`: 0=Low pass, 1=High pass, 2=Band pass, 3=Band stop and 4=Differentiator. If a type is given, the time to calculate the response is halved. If you are not sure what type of filter you have, or you have a type not covered by the `FIRQuick()` types, then omit `type%` or set it to -1. When a type is supplied, the result can be negative, meaning the signal is inverted.

FitCoef()

This command gives you access to the fit coefficients for the next `FitData()` fit. You can return the values from any type of fit and set the initial values and limits and hold values fixed for iterative fits. There are two command variants:

Set and get coefficients

This command variant lets you read back the current coefficient values and set the coefficient values and limits for iterative fitting:

```
Func FitCoef({num%{, new{, lower{, upper}}}});
```

`num%` If this is omitted, the return value is the number of coefficients in the current fit. If present, it is a coefficient number. The first coefficient is number 0. If `num%` is present, the return value is the coefficient value for the existing fit, or if there is no fit, the coefficient value that would be used as the starting point for the next iterative fit is returned.

`new` If present, this sets the value of coefficient `num%` for the next iterative fit.

`lower` If present, this sets the lower limit for coefficient `num%` for the next iterative fit. There is currently no way to read back the coefficient limits. There is also no check made that the limits are set to sensible values.

`upper` If present, this sets the upper limit for coefficient `num%` for the next iterative fit.

Returns The number of coefficients or the value of coefficient `num%`.

Get and set the hold flags

This command variant allows you to hold some coefficients at their current values during the next fit.

```
Func FitCoef(hold%[]);
```

`hold%` An array of integers to correspond with the coefficients. If the array is too long, extra elements are ignored. If it is too short, extra coefficients are not affected. Set `hold%[i%]` to 1 to hold coefficient `i%` and to 0 to fit it. If `hold%[i%]` is less than 0, the hold state is not changed, but `hold%[i%]` is set to 1 if the corresponding coefficient is held and to 0 if it is not held. `hold%[]` cannot be `const`.

Returns This always returns 0.

See also:

`FitData()`, `FitValue()`, `FitExp()`, `ChanFitCoef()`, More about curve fitting

FitData()

This function, together with `FitCoef()` and `FitValue()`, lets you apply the same fitting functions that are available for channel data to data in arrays. You supply arrays of x and y data points and an optional array holding the standard deviation of the input data point y values. There are three command variants:

Initialise fit information

The first variant sets the type of fit. If you select an iterative fit, the initial values of the fitting coefficients are reset to standard values and any "hold" flags set by `FitCoef()` are cleared. You can set your own initial values with the `FitCoef()` command or make a guess at the initial values when performing the fit.

```
Func FitData(type%, order%);
```

`type%` The fit type. 0=Clear any fit, 1=Exponential, 2=Polynomial, 3=Gaussian, 4=Sine, 5=Sigmoid.

`order%` If positive, this is the order of the fit, if negative it is minus the number of fitted coefficients. See the information about each fit for the allowed values for each fit type. If `type%` is 0 this argument is ignored and should be 0.

Returns The number of fit coefficients for the fit or a negative error code.

Exponential fit

This fits multiple exponentials by an iterative method. The data is fitted to the equation:

$y = \mathbf{a}_0 \exp(-x/\mathbf{a}_1) + \mathbf{a}_2 \exp(-x/\mathbf{a}_3)\ldots$ For an even number of coefficients

$y = \mathbf{a}_0 \exp(-x/\mathbf{a}_1) + \mathbf{a}_2 \exp(-x/\mathbf{a}_3)\ldots + \mathbf{a}_n$ For an odd number of coefficients

You can set up to 10 coefficients or orders 1 to 5. If you use a fit order, the number of coefficients is the order times 2. See the `FitExp()` command for more information. Coefficient estimates are effective for orders 1 and 2.

Polynomial fit

This fits $y = \mathbf{a}_0 + \mathbf{a}_1 x + \mathbf{a}_2 x^2 + \mathbf{a}_3 x^3 \ldots$ to a set of (x,y) data points. The fitting is by a direct method; there is no iteration. The fit order is the highest power of x to fit in the range 1 to 10. The number of coefficients is the fit order plus 1.

Gaussian fit

This fits multiple Gaussian curves by an iterative method. The data is fitted to the equation:

$y = \mathbf{a}_0 \exp(-½(x-\mathbf{a}_1)^2/\mathbf{a}_2^2) + \mathbf{a}_3 \exp(-½(x-\mathbf{a}_4)^2/\mathbf{a}_5^2) + \ldots$

The fitted parameters (coefficients) are the $\mathbf{a}_i$. You can fit up to 3 Gaussians (order 1 to 3). The number of coefficients is given by the fit order times 3. Coefficient estimates become less useful as the order increases.

Sine fit

This fits multiple sinusoids by an iterative method. The data is fitted to the equation:

$y = \mathbf{a}_0 \sin(\mathbf{a}_1 x+\mathbf{a}_2) + \mathbf{a}_3 \sin(\mathbf{a}_4 x+\mathbf{a}_5) + \ldots \{+ \mathbf{a}_{3n}\}$

The fitted coefficients are the $\mathbf{a}_i$. Angles are evaluated in radians. You can fit up to 3 sinusoids (order 3) and an optional offset. Although the function is given in terms of sine functions, you can easily convert to cosines by subtracting $\pi/2$ from the phase angle ($\mathbf{a}_2$, $\mathbf{a}_5$, $\mathbf{a}_8$) after the fit. The coefficient count can be set to 3, 4, 6, 7, 9 or 10. If you use orders, the number of coefficients is order times 3 and you cannot set an offset. A useful coefficient estimate is made for a single sinusoid fit.

Sigmoid fit

This fits a single Boltzmann sigmoid by an iterative method. The data is fitted to the equation:

$y = \mathbf{a}_0 + (\mathbf{a}_1 - \mathbf{a}_0)/(1 + \exp((\mathbf{a}_2 - x)/ \mathbf{a}_3))$

In terms of the fitted result, $\mathbf{a}_0$ and $\mathbf{a}_1$ are the low and high fitting limits, $\mathbf{a}_2$ is the X50 point and $\mathbf{a}_3$ is related to the reciprocal of the slope at the X50 point. The slope at the X50 point, where x is $\mathbf{a}_2$, is $(\mathbf{a}_1 - \mathbf{a}_0)/(4\ \mathbf{a}_3)$. You can set order 1 only, or 4 coefficients.

Perform the fit

This variant of the command does the fit set by the previous variant. Use the `FitCoef()` command to preset fit coefficients and to read back the result of the fit.

```
Func FitData(opt%, const y[], const x[]{, const s[]|s{, &err{, maxI%
{,&iTer%{, covar[][]}}}}});
```

`opt%` 1=Estimate the coefficients before fitting, 0=use current values. Note that the estimates are usually only useful for a small number of coefficients.

`y[]` An array of y values to be fitted.

`x[]` A corresponding array of x values.

`s[]|s` A corresponding array of standard deviations for the data points defined by `y[]` and `x`, or a single value, being the standard deviation of each point. If this value is omitted or set to 1.0, the result is a least squares fit. If standard deviations are supplied, the result is a chi-squared fit.

`err` If present, this optional variable is updated with the chi-squared or least-squares error between the fit and the data.

`maxI%` If present, this changes the maximum number of iterations from 100.

`iTer%` If present, this integer variable is updated with the count of iterations done.

`covar` An optional two dimensional array of size at least `[nCoef][nCoef]` that is returned holding the covariance matrix when the fit is complete. It is changed if the return value is -1, 0 or 1. However, the values it contains are probably not useful unless the return value is 0.

Returns 0 if the fit is complete, 1 if max iterations done, or a negative error code: -1=the fit is not making progress (results may be OK), -2=the fit failed due to a singular matrix, -5=the fit caused a floating point error, -6=too little data for the number of coefficients, -7=unknown fitting function, -8=ran out of memory during the fit (too many data points), -9=the fit was set up with bad parameters.

Get fit information

This variant of the command returns information about the current fit.

```
Func FitData({opt%});
```

`opt%` This determines what to return. `opt%` values match `ChanFit()`, where possible. The returned information for each value of `opt%` is:

opt%	Returns	opt%	Returns
0	Fit type of next fit	1	Fit order of next fit
-1	1=a fit exists, 0=no fit exists	-8	Not used
-2	Type of last fit or 0	-9	Not used
-3	Number of coefficients	-10	Not used
-4	Chi or least-squares error	-11	1=chi-square, 0=least-square
-5	Fit probability (estimated)	-12	Last fit result code
-6	Lowest x value fitted	-13	Number of fitted points
-7	Highest x value fitted	-14	Number of fit iterations used

Returns The information requested by the `opt%` argument or 0 if `opt%` is out of range.

FitExp()

This command fits multiple exponentials to arrays of x,y data points, with an optional weight for each point. Fitting is by an iterative method. The data is fitted to the equation:

$y = \mathbf{a}_0 \exp(-x/\mathbf{a}_1) + \mathbf{a}_2 \exp(-x/\mathbf{a}_3)\ldots$ For an even number of coefficients

$y = \mathbf{a}_0 \exp(-x/\mathbf{a}_1) + \mathbf{a}_2 \exp(-x/\mathbf{a}_3)\ldots + \mathbf{a}_n$ For an odd number of coefficients

The fitted coefficients are the $\mathbf{a}_i$; odd numbered $\mathbf{a}_i$ are assumed to be positive. You can fit up to 5 exponentials or 4 exponentials and an offset. However, experience shows that trying to fit more than two exponentials requires care. The fit from even two exponentials should be viewed with caution, especially if the odd coefficients are similar. The commands to implement this are:

Set up the problem

The first command sets the number of exponentials to fit and the data set to be fitted. You must call this function before you call any of the others.

```
func FitExp(nCoef%, const y[], const x[]{, const s[]|s});
```

`nCoef%` The number of coefficients to fit in the range 2 to 10. If this is even, the first form of the function above is used. If it is odd, the final coefficient is an offset.

`y[]` An array of y data values. The length of the array must be at least `nCoef%`.

`x[]` An array of x data values. The length of the array must be at least `nCoef%`.

`s` An optional argument which is either an array with one value for each y data point or a single value for all data points. If the value is the standard deviation, then the error value returned when you iterate to find the best fit is the chi-squared value and the fit is a chi-squared fit.

If this value is proportional to the error in the y values, then the fit is still a chi-squared fit, and the error returned is proportional to the chi-squared value. If you omit this array, the fit is a least-squares error fit, and the error value returned is the sum of the squares of the errors in the y values.

Returns The function returns 0. There is no other return value as all errors stop the script.

The number of data points is set by the smallest of the sizes of the `y[]`, `x[]` or `s[]` (if present) arrays. The number of data points must be at least the number of coefficients.

Set coefficient values and ranges

This variant of the function sets the initial value of each coefficient and optionally sets the range of allowed values. You can call this function at any time after the set up call and before the iterate call (below) has returned 0 indicating that the fit is completed.

It is important that you give reasonable initial values for the coefficients, especially when you fit more than one exponent. You should limit the odd coefficient values (the time constants) so that they cannot be zero and make sure that multiple exponents do not have overlapping ranges. If two exponents get similar values, the fit is degenerate and will wander around forever without getting anywhere. However, setting too rigid a range may damage the fitting process as sometimes the minimisation process has to follow a convoluted n-dimensional path to reach the goal, and the path may need to wander quite a bit. Let experience be your guide.

If you do not give starting values, the command will make a simplistic guess at the fitting values. As we expect that you know more about the "right" answer than the command does, we suggest that you set the values you want.

```
Func FitExp(coef%, value{, lo, hi});
```

`coef%` The coefficient to set. The first coefficient is number 0, the last is `nCoef%`-1.

`value` The initial value to assign to the coefficient. If you have set low and high limits, and the value is outside these limits, it is set to the nearer limit.

`lo,hi` If present, these two values set the acceptable range of values for this coefficient. If omitted, or if both values are set to the same value, there is no limit. The value of the coefficient is tested against these limits after each iterative step, and if it exceeds a limit, it is set to that limit.

Iterate to a solution

Once you have set up the problem and given initial values to your coefficients, you can start the iteration process that will move the coefficients from their starting values to new values that minimise the error (optionally scaled by the `s` argument).

```
func FitExp(a[], &err{, maxI%{, &iTer%{, covar[][]}}});
```

`a[]` An array of size at least `nCoef%` that is returned holding the current set of coefficient values. The first amplitude is in `a[0]`, the first exponent in `a[1]`, the second amplitude in `a[2]`, the second exponent in `a[3]` and so on.

`err` A real variable returned as the sum over the data points of $(y_x[i]-y[i])^2/s[i]^2$ if `s[]` is used or holding the sum of $(y_x[i]-y[i])^2$ if `s[]` is not used, where $y_x[i]$ is the value predicted from the coefficients at the x value `x[i]`.

`maxI%` This is the maximum number of fitting iterations to do before returning from the function. If you omit this value, the function sets 1000. You can set any value from 1 to 10000. If you set more than 10000, the number is limited to 10000.

`iTer%` An optional integer variable that is returned holding the number of iterations done before the function returned.

`covar` An optional two dimensional array of size at least `[nCoef%][nCoef%]` that is returned holding the covariance matrix when the fit is complete. It is not changed unless the function return value is 0 or -1.

Returns 1 if the number of iterations are exhausted and the fit has not converged, 0 if the fit has converged, -1 if the fit is not improving but the result may be OK. Other negative numbers indicate failure.

Select coefficients to fit

Normally the command fits all the coefficients, but you can use this variant to select the coefficients to fit. You can use this command at any time after you have set the problem until the iteration variant returns 0 or -1.

```
func FitExp(const fit%[]);
```

`fit%[]` An array of at least `nCoef%` integers. If `fit%[i]` is 0, coefficient `i` is held constant, otherwise it is fitted. If all elements are 0, all arguments are fitted.

The effect of this command persists until either the iteration variant returns a value less than 1, or you set up a new problem, or you call this variant again.

An example

This is a template for using these commands to fit all the coefficients:

```
const nData%:=50;         'set number of data elements
var x[nData%], y[nData%]; 'space for our arrays
var coefs[4],err;         'coefficients and squared error
...                       'in here goes code to get the data
FitExp(4, y[], x[]);      'fit two exponentials (no sigma array)
FitExp(0, 1.0, 0.2, 4);   'set first amplitude and limit range
FitExp(1, .01, .001, .03);'set first time constant and range
FitExp(2, 2.0, 0.1, 6);   'set second amplitude and limit range
FitExp(3, .08, .03, .15); 'set second time constant and range
repeat
   DrawMyData(coefs[], x[], y[]);  'A function you write to show progress
until FitExp(coefs[], err, 1) < 1;
DrawMyData(coefs[], x[], y[]);     'Show the final state
```

See also:

More about curve fitting, `ChanFit(), FitGauss(), FitLinear(), FitNLUser(), FitPoly(), FitSigmoid(), FitSin()`

FitGauss()

This command fits multiple Gaussians to x,y data points with an optional weight for each point. Fitting is by an iterative method. The input data is fitted to the equation:

$y = \mathbf{a}_0 \exp(-½(x-\mathbf{a}_1)^2/\mathbf{a}_2^2) + \mathbf{a}_3 \exp(-½(x-\mathbf{a}_4)^2/\mathbf{a}_5^2) + \ldots$

The fitted parameters (coefficients) are the $\mathbf{a}_i$. You can fit up to 3 Gaussians. The commands to implement this are:

Set up the problem

The first command sets the number of Gaussians to fit and the data set to be fitted. You must call this function before you call any of the others.

```
func FitGauss(nCoef%, const y[], const x[]{, const s[]|s});
```

`nCoef%` The number of coefficients to fit. The legal values are 3, 6 and 9 for one, two and three Gaussians.

`y[]` An array of y data values. The length of the array must be at least `nCoef%`.

`x[]` An array of x data values. The length of the array must be at least `nCoef%`.

`s` An optional argument which is either an array with one value for each y data point or a single value for all data points. If the value is the standard deviation, then the error value returned when you iterate to find the best fit is the chi-squared value and the fit is a chi-squared fit.

If this value is proportional to the error in the y values, then the fit is still a chi-squared fit, and the error returned is proportional to the chi-squared value. If you omit this array, the fit is a least-squares error fit, and the error value returned is the sum of the squares of the errors in the y values.

Returns The function returns 0. There is no other return value as all errors stop the script.

The number of data points is set by the smallest of the sizes of the `y[]`, `x[]` or `s[]` (if present) arrays. The number of data points must be at least the number of coefficients. If it is not, you will get a fatal error, so check this before calling the function.

Set coefficient values and ranges

This variant of the function sets the initial value of each coefficient and optionally sets the range of allowed values. You can call this function at any time after the set up call and before the iterate call (below) has returned 0 indicating that the fit is completed.

In this type of fitting it is very important that you give reasonable starting values for the coefficients. In particular, when fitting multiple Gaussians, it is usual that the centre of each distribution is easy to determine. If you can set the centres and limit them so that they cannot overlap, the fit usually will proceed without any problems, even for multiple Gaussians. If you do not give starting values, the command will make a simplistic guess at the fitting values. As we expect that you know more about the "right" answer than the command does, we suggest that you set the values you want.

```
func FitGauss(coef%, value{, lo, hi});
```

`coef%` The coefficient to set. The first coefficient is number 0, the last is `nCoef%`-1.

`value` The initial value to assign to the coefficient. If you have set low and high limits, and the value is outside these limits, it is set to the nearer limit.

`lo,hi` If present, these two values set the acceptable range of values for this coefficient. If omitted, or if both values are set to the same value, there is no limit. The value of the coefficient is tested against these limits after each iterative step, and if it exceeds a limit, it is set to that limit.

As long as you make a reasonable estimate of the centre points, there should be no problems fitting multiple Gaussians.

Iterate to a solution

Once you have set up the problem and given initial values to your coefficients, you can start the iteration process that will move the coefficients from their starting values to new values that minimise the error (optionally scaled by the `s` argument).

```
func FitGauss(a[], &err{, maxI%{, &iTer%{, covar[][]}}});
```

`a[]` An array of size at least `nCoef%` that is returned holding the current set of coefficient values. The first amplitude is in `a[0]`, the first centre in `a[1]`, the first sigma in `a[2]`, the second amplitude in `a[3]` and so on.

`err` A real variable returned as the sum over the data points of $(y_x[i]-y[i])^2/s[i]^2$ if `s[]` is used or holding the sum of $(y_x[i]-y[i])^2$ if `s[]` is not used, where $y_x[i]$ is the value predicted from the coefficients at the x value `x[i]`.

`maxI%` This is the maximum number of fitting iterations to do before returning from the function. If you omit this value, the function sets 1000. You can set any value from 1 to 10000. If you set more than 10000, the number is limited to 10000.

`iTer%` An optional integer variable that is returned holding the number of iterations done before the function returned.

`covar` An optional two dimensional array of size at least `[nCoef%][nCoef%]` that is returned holding the covariance matrix when the fit is complete. It is not changed unless the function return value is 0 or -1.

Returns 1 if the number of iterations are exhausted and the fit has not converged, 0 if the fit has converged, -1 if the fit is not improving but the result may be OK. Other negative numbers indicate failure.

Remember that even when a minimum is found, there is no guarantee that this is the minimum. It is the best minimum that this algorithm can find given the starting point.

Select coefficients to fit

Normally the command fits all the coefficients, but you can use this variant to select the coefficients to fit. You can use this command at any time after you have set the problem until the iteration variant returns 0 or -1.

```
func FitGauss(const fit%[]);
```

`fit%[]` An array of at least `nCoef%` integers. If `fit%[i]` is 0, coefficient `i` is held constant, otherwise it is fitted. If all elements are 0, all arguments are fitted.

The effect of this command persists until either the iteration variant returns a value less than 1, or you set up a new problem, or you call this variant again.

An example

This is a template for using these commands to fit all the coefficients:

```
const nData%:=50;           'set number of data elements
var x[nData%], y[nData%];   'space for our arrays
var s[nData%];              'space for sigma of each point
var coefs[4], err;          'coefficients and error squared
...                         'in here goes code to get the data
FitGauss(3, y[], x[], s[]); 'fit one gaussian
FitGauss(0, 1.0, 0.2, 4);   'set amplitude and limit range
FitGauss(1, 2, 1.5, 2.5);   'set centre of the gaussian and range
FitGauss(2, 0.5, 0.3, 1.9); 'set width and limit range
repeat
   DrawMyData(coefs[], x[], y[]);  'Some function to show progress
until FitGauss(coefs[], err, 1) < 1;
DrawMyData(coefs[], x[], y[]);     'Show the final state
```

See also:

More about curve fitting, `ChanFit(), FitExp(), FitLinear(), FitNLUser(), FitPoly(), FitSigmoid(), FitSin()`

FitLine()

This function calculates the least-squares best-fit line to a set of data points from a time view waveform, RealWave or WaveMark channel or a result view. See `ChanFit()` for XY views. It fits the expression:

$y = \mathbf{m}\,x + \mathbf{c}$

through the data points (x_i, y_i) so as to minimise the error given by:

$\mathrm{Sum}_i(y_i - \mathbf{m}\,x_i - \mathbf{c})^2$

In this expression, **m** is the gradient of the line and **c** is the y axis intercept when x is 0.

```
Func FitLine(chan%, start, finish, &grad{, &inter{, &corr}});
```

`chan%` A channel holding suitable data in the current time or result view.

`start` The start time for processing in a time view, the start bin in a result view.

`finish` The end time for processing in a time view, the end bin in a result view. Data at the `finish` time, or in the `finish` bin, is included in the calculation.

`grad` This is returned holding the gradient of the best fit line (m).

`inter` Optional, returned holding the intercept of the line with the y axis (c).

`corr` Optional, returned holding correlation coefficient indicating the "goodness of fit" of the line. Values close to 1 or -1 indicate a good fit; values close to 0 indicate a very poor fit. This parameter is often referred to as r in textbooks.

Returns 0 if all was OK, or -1 if there were not at least 2 data points.

The results are in user units, so in a time view with a waveform measured in Volts, the units of the gradient are Volts per second and the units of the intercept are Volts. In a result view, the units are y axis units per x axis unit. They are not y units per bin.

See also:
`ChanFit(), ChanMeasure(), FitLinear(), FitPoly(),` More about curve fitting

FitLinear()

This command fits $y = \mathbf{a}_0 f_0(x) + \mathbf{a}_1 f_1(x) + \mathbf{a}_2 f_2(x)\ldots$ to a set of (x,y) data points provided that the functions $fn(x)$ are linearly independent. If you can provide error estimates for each y value, you can use the covariance output from this command to provide confidence limits on the calculated coefficients and you can use the returned chi-square value to test if the model is likely to fit the data. The command is:

```
func FitLinear(coef[], const y[], const x[][]{, const s[]|s{, covar[][]{,
r[]{, mR}}}});
```

`coef[]` A real array which sets the number of coefficients to fit and which return the best fit set of coefficients. The array must be between 2 and 10 elements long. The coefficient a0 is returned in `coef[0]`, a1 in `coef[1]` and so on.

`y[]` A real array of y values.

`x[][]` This array specifies the values of the functions f(x) at each data point. If there are `nc` coefficients and `nd` data values, this array must be of size at least `[nd][nc]`. Viewed as a rectangular grid with the coefficients running from left to right and the data running from top to bottom, the values you must fill in are:

$$
\begin{array}{llllll}
f_0(x_0) & f_1(x_0) & f_2(x_0) & f_3(x_0) & \ldots & f_{nc-1}(x_0) \\
f_0(x_1) & f_1(x_1) & f_2(x_1) & f_3(x_1) & \ldots & f_{nc-1}(x_1) \\
f_0(x_2) & f_1(x_2) & f_2(x_2) & f_3(x_2) & \ldots & f_{nc-1}(x_2) \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
f_0(x_{nd-1}) & f_1(x_{nd-1}) & f_2(x_{nd-1}) & f_3(x_{nd-1}) & \ldots & f_{nc-1}(x_{nd-1})
\end{array}
$$

`s` This is either a real array holding the standard deviations of the `y[]` data points, or a real value holding the standard deviation of all data points. If `s` is omitted or zero, a least-squares error fit is performed, otherwise a chi-squared fit is done.

`covar` An optional two dimensional array of size at least `[nc][nc]` (`nc` is the number of coefficients fitted) that is returned holding the covariance matrix.

`r[]` An optional array of size at least `[nc]` (`nc` is the number of coefficients fitted) that is returned holding diagnostic information about the fit. The less relevant a fitting function $f_n(x)$ is to the fit, the smaller the value returned. The element of the array that corresponds to the most relevant function is returned as 1.0, smaller numbers indicate less relevance.

If your fitting functions are not independent of each other, several coefficients may have low `r` values. The solution is to remove one of the functions from the fit, or to set the `mR` argument to exclude one of the functions, then fit again. If the remaining coefficients become relevant, you have excluded a function that was a linear combination of the others. If the remaining coefficients still are not relevant, you have eliminated a function that did not contribute to the fit.

`mR` You can use this optional variable to set the minimum relevance for a function. Functions that have less relevance than this are "edited" out of the fit and their coefficient is returned as 0. If you do not provide this value, the minimum is set to 10^{-15}, which will probably not exclude any values.

Returns The function returns the chi-square value for the fit if `s[]` or `s` is given (and non-zero), or the sum of squares of the errors between the data points and the best fit line if `s` is omitted or is zero.

The smallest of the sizes of the `y[]` array (and `s[]` array, if provided) and the second dimension of `x[][]` sets the number of data points. It is a fatal error for the number of data points to be less than the number of coefficients.

An example

This demonstrates how to fit data to the function `y = a*sin(x/10) + b*cos(x/20)`. The x values vary from 0 to 49 in steps of 1. The function `MakeFunc()` calculates a trial data set plus noise. We set `s` to 1.0, so `FitLinear()` returns the sum of squares of the errors between the fitted and input data. If you run this example, you will notice that the returned value is slightly less than the sum of squares of the added errors.

```
const NCOEF%:=2,NDATA%:=50;' coefficient and data sizes
var x[NDATA%][NCOEF%];     ' array of function information
const noise := 0.01;       ' controls how much noise we add
var data[NDATA%], err := 0;' space for our function and errors

' Generate raw data. Fit y = a*sin(x/10)+b*cos(x/20)
var coef[NCOEF%], i%, r;   ' coefficients, index, random noise
coef[0]:=1.0; coef[1]:=2;  ' set coefficients for generated data
MakeFunc(data[], coef[], x[][]); ' Generate the data, then...
for i%:=0 to NDATA%-1 do   ' ...add noise to it
   r := (rand()-0.5)*noise;' the noise to add
   data[i%] += r;          ' add noise to the data
   err := err + r*r;       ' accumulate sum of squared noise
   next;

var covar[NCOEF%][NCOEF%]; ' covariance array
var sig2, a[NCOEF%];       ' sigma, fitted coefficients
var rel[NCOEF%];           ' array for "relevance" values
sig2 := FitLinear(a[], data[], x[][], 1, covar[][], rel[]);
Message("sig^2=%g, err=%g\ncoefs=%g\nrel=%g",sig2,err,a[],rel[]);
halt;

'y is the output array (x values are 0, 1, 2...), a is the array
'of coefficients. Y = a*sin(x/10)+b*cos(x/20)
proc MakeFunc(y[], a[], x[][])
var nd%,v;                 ' coefficient index, work space
for nd% := 0 to NDATA%-1 do
    v := Sin(nd% / 10.0);  ' first funcion
    x[nd%][0] := v;        ' save the value;
    y[nd%] := a[0] * v;    ' start to build the result
    v := Cos(nd%/20.0);    ' second function
    x[nd%][1] := v;        ' save it
    y[nd%] += a[1]*v;      ' full result
next;
end;
```

See also:

More about curve fitting, `ChanFit()`, `FitExp()`, `FitGauss()`, `FitNLUser()`, `FitPoly()`, `FitSin()`

FitNLUser()

This command uses a non-linear fitting algorithm to fit a user-defined function to a set of data points. The function to be fitted is of the form $y = f(x, \mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2 \ldots)$ where the $\mathbf{a}_i$ are coefficients to determine. You must be able to calculate the differential of the function f with respect to each of the coefficients. You can optionally supply an array to weight each data point. The commands to implement this are:

Set up the problem

The first command sets the user-defined function, the number of coefficients you want to fit, the number of data points and optionally, you can set the weight to give each data point. You must call this function before you call any of the others.

```
func FitNLUser(User(ind%, a[], dyda[]), nCoef%, nData%{, const s[]|s});
```

`User()` A user-defined function which is called by the fitting routine. The function is passed the current values of the coefficients. It returns the error between the function and the data point identified by `ind%` and the differentials of the function with respect to each of the coefficients at that point. The return value should be the y data value at the index minus the calculated value of the function at the x value, using the coefficients passed in.

`ind%` The index into the data points at which to evaluate the error and differentials. If there are `n` data points, `ind%` runs from 0 to `n`-1. You can rely on the function being called with the same coefficients as `ind%` increments from 0 to `n`-1, which may be useful if you have complex functions of the coefficients to evaluate.

`a` An array of length `nCoef%` holding the current values of the coefficients. The coefficients are refreshed for each call to the user-defined function, so it is not an error to change them; however this is usually not done. We may make this a `const` array in a future revision.

`dyda` An array of length `nCoef%` which your function should fill in with the values of the partial differential of the function with respect to each of the coefficients. For example, if you were fitting $y = \mathbf{a}_0$`*exp(-`$\mathbf{a}_1$`*x)` then set `dyda[0]` = $\delta y/\delta \mathbf{a}_0$ = `exp(-`$\mathbf{a}_1$`*x)` and `dyda[1]` = $\delta y/\delta \mathbf{a}_1 = -\mathbf{a}_0$`*`$\mathbf{a}_1$`*exp(-`$\mathbf{a}_1$`*x)`.

`nCoef%` The number of coefficients to fit in the range 1 to 10.

`nData%` The number of data points you will be fitting. If `s[]` is provided as an array, the value of `nData%` used is the smaller of `nData%` and the length of the `s[]` array. It is a fatal error for the number of data points used to be less than `nCoef%`.

`s` This argument is optional. It is either an array of weights to be given to each data point in the fit or a single weight to apply to all data points. If this value is the expected standard deviation of the y value of the data points, then the error value returned is the chi-squared value and the fit is a chi-squared fit. If this value is proportional to the expected error at the data point, then the fit is still a chi-squared fit, and the error returned is proportional to the chi-squared value. If you omit this argument, the fit is a least-squares error fit, and the error value returned is the sum of the squares of the errors in the y values.

Returns The function returns 0. There is no other return value as all errors stop the script.

Unlike the other fitting routines, you will notice that the x and y data values are not passed into the command. Instead, the user-defined function is passed an index to the data values. It is assumed that the data is accessible by the user function.

Due to restrictions in the implementation of the script language, you cannot debug through the user-defined function. If you set a break point in it, or attempt to step into it you will get errors. We recommend that you check the returned values from the user-defined function by calling it from your own script code.

Set coefficient values and ranges

This variant of the function sets the initial value of each coefficient and optionally sets the range of allowed values. You can call this function at any time after the set up call and before the iterate call (below) has returned 0 indicating that the fit is completed.

In this type of fitting it is very important that you give reasonable starting values for the coefficients. If you do not give starting values, the command will set them all to zero, which is unlikely to be correct.

```
func FitNLUser(coef%, value{, lo, hi});
```

`coef%` The coefficient to set. The first coefficient is number 0, the last is `nCoef%`-1.

`value` The initial value to assign to the coefficient. If you have set low and high limits, and the value is outside these limits, it is set to the nearer limit.

`lo,hi` If present, these two values set the acceptable range of values for this coefficient. If omitted, or if both values are set to the same value, there is no limit. The value of the coefficient is tested against these limits after each iterative step, and if it exceeds a limit, it is set to that limit.

Iterate to a solution

Once you have set up the problem and given initial values to your coefficients, you can start the iteration process that will move the coefficients from their starting values to new values that minimise the error (optionally scaled by the `s` argument).

```
func FitNLUser(a[], &err{, maxI%{, &iTer%{, covar[][]}}});
```

`a[]` An array of size at least `nCoef%` that is returned holding the current set of coefficient values.

`err` A real variable returned as the sum over the data points of $(y_x[i]-y[i])^2/s[i]^2$ if `s[]` is used or holding the sum of $(y_x[i]-y[i])^2$ if `s[]` is not used, where $y_x[i]$ is the value predicted from the coefficients at the x value `x[i]`.

`maxI%` This is the maximum number of fitting iterations to do before returning from the function. If you omit this value, the function sets 1000. You can set any value from 1 to 10000. If you set more than 10000, the number is limited to 10000.

`iTer%` An optional integer variable that is returned holding the number of iterations done before the function returned.

`covar` An optional two dimensional array of size at least `[nCoef%][nCoef%]` that is returned holding the covariance matrix when the fit is complete. It is not changed unless the function return value is 0 or -1.

Returns 1 if the number of iterations are exhausted and the fit has not converged, 0 if the fit has converged, -1 if the fit is not improving but the result may be OK, -2 if the matrix built to solve the problem is singular (indicating that the coefficients are not linearly independent). Other negative numbers indicate failures like out of memory.

Remember that even when a minimum is found, there is no guarantee that it is the minimum. It is the best minimum that this algorithm can find given the starting point.

return value -1

This can mean that you have started the fit at a minimum point in the error function and the system is unable to make more progress. You can sometimes recover by adjusting your initial guess. We have seen this when we start by fitting a model with n free parameters, then change to one with n+1 and the added parameter is similar in effect to one of the previous ones and we have started it at the same value as the previous one. The previous solution was at a minimum in the error space, and it happened that setting the two parameters the same ended up at the same minimum.

return value -2

The fitting process uses the user defined function to build a matrix, then inverts the matrix to solve the problem. It is possible for this matrix to be singular (which means that a factor used for division has become so small that the result has lost all accuracy). This is usually caused by the coefficients not being mathematically independent. If any coefficient can be derived from the others, it is not independent. If you have a non-independent coefficient, you must recast the equations to fix this. You can also get this situation if in some range of the calculation, a pair of coefficients become related (due to other terms becoming very small, for instance). You may be able to work around this by restarting with a different guess, or by iterating through the coefficients, holding one constant at a time.

Select coefficients to fit

Normally the command fits all the coefficients, but you can use this variant to select the coefficients to fit. You can use this command at any time after you have set the problem until the iteration variant returns 0 or -1.

```
func FitNLUser(const fit%[]);
```

`fit%[]` An array of at least `nCoef%` integers. If `fit%[i]` is 0, coefficient `i` is held constant, otherwise it is fitted. If all elements are 0, all arguments are fitted.

The effect of this command persists until either the iteration variant returns a value less than 1, or you set up a new problem, or you call this variant again.

See also:
Simple example, More complex example, More about curve fitting, `FitExp()`, `FitGauss()`, `FitLinear()`, `FitPoly()`, `FitSin()`

A simple example

The following is an example of using this set of commands to fit the user-defined function `y = a * exp(-b*x)`. In this example we generate some test data and add to it a random error. There are two coefficients to be fitted (a and b). To use `FitNLUser()` we need to calculate the differential of the function with respect to `a` and `b`:

```
dy/da = exp(-b*x)
dy/db = -x * a * exp(-b*x)
```

These are standard results. Differential calculus is too large a subject to attempt a summary in this help. If you need to refresh your knowledge, this reference may help.

```
const NDATA%:=100, NCOEF% := 2;
var x[NDATA%],y[NDATA%],i%;
for i% := 0 to NDATA%-1 do  ' Generate data
   x[i%] := i%;             ' a:=1, b:=0.05 and add some noise
   y[i%] := exp(-0.05*i%)+(rand()-0.5)*0.01;
   next;

FitNLUser(UserFnc, NCOEF%, NDATA%); 'Link user function
FitNLUser(0, 0.5, 0.01, 2);   'Set range of amplitude
FitNLUser(1, 0.01, 0.001, 1); 'Set range of exponent

var coefs[NCOEF%], err, iter%;
i% := FitNLUser(coefs[], err, 100, iter%);
Message("fit=%d, Err=%g, iter=%d, coefs=%g",i%,err,iter%,coefs[]);
halt;

' The user-defined function: y = a * exp(-b*x);
' dy/da = exp(-b*x), dy/db = -x * a * exp(-b*x)
func UserFnc(ind%, a[], dyda[])
var xi,yi,r;
xi := x[ind%];            ' local copy of x value
yi := y[ind%];            ' local copy of y value
dyda[0] := exp(-a[1]*xi); ' differential of y with respect to a
r := dyda[0] * a[0];      ' intermediate value
dyda[1] := -xi * r;       ' differential of y with respect to b
return yi-r;
end
```

The script is in 4 sections:

1. The header declares the number of data points and the number of coefficients plus space to store a set of x,y data points. It then fills in the data points with the function y = exp(-0.05*x) and adds some random numbers in the range -0.005 up to 0.005. This will be the test data for the fitting.
2. The second section sets up the problem by telling `FitNLUser()` the name of the user function, the number of data points and the number of coefficients. Then for each coefficient, it sets a starting guess value and a range of acceptable values. The better your initial guess is, the more likely you are to get a useful result. Sometimes you will know a likely range of values for each coefficient, on other occasions, you may be able to make some reasonable guess from the initial data. Conversely, if you make a perversely bad initial guess, you may get results that are totally useless. It is not a good idea to set limits that are too close to the guess; sometimes a function has to chase a minimum along a tortuous route and setting very tight limits may prevent it doing this.
3. The third section declares space for the coefficients, tells the function to iterate a maximum of 100 times and prints the results.
4. The fourth section holds the user function. This is called once for each data point per iteration, so try to minimise the calculations required. Within each iteration, it is called with the index argument having the values 0 to the number of data points minus 1. For each data point we must fill in the differential of the

fitting function with respect to each coefficient and we must return the error between the y value and the fitting function.

Fitting a Gabor function

This demonstrates the use of the `FitNLUser()` script function. This is used to fit to a one dimensional Gabor function (the product of a Gaussian and a cosine wave) defined as:

```
y(x) = b + a*exp(-g*x*x*0.5)*Sin(l*x+p)
```

where `b` is an offset, `a` is the amplitude, `g` is twice the variance of the Gaussian, `l` is 2*pi/wavelength of the sinusoid and p is the phase of the sinusoid. There are 5 parameters to fit, being: `b`, `a`, `g`, `l`, `p`. The script starts with a header that defines the number of data points and number of coefficients, plus how much noise to add (to make this a more realistic example). It also sets the values of the coefficients that we are to attempt to fit.

```
'$FitGABOR|Demonstration of using the NL fitting to fit a Gabor function
'Author: Greg Smith, Cambridge Electronic Design Limited
const NDATA%:=100, NCOEF% := 5;
var x[NDATA%],y[NDATA%],i%, xv;
' coef#      0        1        2       3       4
const b:=0.002, a:=1.00, g:=0.01, l:=0.1, p:=0.0;
const noise := 0.04;        ' maximum noise amplitude to add

' Phase 1. Generate some sample data at equal-spaced x values. If you want
'          other than x increments of 1, set xv to i%*width where width is
'          the desired x spacing. Note that random noise is added to the
'          data values.
for i% := 0 to NDATA%-1 do  ' Generate data
   xv := i%;
   x[i%] := i%;             ' a:=1, b:=0.05 and add some noise
   y[i%] := b + a*exp(-g*xv*xv*0.5)*Sin(l*xv+p)+(rand()-0.5)*noise;
next;

' Phase 2. Set up the problem. This tells the script language the name of
'          the function to use to get the data values, then sets a range of
'          acceptable values for each argument and a starting guess for each
'          value. It is MOST important that the starting guess is reasonable.
'          If it is not, the fitting may yield stupid results.
FitNLUser(UserFnc, NCOEF%, NDATA%); 'Link user function
FitNLUser(0, 0, -0.1, 0.1);   'Set range of offset b
FitNLUser(1, 0.3, 0.5, 2);    'Set range of amplitude a
FitNLUser(2, 0.04, 0.01, 0.1);'Set range of g (1/sigma squared)
FitNLUser(3, 0.03, 0.01, 0.3);'Set range of omega (frequency) l
FitNLUser(4, 1,    -3, 3);    'Set range of the phase p

' Phase 3. Solve the problem. This tells the system to improve the initial
'          guess. Each iteration will call the User function NDATA% times.
'          The result is 0 if it worked, err is the sum of squares of the
'          difference between the fitted curve and the raw data, iterations
'          is the number of times round the function went and coefs are the
'          fitted parameters.
var coefs[NCOEF%], err, iter%, covar[NCOEF%][NCOEF%];
i% := FitNLUser(coefs[], err, 100, iter%, covar[][]);

Message("fit=%d, Err=%.4f, iterations=%d, coefs=%.4f", i%, err ,iter% ,coefs[]);

'
' If you need to know the likely errors in the fitted parameters, this is
' available from the covar array (co-variance). The diagonal of the array
' holds the expected variance (sigma squared) of each parameter, on the
' assumption that the errors in the original data have a normal distribution.
'
' See the on-line documentation for the FitNLUser() script function and read the
' linked section "More about curve fitting" for additional information.
halt;
```

```
' The next section is the user-defined function. Differentials of functions with
' several factors will have common expressions that need only be calculated once.
' Although the functions is quite complicated, the calculations turn out to be
' relatively simple.
'
' The user-defined function: y = b + a*exp(-g*x*x*0.5)*Sin(l*x+p);
' dy/db = 1;
' dy/da = exp(-g*x*x*0.5)*Sin(l*x+p)
' dy/dg = -x*x*0.5*a*exp(-g*x*x*0.5)*Sin(l*x+p)
' dy/dl = a*x*exp(-g*x*x*0.5)*Cos(l*x+p)
' dy/dp = a*exp(-g*x*x*0.5)*Cos(l*x+p)

func UserFnc(ind%, a[], dyda[])
var yv,xi,yi,r, xsq2, gauss, angle;
xi := x[ind%];                  ' local copy of x value
yi := y[ind%];                  ' local copy of y value
xsq2 := xi*xi*0.5;              ' calculate once as used twice
gauss := Exp(-a[2]*xsq2);       ' calculate once as used twice
angle := a[3]*xi+a[4];          ' used by Sin() and Cos()

dyda[0] := 1;                   ' dy/db is 1
dyda[1] := gauss*Sin(angle);    ' exp(-g*x*x*0.5)*Sin(l*x+p)
dyda[2] := -xsq2*a[1]*dyda[1];  ' -x*x*0.5*a*exp(-g*x*x*0.5)*Sin(l*x+p)
dyda[4] := a[1]*gauss*Cos(angle); ' a*exp(-g*x*x*0.5)*Cos(l*x+p)
dyda[3] := dyda[4]*xi;          ' a*x*exp(-g*x*x*0.5)*Cos(l*x+p)
yv := a[0] + a[1]*dyda[1];
return yi-yv;
end
```

FitPoly()

This command fits the polynomial $y = \mathbf{a}_0 + \mathbf{a}_1 x + \mathbf{a}_2 x^2 + \mathbf{a}_3 x^3 \ldots$ to a set of (x,y) data points. If you can provide error estimates for each y value, you can use the covariance output from this command to provide confidence limits on the calculated coefficients and you can use the returned chi-squared value to test if the model is likely to fit the data. The command is:

```
func FitPoly(coef[], const y[], const x[]{, const s[]|s{, covar[][]}});
```

`coef[]` A real array that sets the number of coefficients and returns the coefficient values. The array must be between 2 and 10 elements long. The coefficient a0 is returned in `coef[0]`, a1 in `coef[1]` and so on.

`y[]` A real array of y values. The smaller of the sizes of the `x[]` and `y[]` arrays (and `s[]` array, if provided), sets the number of data points. It is a fatal error for the number of data points to be less than the number of coefficients.

`x[]` A real array of x values.

`s` This is an optional argument. It is either a real array holding the standard deviations of each of the `y[]` data points, or it is a real value holding the standard deviation of all of the data points. If the argument is omitted or set to zero, a least-squares error fit is performed, otherwise a chi-squared fit is done.

`covar` An optional two dimensional array of size at least `[nc][nc]` (`nc` is the number of coefficients fitted) that is returned holding the covariance matrix.

Returns The function returns chi-squared if `s[]` or `s` is given (and non-zero), otherwise it returns the sum of squares of the errors between the raw and fitted data.

An example

This example generates test data, adds random noise, then fits a polynomial to the data.

```
const NCOEF% := 5;        ' number of coefficients
const NDATA%:=50;         ' number of data points
var y[NDATA%], x[NDATA%]; ' space for x and y values for fit
const noise := 1;         ' noise to add
var err := 0.0;           ' will be sum of squares of added noise
var cf[NCOEF%], i%, r;
```

```
cf[0]:=1.0; cf[1]:=-80; cf[2]:=-2.0; cf[3]:=0.5; cf[4]:=-0.009;
MakePoly(cf[],x[],y[]);   ' generate ideal data as polynomial
for i%:=0 to NDATA%-1 do  ' now add some noise to it
   r := (rand()-0.5)*noise;
   y[i%] += r;            ' add noise to the data
   err += r*r;            ' sum of squares of added noise
   next;
var sig2, a[NCOEF%];      ' a[] will be the fitted coefficients
sig2 := FitPoly(a[], y[], x[]);
Message("sig2=%g, noise=%g\nfitted=%8.4f\nideal =%8.4f",
                                     sig2, err, a[], cf[]);
halt;
'a[]  input array of coefficients
'x[]  output x co-ordinates, y[] output data values
proc MakePoly(a[], x[], y[])
var i%,j%,xv,s;
for i% := 0 to Len(y[])-1 do
   s := 0.0;
   xv := 1;
   for j% := 0 to NCOEF%-1 do
      s += a[j%]*xv;
      xv *= i%;
      next;
   y[i%] := s;
       x[i%] := i%;
   next;
end;
```

See also:
More about curve fitting, `ChanFit(), FitExp(), FitGauss(), FitLinear(), FitNLUser(), FitSigmoid(), FitSin()`

FitSigmoid()

This command a single Sigmoid function to x,y data points with an optional weight for each point. Fitting is by an iterative method. The input data is fitted to the equation:

$y = \mathbf{a}_0 + (\mathbf{a}_1 - \mathbf{a}_0) / (1 + \exp((\mathbf{a}_2 - \mathbf{x}) / \mathbf{a}_3))$

The fitted parameters (coefficients) are the $\mathbf{a}_i$. You can only fit 1 Sigmoid .In terms of the fitted result, $\mathbf{a}_0$ and $\mathbf{a}_1$ are the low and high fitting limits, $\mathbf{a}_2$ is the X50 point and $\mathbf{a}_3$ is related to the reciprocal of the slope at the X50 point. The slope at the X50 point, where x is $\mathbf{a}_2$, is $(\mathbf{a}_1 - \mathbf{a}_0)/(4\ \mathbf{a}_3)$. The commands to implement this are:

Set up the problem

The first command sets the number of Sigmoid curves to fit and the data set to be fitted. You must call this function before you call any of the others.

```
func FitSigmoid(nCoef%, const y[], const x[]{, const s[]|s});
```

`nCoef%` The number of coefficients to fit. The only legal value is 4 for one Sigmoid.

`y[]` An array of y data values. The length of the array must be at least `nCoef%`.

`x[]` An array of x data values. The length of the array must be at least `nCoef%`.

`s` An optional argument which is either an array with one value for each y data point or a single value for all data points. If the value is the standard deviation, then the error value returned when you iterate to find the best fit is the chi-squared value and the fit is a chi-squared fit.

If this value is proportional to the error in the y values, then the fit is still a chi-squared fit, and the error returned is proportional to the chi-squared value. If you omit this array, the fit is a least-squares error fit, and the error value returned is the sum of the squares of the errors in the y values.

Returns The function returns 0. There is no other return value as all errors stop the script.

The number of data points is set by the smallest of the sizes of the y[], x[] or s[] (if present) arrays. The number of data points must be at least the number of coefficients. If it is not, you will get a fatal error, so check this before calling the function.

Set coefficient values and ranges

This variant of the function sets the initial value of each coefficient and optionally sets the range of allowed values. You can call this function at any time after the set up call and before the iterate call (below) has returned 0 indicating that the fit is completed.

In this type of fitting it is very important that you give reasonable starting values for the coefficients. In particular, when fitting a Sigmoid, it is usual that the two levels (a0 and a1) are easy to determine. If you can set the levels and limit them so that they cannot overlap, the fit usually will proceed without any problems. If you do not give starting values, the command will make a simplistic guess at the fitting values. As we expect that you know more about the "right" answer than the command does, we suggest that you set the values you want.

```
func FitSigmoid(coef%, value{, lo, hi});
```

coef% The coefficient to set. The first coefficient is number 0, the last is nCoef%-1.

value The initial value to assign to the coefficient. If you have set low and high limits, and the value is outside these limits, it is set to the nearer limit.

lo,hi If present, these two values set the acceptable range of values for this coefficient. If omitted, or if both values are set to the same value, there is no limit. The value of the coefficient is tested against these limits after each iterative step, and if it exceeds a limit, it is set to that limit.

As long as you make a reasonable estimate of the two levels, there should be no problems fitting a Sigmoid.

Iterate to a solution

Once you have set up the problem and given initial values to your coefficients, you can start the iteration process that will move the coefficients from their starting values to new values that minimise the error (optionally scaled by the s argument).

```
func FitSigmoid(a[], &err{, maxI%{, &iTer%{, covar[][]}}});
```

a[] An array of size at least nCoef% that is returned holding the current set of coefficient values. The first amplitude is in a[0], the first centre in a[1], the first sigma in a[2], the second amplitude in a[3] and so on.

err A real variable returned as the sum over the data points of $(y_x[i]-y[i])^2/s[i]^2$ if s[] is used or holding the sum of $(y_x[i]-y[i])^2$ if s[] is not used, where $y_x[i]$ is the value predicted from the coefficients at the x value x[i].

maxI% This is the maximum number of fitting iterations to do before returning from the function. If you omit this value, the function sets 1000. You can set any value from 1 to 10000. If you set more than 10000, the number is limited to 10000.

iTer% An optional integer variable that is returned holding the number of iterations done before the function returned.

covar An optional two dimensional array of size at least [nCoef%][nCoef%] that is returned holding the covariance matrix when the fit is complete. It is not changed unless the function return value is 0 or -1.

Returns 1 if the number of iterations are exhausted and the fit has not converged, 0 if the fit has converged, -1 if the fit is not improving but the result may be OK. Other negative numbers indicate failure.

Remember that even when a minimum is found, there is no guarantee that this is the minimum. It is the best minimum that this algorithm can find given the starting point.

Select coefficients to fit

Normally the command fits all the coefficients, but you can use this variant to select the coefficients to fit. You can use this command at any time after you have set the problem until the iteration variant returns 0 or -1.

```
func FitSigmoid(const fit%[]);
```

fit%[] An array of at least nCoef% integers. If fit%[i] is 0, coefficient i is held constant, otherwise it is fitted. If all elements are 0, all arguments are fitted.

The effect of this command persists until either the iteration variant returns a value less than 1, or you set up a new problem, or you call this variant again.

An example

This is a template for using these commands to fit all the coefficients:

```
const nData%:=50;               'set number of data elements
var x[nData%], y[nData%];       'space for our arrays
var s[nData%];                  'space for sigma of each point
var coefs[4], err;              'coefficients and error squared
...                             'in here goes code to get the data
FitSigmoid(4, y[], x[], s[]);   'fit one gaussian
FitSigmoid(0, 1.0, 0.2, 4);     'set base level and limit range
FitSigmoid(1, 20, 15, 25);      'set end level and range
FitSigmoid(2, 8);               'initial 50% point in X units
FitSigmoid(3, 0.5);             'initial slope
repeat
   DrawMyData(coefs[], x[], y[]);  'Some function to show progress
until FitSigmoid(coefs[], err, 1) < 1;
DrawMyData(coefs[], x[], y[]);  'Show the final state
```

See also:

More about curve fitting, `ChanFit()`, `FitExp()`, `FitLinear()`, `FitGauss()`, `FitNLUser()`, `FitPoly()`, `FitSin()`

FitSin()

This command fits multiple sinusoids to arrays of x,y data points, with an optional weight for each point. Fitting is by an iterative method. The input data is fitted to the equation:

$y = \mathbf{a}_0 \sin(\mathbf{a}_1 x + \mathbf{a}_2) + \mathbf{a}_3 \sin(\mathbf{a}_4 x + \mathbf{a}_5) + \ldots \{+ \mathbf{a}_{3n}\}$

The fitted coefficients are the $\mathbf{a}_i$. Angles are evaluated in radians. You can fit up to 3 sinusoids and an optional offset. Although the function is given in terms of sine functions, you can easily convert to cosines by subtracting $\pi/2$ from the phase angle ($\mathbf{a}_2$, $\mathbf{a}_5$, $\mathbf{a}_8$) after the fit. The commands to implement this are:

Set up the problem

The first command sets the number of sinusoids to fit and the data set to be fitted. You must call this function before you call any of the others.

```
func FitSin(nCoef%, const y[], const x[]{, const s[]|s});
```

`nCoef%` The number of coefficients to fit. The only legal values are 3, 6 and 9 for one, two and three sinusoids or 4, 7 and 10 to include an offset as the last coefficient.

`y[]` An array of y data values. The length of the array must be at least `nCoef%`.

`x[]` An array of x data values. The length of the array must be at least `nCoef%`.

`s` An optional argument which is either an array with one value for each y data point or a single value for all data points. If the value is the standard deviation, then the error value returned when you iterate to find the best fit is the chi-squared value and the fit is a chi-squared fit.

If this value is proportional to the error in the y values, then the fit is still a chi-squared fit, and the error returned is proportional to the chi-squared value. If you omit this array, the fit is a least-squares error fit, and the error value returned is the sum of the squares of the errors in the y values.

Returns The function returns 0. There is no other return value as all errors stop the script.

The number of data points is set by the smallest of the sizes of the `y[]`, `x[]` or `s[]` (if present) arrays. The number of data points must be at least the number of coefficients. If it is not, you will get a fatal error, so check this before calling the function.

Set coefficient values and ranges

This variant of the function sets the initial value of each coefficient and optionally sets the range of allowed values. You can call this function at any time after the set up call and before the iterate call (below) has returned 0, indicating that the fit is complete.

In this type of fitting it is very important that you give reasonable starting values for the coefficients. In particular, when fitting multiple sinusoids you will usually either know, or have a good idea of the frequencies. You should limit the range of each frequency so that they cannot overlap. If you can do this, the fit will proceed quickly. If you do not give starting values, the command will make a simplistic guess at the fitting values. As we expect that you know more about the "right" answer than the command does, we suggest that you set the values you want.

```
func FitSin(coef%, value{, lo, hi});
```

`coef%` The coefficient to set. The first coefficient is number 0, the last is `nCoef%`-1.

`value` The initial value to assign to the coefficient. If you have set low and high limits, and the value is outside these limits, it is set to the nearer limit.

`lo,hi` If present, these two values set the acceptable range of values for this coefficient. If omitted, or if both values are set to the same value, there is no limit. The value of the coefficient is tested against these limits after each iterative step, and if it exceeds a limit, it is set to that limit.

Iterate to a solution

Once you have set up the problem and given initial values to your coefficients, you can start the iteration process that will move the coefficients from their starting values to new values that minimise the error (optionally scaled by the `s` argument).

```
func FitSin(a[], &err{, maxI%{, &iTer%{, covar[][]}}});
```

`a[]` An array of size at least `nCoef%` that is returned holding the current set of coefficient values. The first amplitude is in `a[0]`, the first frequency in `a[1]`, the first phase angle in `a[2]`, the second amplitude in `a[3]` and so on.

`err` A real variable returned as the sum over the data points of $(y_x[i]-y[i])^2/s[i]^2$ if `s[]` is used or holding the sum of $(y_x[i]-y[i])^2$ if `s[]` is not used, where $y_x[i]$ is the value predicted from the coefficients at the x value `x[i]`.

`maxI%` This is the maximum number of fitting iterations to do before returning from the function. If you omit this value, the function sets 1000. You can set any value from 1 to 10000. If you set more than 10000, the number is limited to 10000.

`iTer%` An optional integer variable that is returned holding the number of iterations done before the function returned.

`covar` An optional two dimensional array of size at least `[nCoef%][nCoef%]` that is returned holding the covariance matrix when the fit is complete. It is not changed unless the function return value is 0 or -1.

Returns 1 if the number of iterations are exhausted and the fit has not converged, 0 if the fit has converged, -1 if the fit is not improving but the result may be OK. Other negative numbers indicate failure.

Even when a minimum is found, there is no guarantee that this is the minimum, only that it is the best minimum that this algorithm can find given the starting point.

Select coefficients to fit

Sometimes you may wish to hold some coefficients fixed while you fit others. Normally the command will fit all the coefficients, but you can use this command variant to select the coefficients to fit. You can use this command at any time after you have set the problem until the iteration variant returns 0 or -1.

```
func FitSin(const fit%[]);
```

`fit%[]` An array of at least `nCoef%` integers. If `fit%[i]` is 0, coefficient `i` is held constant, otherwise it is fitted. If all elements are 0, all arguments are fitted.

The effect of this command persists until either the iteration variant returns a value less than 1, or you set up a new problem, or you call this variant again. For a sinusoidal fit it is likely that you will know the frequency to fit, so you may well hold this constant.

An example

The following is a template for using this command (assuming you don't want to fit the frequency, which we assume you know).

```
const nData%:=50;         'set number of data elements
var x[nData%], y[nData%]; 'space for our arrays
var s[nData%];            'space for sigma of each point
var fit%[3];              'we want to hold the frequency
var coefs[4];             'space for coefficients
var err;                  'will hold error squared
...                       'in here goes code to get the data
FitSin(3, y[], x[], s[]); 'fit one sinusoid
'Note that we let the phase take any value
FitSin(0, 1.0, 0.2, 4);   'set amplitude and limit range
FitSin(1, .02, .01, .03); 'set frequency
FitSin(2, 0., 0.3, 1.9);  'set width and limit range
'Now we say that we don't want to fit the frequency
ArrConst(fit%[],1);       'set all elements to 1
fit%[1] := 0;             'but not element 1 (=frequency)
FitSin(fit%[]);           'so the frequency is fixed
repeat
   DrawMyData(coefs[], x[], y[]);  'Some function to show progress
until FitSin(coefs[], err, 1) < 1;
DrawMyData(coefs[], x[], y[]);     'Show the final state
```

See also:

More about curve fitting, `FitExp()`, `FitGauss()`, `FitLinear()`, `FitNLUser()`, `FitPoly()`, `FitSigmoid()`

FitValue()

This function returns the value at a particular x position of the fitted function set by the last `FitData()` command.

```
Func FitValue(x{, &valid%});
```

x The x value at which to evaluate the current fit. You should be aware that some of the fitting functions can overflow the floating point range if you ask for x values beyond the fitted range of the function.

`valid%` If present, this integer variable is set to 1 if the returned value is valid, 0 if not.

Returns The value of the fitted function at x. If the result is out of floating point range, the function may return a floating point infinity or a NaN (Not a Number) value or a 0. If there is no fit, the result is always 0.

See also:

`FitCoef()`, `FitData()`, `FitExp()`, `ChanFitValue()`, More about curve fitting

Floor()

Returns the next lower integral number of the real number or array. `Floor(4.7)` is `4.0`, `Floor(4)` is `4`. `Floor(-4.7)` is `-5`.

```
Func Floor(x|x[]{[]...});
```

`x` A real number or a real array.

Returns 0 or a negative error code for an array or the next lower integral value.

See also:

`Abs()`, `ATan()`, `Ceil()`, `Cos()`, `Exp()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`

FocusHandle()

This function returns the view handle of the script-controllable window with the keyboard input focus (the active window). Unlike `FrontView()`, it can return any type of window, for example multimedia, cursor regions and spike shape windows.

```
Func FocusHandle({noRst%});
```

`noRst%` Optional. If present and set to -1 and used in a function linked to a Toolbar button, this prevents the keyboard focus being restored when the function returns control back to Spike2. See below for details. This was added at version [10.14].

Returns The handle of a window that the script can manipulate, or 0 if the focus is not in such a window.

Keyboard focus interaction with the Toolbar

When you type characters on the keyboard, the operating system must decide where these characters are sent. Normally, they go to the window with the keyboard focus. If this is a text window it will usually have a blinking caret. When you are sampling data with Spike2, you usually want the sampling data window to have the input focus so that keystrokes are recorded and can control sampling actions (output sequencer, arbitrary waveform output).

When you click on a button in Windows, the standard action is for the input focus to transfer to the button, or to the nearest window that owns the button that can accept the input focus. If you have the Toolbar or Interact bar displayed when sampling, a click on any button in the bar would move the focus. As this is unlikely to be what you wanted during Sampling (and possibly in other situations), we have added code to these bars that restores the focus to where it was previously.

Due to the way that Windows works, it turns out that when you link a function to a toolbar button (a *call-back* function), the restoration of focus happens when the call-back functions returns. This means that if you set the active view within the function, the input focus will be redirected somewhere else when the function returns. You can prevent this by a call to `FocusHandle(-1)` within the call-back function as illustrated in this script:

```
View(App(3)).WindowVisible(0); 'Hide running script

var tv% := FileNew(1, 1);    'Make a new text view
Window(0,0, 100, 60);        'Position it
Print("New Text View");      'Identify it

var lv% := LogHandle();      'Log view
View(lv%).WindowVisible(1); 'Ensure visible
View(lv%).Window(0,60,100,100); 'Position it
PrintLog("Log view");        'Identify it

ToolbarSet(1, "Quit");
ToolbarSet(3, "Text", TextView%);
ToolbarSet(4, "Log", LogView%);
ToolBar("Buttons set the focus", 1023);
View(tv%);
FileClose();                 'Close the text view

Func TextView%()             'Button 3 callback
FrontView(tv%);              'Make front (give focus)
FocusHandle(-1);             'Prevent focus restore
return 1;                    'Keep toolbar active
end;

Func LogView%()              'Button 4 callback
FrontView(lv%);              'Make front (give focus)
FocusHandle(-1);             'Prevent focus restore
return 1;                    'Keep toolbar active
end;
```

This script hides the running script, then creates and positions a text view and positions the Log view, creates a tool bar with buttons `"Text"`, `"Log"` and `"Quit"`, and then waits for the user to click buttons or type in the views.

Clicking the "Text" or "Log" buttons makes the view active and gives it the keyboard focus. If you comment out the **FocusHandle(-1)** lines, clicking on the buttons will make that window the font view, but the keyboard focus (the view with the caret) stays where it is. In fact, the focus does move, but it is restored when the call-back function ends.

See also:
FrontView()

FontGet()

This function gets the name of the font, and its characteristics for the current view.

```
Func FontGet(&name$, &size, &flags%{, style%{,&fore%{,&back%}}});
```

name$ This string variable is returned holding the name of the font.

size The real number variable is returned holding the point size of the font.

flags% Returned holding the sum of the style values: 1=Italic, 2=Bold, 4=Underline, 8=Force upper case, 16=force lower case. Only one of 8 or 16 will be returned.

style% Text-based views have all have default style (32) that is used as the basis of all other styles, plus a number of additional styles that are used to highlight keywords and the like in output sequencer and script windows. The style codes for other styles start at 0 and run upwards. You can find a list of styles for each view type in the Edit Setup dialog and they are tabulated in the description for script, output sequence and other text files. If you omit style% the settings for the default style (32) are returned.

The back% colour for the default style (32) also sets the background colour of the text view.

All other view types ignore style% (which should be set to 0, if supplied) except time views, which use style 0 for the normal view style and style 1 for the font for vertical markers.

fore% This value returns the foreground colour of the style. Bits 0-7 hold the red intensity, bits 8-15 hold the green and bits 16-23 hold the blue.

back% This value returns the background colour of the style.

Returns The function returns 0 if all was well or a negative error code. If an error occurs, the variables are not changed.

The arguments from style% onwards and the flag% values 4, 8 and 16 only apply to text-based views and were added in version 6.

See also:
FontSet(), TabSettings()

FontSet()

This function sets the font for the current view. Text-based views (text, sequencer and script) normally avoid proportionally spaced fonts as they did not display correctly before Spike2 version 6. Arguments from style% onwards are for text-based views.

```
Func FontSet(name$|code%, size, flags%{, style%{, fore%{, back%}}});
```

name$ A string holding the font name to use or you can specify a font code:

code% This is an alternative method of specifying a font. We recognise these codes:

0	The standard system font, whatever that might be
1	A non-proportionally spaced font, usually Courier-like
2	A proportionally spaced non-seriffed font, such as Helvetica or Arial
3	A proportionally spaced seriffed font, such as Times Roman
4	A symbol font
5	A decorative font, such as Zapf-Dingbats or TrueType Wingdings

size The point size required. Your system may limit the allowed range.

`flags%` The sum of the style values to set: 1=Italic, 2=Bold, 4=Underline, 8=Force upper case, 16=force lower case. If both 8 and 16 are set, 16 is ignored. Values from 4 upwards are only supported by text-based views.

`style%` Text-based views have a default style (32) plus a number of additional styles that are used to highlight keywords in output sequencer and script windows. The style codes for other styles start at 0 and run upwards. You can find a list of styles for each view type in the Edit Setup dialog and they are tabulated in the description for script, output sequence and other text files. If you omit `style%`, the default style is changed. Set the value -1 to set all styles to the values you give.

All other views ignore `style%` (which should be set to 0, if supplied) except time views, which use style 0 for the normal view style and style 1 for the font for vertical markers.

`fore%` This sets the style foreground colour or is set to -1 to make no change. Bits 0-7 hold the red intensity, bits 8-15 hold the green and bits 16-23 hold the blue. Bits 24-31 are 0. It is convenient to code this as a hexadecimal number, for example:

```
const red%:=0x0000ff, green%:=0x00ff00, blue%:=0xff0000;
const gray%:=0x808080, yellow% :=0x00ffff;
```

`back%` This sets the background colour of the style in the same format as `fore%`.

Returns The function returns 0 if the font change succeeded, or a negative error code.

See also:
Edit setup dialog, `FontGet(), TabSettings()`

Frac()

Returns the fractional part of a real number or converts a real array to its fractional parts.

```
Func Frac(x|x[]{[]...});
```

`x` A real number or an array of real numbers.

Returns For arrays, it returns 0 or a negative error code. If `x` is not an array it returns a real number equal to `x-Trunc(x)`. `Frac(4.7)` is 0.7, `Frac(-4.7)` is -0.7.

See also:
`Abs(), ATan(), Clamp(), Cos(), Exp(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan(), Trunc()`

FrontView()

The front (or active) view is the Time, Result, XY, Grid or text-based view that decides which menu appears in the main window. You can use this command to get the active view (view with the input focus), or to set it, or to move any other script-controlled window to the front (to make it visible). When a view becomes the front view, it is moved on top of other views in Spike2 and made the current view. If an invisible or iconised view is made the front view, it is made visible automatically, (equivalent to `WindowVisible(1)`).

You can also use this command to move other windows (for example the multimedia, cursor and spike shape windows) to the front and make them the current view, but such windows are not returned by this command. Use `FocusHandle()` to return these windows.

```
Func FrontView({vh%});
```

`vh%` Either 0 or omitted to return the front view handle, a view handle to be set, or -n, meaning the n^{th} duplicate of the time view associated with the current view. If `vh%` is the view handle of a Time, Result, XY or text-based view, this makes it the Active view. If `vh%` is the handle of any other view, that view is moved to the front and made the current view, but the active view is not changed.

Returns 0 if there are no visible views, -1 if the view handle passed is not a valid handle, otherwise it returns the view handle of the Active Time, Result, XY, Grid or text-based view at the time of the call.

See also:
FocusHandle(), View(), WindowVisible()

G

GammaP() and GammaQ()
Distributions based on GammaP()
GammaQ()
GrdAlign()
GrdColWidth()
GrdGet()
GrdSet()
GrdShow()
GrdSize()
Grid()
Gutter()

GammaP() and GammaQ()

These functions return the incomplete gamma function *P(a, x)*. It is defined as:

$$P(a,x) = \frac{\int_0^x e^{-t}t^{a-1}dt}{\Gamma(a)} \quad \text{where} \quad \Gamma(a) = \int_0^\infty t^{a-1}e^{-t}dt$$

This is named GammaP() in the script. We also define the complement of this function, GammaQ(a,x), which is 1.0-GammaP(a,x). In some literature, these are referred to as the Regularised Gamma functions.

```
Func GammaP(a, x);
Func GammaQ(a, x);
```

a This must be positive, it is a fatal error if it is not.

x This must be positive, it is a fatal error if it is not.

Returns These functions return the incomplete Gamma function and the complement of the incomplete Gamma function.

These functions are not of that much direct interest. However, from them are obtained the error function, the cumulative Poisson probability function and the Chi-squared probability function.

See also:
BetaI(), LnGamma(), Distributions based on GammaP()

Distributions based on GammaP()

Error function

The error function *erf(x)* = GammaP(0.5, x*x)

This is defined as:

$$erf(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}$$

which sounds uninteresting until you realise that this is the integral of a Normal distribution. As defined, the result is always positive. If you want a version that gives a negative result for negative x, use:

```
Func Erf(x)
var r;
r := GammaP(0.5, x*x);
return x >= 0 ? r : -r;
end;
```

The functions use becomes more obvious when the *x* value is expressed in terms of the standard deviation of a Normal distribution. For example, if you want to know what proportion of normally distributed data lies within `n` standard deviations of the mean, you can use:

```
Func NormProp(n)
return GammaP(0.5, n*n*0.5);
end;
```

Multiply the result by 100 to get a percentage. The proportion of data that lies outside `n` standard deviations of the mean is:

```
Func NormPropComp(n)
return GammaQ(0.5, n*n*0.5);
end;
```

Normal distribution cumulative distribution function

To get the proportion of normally distributed data that lies below `x` (from minus infinity to `x`) given a mean and a standard deviation, you can use:

```
'x       The value to test
'mean    The mean value of the Normal distribution
'sd      The standard deviation of the distribution
'Returns The proportion of data that is less than or equal to x
Func NormDistCDF(x, mean, sd)
x -= mean;
x /= sd;
return (x>=0) ? 0.5*(1 + GammaP(0.5, x*x*0.5)) : 0.5*GammaQ(0.5, x*x*0.5);
end;
```

The proportion above `x` (from `x` to plus infinity) is given by the complement of `NormDistCDF()`:

```
Func NormDistCDFComp(x, mean, sd)
x -= mean;
x /= sd;
return (x>=0) ? 0.5*GammaQ(0.5, x*x*0.5) : 0.5*(1 + GammaP(0.5, x*x*0.5));
end;
```

Do not be tempted by:

```
1.0 - NormDistCDF(x, mean, sd);
```

as this severely limits the accuracy when the result is small (that is when `NormDistCDF()` is close to 1.0).

You can also perform the inverse of these function (that is, given a proportion, what is the value) with `ZeroFind()`. If you are attempting to find values that are very close to 1, you would do much better to use the complement function and search for values that are very close to 0. That is, to find the position where `NormDistCDF()` is 0.99999, you should find the position where `NormDistCDFComp()` is 0.00001 as this will give a much more accurate result.

Cumulative Poisson probability function

The cumulative Poisson probability function relates to a Poisson process of random events and is the probability that, given an expected number of events `r` in a given time period, the actual number was greater than or equal to `n`. This turns out to be `GammaP(n,r)`. Also, the probability that there are less than `n` events is `GammaQ(n,r)`.

Chi-squared probability function

The Chi-squared probability function is useful where we are fitting a model to data. Given a fitting function that fits the data with `n` degrees of freedom (if you have `nData` data points and `nCoef` coefficients you usually have `nData-nCoef` degrees of freedom), and given that the errors in the data points are normally distributed, the probability of a Chi-squared value less than `chisq` is `GammaP(n/2, chisq/2)`. Similarly, the probability of a `chisq` value at least as large as `chisq` is `GammaQ(n/2, chisq/2)`. So, if you know the chi-squared value from a fitting exercise, you can ask "What is the probability of getting this value (or a greater one) given that my model fits the data?" If the probability is very small, it is likely that your model does not fit the data, or your fit has not converged to the correct solution.

See also:
BetaI(),LnGamma(),Binomial Distribution, Student's T Distribution, F-Distribution

GammaQ()

The complement of GammaP(); GammaQ(a,x) is 1.0-GammaP(a,x).

```
Func GammaQ(a, x);
```

a This must be positive, it is a fatal error if it is not.

x This must be positive, it is a fatal error if it is not.

Returns The complement of the incomplete Gamma function.

See also:
GammaP()

Grd...()

GrdAlign()
GrdColourGet()
GrdColourSet()
GrdColWidth()
GrdGet()
GrdSet()
GrdShow()
GrdSize()

GrdAlign()

This command sets the alignment of grid columns.

```
Func GrdAlign(align%, col%)
```

align% Set -1 to make no change, 0 for left aligned, 1 for centred and 2 for right aligned columns.

col% The column number to align or -1 to align all columns that intersect the current selection or -2 for all columns (-2 was added at version 8.10).

Returns The alignment of the lowest numbered column defined by col% as 0 for left, 1 for centred and 2 for right aligned before any change made by this command is applied.

See also:
GrdGet(), GrdSet(), GrdColWidth(), GrdShow(), GrdSize()

GrdColourGet()

You can control the colour of cell backgrounds and cell text. The Grid colours cells by cells and by column. This means that if a cell has a colour set, it is used, otherwise the column colour is used. Unless you explicitly set them, the column colours are set to the View colours, if these are defined, otherwise to the application colours. The grid also has a Default colour, but this is used only when creating new columns. The Default colour is the View colour if set, otherwise it is the application colour.

This command allows you to read back the current state of a particular cell or column or the Default. This command was added at version [10.05].

```
Func GrdColourGet(item%, col%, row%{, &r, &g, &b});
```

item% Identifies the item: 0= background, 1=text (foreground).

`col%` The 0-based column number or -1 to get information for the grid default.

`row%` The 0-based row number, or -1 to return the column state. If `col%` is -1, `row%` is ignored.

`r g b` If present, returned as the red, green and blue colour values for the item chosen by `col%` and `row%` in the range 0.0 to 1.0. If the colour is not set and `col%` and `row%` are both positive, the returned colour is the display colour of the cell, otherwise the returned colour is meaningless and likely to be (0, 0, 0).

Return 1 if colour is set, 0 if not, -1 if the column or row do not exist. A set colour is one that has been applied using `GrdColourSet()` or the interactive equivalent to set the colour of selected cells. Any cell that has no set cell colour or column colour displays in the view colour, or failing that, in the application colour.

See also:
`GrdGet()`, `GrdSet()`, `GrdColourSet()`, `GrdColWidth()`, `GrdShow()`, `GrdSize()`, `ViewColourGet()`, `ColourGet()`

GrdColourSet()

Grids use the background and text colour set for the grid view, as described for `ViewColourSet()`. You can override these colours by column or by cell both interactively from the Grid context menu and with this script command. If you change the colour of the background, you should be aware that the borders around each cell are drawn in a light grey (`r`, `g`, `b`) = (0.86, 0.86, 0.86), so you should attempt to preserve the contrast.

If a cell has a colour set, it is used. Otherwise the colour set for the column that the cell is part of is used. If you change the View colours, this is applied by setting the colour of all the columns. If you want to set column colours, apply your changes after setting View colours. Cell colours can be though of as having a 4-layer inheritance: cell, column, view, application. They take the colour of the nearest set colour in the hierarchy.

You can colour the current cell selection, a particular cell or column or the grid default. You can also clear colours so that they revert to the column, view or application colour. The grid default is used only when creating new columns; if not set explicitly it is the View colour and failing that, the Application colour. This script command was added at version [10.05].

There are two command variants:

Set or clear a rectangular range of cells

This sets or clears the colour value of an item for a range of cells.

```
Func GrdColourSet(item%, col%, row%, cEnd%, rEnd%{, r, g, b});
```

`item%` Identifies the item: 0= background, 1=text.

`col%` If `col%` is negative, `cEnd%` and `rEnd%` are ignored and the command behaves as the other variant (this use is deprecated). Otherwise this is the 0-based left-hand column of the cell or column range.

`row%` The 0-based top row of the cell range or -1 to set the colour for the column range defined by `col%` and `cEnd%`.

`cEnd%` The right-hand column of the cell range. This must be greater than or equal to `col%`.

`rEnd%` The bottom row of the range to set or clear. This must be greater than or equal to `row%`.

`r g b` If present, sets the red, green and blue values in the range 0 (no colour) to 1.0 (maximum colour). Values outside this range are limited to the range 0.0 to 1.0. If omitted, this clears any cell colour set; the cell uses the column colour.

Return 0 for success, -1 if you use an illegal combination of arguments.

Set a single cell, single column default or grid default

Clearing a column default or the grid default sets it to the view colour, and if that is not set, to the application colour.

```
Func GrdColourSet(item%, col%{, row%{, r, g, b}});
```

`item%` Identifies the item: 0= background, 1=text.

col% The 0-based column number of the cell or column or -1 to set the grid Default or set the colour of the current cell selection.

row% The 0-based row of the cell range or -1 to set the colour of the column identified by col%. If col% is -1, set -1 to set the grid default and all the column defaults or 0 to set the currently selected cells.

r g b If present, sets the red, green and blue values in the range 0 (no colour) to 1.0 (maximum colour). Values outside this range are limited to the range 0.0 to 1.0. If omitted, this clears any cell colour set; the cell uses the column colour. You may not omit these arguments if you are targeting the column or grid defaults. Clearing the grid or column defaults sets them to the current colour set for the view.

Return 0 for success, -1 if you use an illegal combination of arguments.

See also:
GrdGet(), GrdSet(), GrdColourGet(), GrdColWidth(), GrdShow(), GrdSize(), ViewColourSet()

GrdColWidth()

This command gets and optionally sets the width (in pixels) of a nominated column, or all columns that intersect with the current selection. It returns the width of the item. It can also get the width of the left-hand size column holding row titles and the right-hand vertical scroll bar. There are two variants:

```
Func GrdColWidth(col%{, width%});
Func GrdColWidth(col%, text$);
```

col% The column to use or:

- -1 To set all columns that intersect the currently selected cells in the grid
- -2 To set all columns. If you omit width% it returns the width of the first column.
- -3 Returns the width of the view area available for columns of data (between the row numbers and the vertical scroll bar).
- -4 Returns the width of the row number column on the far left (or 0 if the row numbers are not displayed).
- -5 Returns the width of the vertical scroll bar (or 0 if the vertical scroll bar is not displayed).

width% Optional and ignored if col% is -3 or less. If present, and positive it sets the width of the columns nominated by col% in pixels. You are allowed to set a 0 width, which hides the column(s). You can also set

- -1 Set the default column widths.
- -2 Optimise the column widths based on the contents.

text$ Set the column wide enough to display this text, based on the current font set for the body of the grid (not the headings).

Returns The width of the first column set by col%, in pixels or -1 if col% is too large to refer to a column. If you set the width, the returned value is the new width.

Example

This code example sets the column widths of the current grid view to exactly fill the grid view window. The column widths are resized in proportion to their original sizes.

```
if ViewKind() <> 17 then Message("Current view not a grid!"); halt endif;

Draw(0);                              'place first column on left
var nCol% := GrdSize();               'get the number of columns
var c%, colW%[nCol%];                 'space for original column widths
for c% := 0 to nCol%-1 do
    colW%[c%] := GrdColWidth(c%);     'Save original widths
next;

var used% := ArrSum(colW%);                  'space currently used
var spaceLeft% := GrdColWidth(-3) - used%;   'can be more or less than 0

'Now allocate extra space based on remaining space and initial allocation.
```

```
var delta%, width%;
for c% := 0 to nCol%-1 do
    width% := colW%[c%];
    delta% := round((spaceLeft% * width% * 1.0) / used%);
    spaceLeft% -= delta%;               ' remaining space to allocate
    used% -= width%;                    ' original width remaining
    GrdColWidth(c%, width% + delta%);   ' set new width
next;
```

See also:
GrdGet(), GrdSet(), GrdAlign(), GrdShow(), GrdSize()

GrdGet()

Collect data from the current grid view. You can read back data from a single cell, or from a column or a row, or from a rectangular region of the grid.

```
Func GrdGet(&text$|text$[]{[]}, col%, row%);
```

`text$` This is either a simple string variable or a string vector or a string matrix. With a simple variable, you read back one cell from `row%`, `col%`. With a vector, you read back a column of data starting at `row%`, `col%`. With a matrix, you read back a rectangular grid of items starting at `row%`, `col%`. To read back a row of data into a vector, use the transpose operator on the vector (`trans(text$[])` or `text$``).

`col%` This sets the start column. The first data column is 0.

`row%` This sets the start row. You can read back the column headers by setting this to -1. The first data column is 0.

Returns The number of elements of `text$` that were set.

If either `row%` or `col%` exceed the size of the grid, nothing is read and 0 is returned.

See also:
GrdSet(), GrdColWidth(), GrdAlign(), GrdShow(), GrdSize()

GrdSet()

Set data in the current grid view. You can write data to a single cell, or to a column or a row or to a square region of the grid.

```
Func GrdSet(text$|const text$[]{[]}, col%, row%);
```

`text$` This is either a simple string or a string vector or a string matrix. With a simple variable, you set one cell at `row%`, `col%`. With a vector, you set a column of data starting at `row%`, `col%`. With a matrix, you set a rectangular grid of items starting at `row%`, `col%`. To set a row of data from a vector `text$[]`, use the transpose operator on the vector (`trans(text$)` or `text$``).

`col%` This sets the start column. The first data column is 0.

`row%` This sets the start row. You can set the column headers by setting this to -1. The first data column is 0. If you set a blank column header, the displayed heading reverts to the default.

Returns The number of cells of the grid that were set.

If either `row%` or `col%` exceed the size of the grid, nothing is set and 0 is returned.

See also:
GrdGet(), GrdColWidth(), GrdAlign(), GrdShow(), GrdSize()

GrdShow()

Show, hide and report visibility of column and row headers. To hide a column use `GrdColWidth()` and set the column width to 0.

```
Func GrdShow({cHead%, rHead%});
```

`cHead%` Omit or set -1 for no change, 0 to hide and 1 to show the column header.

`rHead%` Omit or set -1 for no change, 0 to hide and 1 to show the row header.

Return The original state as the sum of 1 (for column headers visible) and 2 (for row headers visible), so possible return values are 0, 1, 2 or 3.

See also:
`GrdGet(), GrdSet(), GrdColWidth(), GrdAlign(), GrdSize()`

GrdSize()

This function resizes the grid or reports the grid size. If you change the size, existing cell contents in surviving cells are preserved. There are two command variants.

Change grid size

```
Func GrdSize(cols%, rows%);
```

`cols%` The number of columns to set in the grid or -1 for no change.

`rows%` The number of rows to set in the grid or -1 for no change.

Returns 0 if done or a negative error code (for example, too many cells).

Get grid size

The second variant returns the number of rows or columns:

```
Func GrdSize({get%});
```

`get%` Set 0 or omit to return the number of columns. Set -1 to return the number of rows.

Returns The information requested by `get%`.

See also:
`GrdGet(), GrdSet(), GrdColWidth(), GrdAlign(), GrdShow(), WindowSize()`

Grid()

This function turns the background grid on and off for the current time, result or XY view and returns the state of the grid. The grid is a mesh of lines that follow the big and small ticks on the x and y axes that is drawn on top of the data view background and under the data. Separate control of x and y grid lines was added at version [7.02]. The ability to display axes in the grid area was added at version `[10.07]`.

```
Func Grid({on%});
```

`on%` Optional, sets the grid state. Omit or set -1 for no change, 0 = no grid, 1 = both x and y grid, 2=x grid only, 3=y grid only. From [10.07] add 4 for the x axis in the data area. Add 8 for the y axis in the grid area.

Returns The state of the grid at the time of the call as 0-15 or a negative error code. Changes made by this function mark the grid area as invalid; they do not cause an immediate redraw.

See also:
`XAxis(), XScroller(), YAxis()`

Gutter()

The gutter is the area on the left of a text-based window where bookmarks and script break points appear. This function returns and optionally sets the gutter visible state. If you set a large font size you may wish to hide the gutter.

```
Func Gutter({show%});
```

`show%` Optional, sets the gutter state. 0 = hide, 1 = show, -1 or omitted for no change.

Returns The gutter state at the time of the call: 0 = hidden, 1=visible.

See also:
`ViewLineNumbers()`

H

`HCursor()`
`HCursorX()`
`HCursorActive()`
`HCursorChan()`
`HCursorDelete()`
`HCursorExists()`
`HCursorLabel()`
`HCursorLabelPos()`
`HCursorNew()`
`HCursorRenumber()`
`HCursorValid()`
`HCursorVisible()`
`Help()`

HCursor()

This function returns the position of a horizontal cursor, and optionally sets a new position. You can get and set positions of cursors attached to invisible channels or channels that have no y axis. It is also used with spike shape windows that have horizontal cursors to get and set trigger levels and in an offline Edit WaveMark window to read back the last set values. See `HCursorX()` to get the previous position.

```
Func HCursor(num% {,where {,chan%}});
```

`num%` The cursor to use. It is an error to attempt this operation on an unknown cursor.

In a Spike Shape window, cursor 1 is the low trigger level, 2 is the high trigger, 3 is the low limit (if enabled), 4 is the high limit (if enabled). To set the cursor for trace `n` (0-3) add 4*`n` to the cursor number.

`where` If this parameter is given it sets the new position of the cursor.

In a Spike Shape window, the position must make sense for the cursor; horizontal cursors are expected to be in the order 4, 2, 1, 3 (from top to bottom) and cursors 4 and 2 are limited to the upper part of the display and 1 and 3 to the lower part.

`chan%` If this parameter is given, it sets the channel number. In XY or spike shape views you should omit this argument as it is ignored.

Returns The function returns the position of the cursor at the time of the call, before any change made by the `where` argument.

Use in a Spike shape view

You can use code like the following to set a horizontal cursor level in a spike shape window.

```
var sv%;
sv% := SSOpen(0);  'get handle of open spike shape dialog
if sv% > 0 then View(sv%).HCursor(2,1.3) endif;
```

You can use `HCursor()` in an offline Edit WaveMark view to read back (but not to set) the last set cursor levels. These may be the levels used to create the current set of spikes. However, the user was perfectly at liberty to change the levels after the events were created, but before the spike shape window closed, so you cannot rely on these levels.

The order of the cursors is strictly enforced and the cursors cannot cross the zero line. When you have 4 cursors enabled you must move them bearing in mind that the upper and lower cursor pairs limit each others movement.

See also:
`HCursorChan(), HCursorDelete(), HCursorLabel(), HCursorLabelPos(), HCursorNew(), HCursorRenumber()`

HCursorActive()

This function sets and retrieves the parameters used by an active horizontal cursor. The setting functions are equivalent to the Horizontal cursor active mode dialog to which you are referred for details of the active horizontal cursor modes.

Set active cursor mode

This form of the function sets the active horizontal cursor mode and parameters. All arguments except the cursor number are optional. However, the command starts by reading the current active state for the nominated cursor and any required value that you do not supply will inherit the current value, which may not be the value you intended.

```
Func HCursorActive(num%{, mode% {,start|start$|expr$, end|end$ {,factor
{,seq%}}}});
```

`num%` The horizontal cursor number in the range 1 to 10. If only this argument is provided the function returns the current cursor mode and makes no changes.

`mode%` The active cursor mode (see here for a detailed description of the modes). The possible values of `mode%` are:

0	Static	5	Minimum value
1	Value at point	6	Absolute maximum value
2	Expression	7	Mean level + (SD * factor)
3	Mean level	8	Median
4	Maximum value	9	Median + (Size * factor)

`start` The start time for the measurement in seconds, or the time for the Value at point measurement.

`start$` The start time for the measurement as a string. Time expressions such as `"C1+0.2"` can be used for the start time.

`expr$` For Expression mode only, this is a channel expression relating to the channel on which the cursor is placed such as `"HCursor(1)"` or `"Mean(0.1, 0.2)"`.

`end` The end time for the measurement, in seconds.

`end$` The end time for the measurement as a string. Time expressions can be used.

`factor` This is a scale factor used in mode 7.

`seq%` This sets when the measurement should be evaluated and the cursor repositioned relative to the sequence of active vertical cursor repositioning that occurs when vertical cursor 0 is iterated. Set `seq%` to 0 for automatic sequencing, 1 for evaluation before all the vertical cursors and 2 for after all the vertical cursors.

Returns The active cursor mode set at the time that the function was called.

Get active cursor mode

This form of the function is used to read back the active cursor mode and parameters.

```
Func HCursorActive(num%{, item% {, &value$}});
```

`num%` The horizontal cursor number for which to retrieve information, from 1 to 10.

`item%` This specifies the parameter for which to retrieve the current value. If omitted it is treated as -1. Values are:

-1 `mode%` -3 `end$` -5 `factor`
-2 `start` -4 `expr` -6 `seq%`

`value$` This optional string variable will be updated with the parameter value as a string when `item%` is -2, -3 or -4.

Returns The numerical value of the selected value. For items -2, -3 and -4 this value is generated by parsing the associated string so it will reflect the position of any cursors or data values used. The result is 0.0 if the string parsing fails.

See also:
`ChanMeasure(), ChanValue(), HCursor(), HCursorDelete(), HCursorNew(), HCursorValid(), CursorActive()`

HCursorChan()

This function returns the channel number that a particular cursor is attached to.

```
Func HCursorChan(num%);
```

`num%` The horizontal cursor number.

Returns It returns the channel number that the cursor is attached to, or 0 if this cursor is not attached to any channel or if the channel number is out of the allowed range. XY views return 1 as the channel number.

See also:
`HCursor(), HCursorDelete(), HCursorLabel(), HCursorLabelPos(), HCursorNew(), HCursorRenumber()`

HCursorDelete()

This deletes the designated horizontal cursor. Deleting an unknown cursor has no effect.

```
Func HCursorDelete({num%});
```

`num%` The number of the cursor to delete. If this is omitted, the highest numbered cursor is deleted. Set -1 to delete all horizontal cursors.

Returns The number of the deleted cursor or 0 if no cursor was deleted.

See also:
`HCursor(), HCursorChan(), HCursorLabel(), HCursorLabelPos(), HCursorNew(), HCursorRenumber()`

HCursorExists()

Use this function to determine if a horizontal cursor exists.

```
Func HCursorExists(num%);
```

`num%` The cursor number in the range 1-9.

Returns 0 if the cursor does not exist, 1 if it does.

See also:
`HCursorNew(), CursorExists()`

HCursorLabel()

This gets and optionally sets the horizontal cursor label style for the view or for a cursor. You can label cursors with a position and/or the cursor number, or with user-defined text. There are two variants: the first sets styles and labels; the second reads back user-defined labels.

```
Func HCursorLabel({style%{, num%{, form$}}});
Func HCursorLabel(&form$, num%);
```

`style%` Set 0=None, 1=Position, 2=Number, 3=Both, 4=User-defined. Omit for no change. Style 4 is used with a format string. Styles 0-3 set the styles of cursors selected by `num%` and the view style for new cursors if `num%` is -1 or omitted. Style 4 is applied to cursors set by `num%`; it does not set the view style.

`num%` 1-9 to select a single cursor or to return the label string. Omit `num%` or set it less than 1 (use -1 in case we allow a cursor 0 in the future) for all cursors and to get and set the style for new cursors.

`form$` The user-defined label string for style 4. The string has replaceable parameters `%p`, and `%n` for position and number. We also allow `%w.dp` where `w` and `d` are numbers that set the field width and decimal places.

Returns A cursor style before any change as 0-4 if a single cursor is selected, or the view cursor style as 0-3. If `style%` is omitted or not 0-3, the current view cursor style is not changed.

See also:
`HCursor(), HCursorChan(), HCursorDelete(), HCursorLabelPos(), HCursorNew(), HCursorRenumber()`

HCursorLabelPos()

This lets you set and read the position of the cursor label.

```
Func HCursorLabelPos(num% {,pos});
```

`num%` The cursor number. Setting a silly number does nothing and returns -1.

`pos` If present, the command sets the position. The position is a percentage of the distance from the left of the cursor at which to position the value. Out of range values are set to the appropriate limit.

Returns The cursor position before any change was made.

See also:
`HCursor(), HCursorChan(), HCursorDelete(), HCursorLabel(), HCursorNew(), HCursorRenumber()`

HCursorNew()

This function creates a new horizontal cursor in a time or result view and assigns it to a channel. You can create up to 9 horizontal cursors.

```
Func HCursorNew(chan% {,where});
```

`chan%` A channel for the new cursor. If the channel is hidden, the cursor is not visible. You should use a channel number of 1 for XY views.

`where` An optional argument setting the cursor position. If this is omitted, the cursor is placed in the middle of the y axis or at zero if there is no y axis.

Returns It returns the horizontal cursor number or 0 if all cursors are in use.

See also:
`HCursor(), HCursorChan(), HCursorDelete(), HCursorLabel(), HCursorLabelPos(), HCursorRenumber()`

HCursorRenumber()

This command renumbers the cursors from bottom to top. There are no arguments.

```
Func HCursorRenumber();
```

Returns The number of cursors found in the view.

See also:
`HCursor(), HCursorChan(), HCursorDelete(), HCursorLabel(), HCursorLabelPos(), HCursorNew()`

HCursorValid()

Use this function to test if the last measurement for a level of an active horizontal cursor succeeded. Horizontal cursor positions are valid if a measurement succeeds or if the cursor is positioned manually or by a script command. The position of a newly created cursor is valid.

```
Func HCursorValid(num%);
```

`num%` The horizontal cursor number to test for a valid measurement result.

Returns The result is 1 if the position of the nominated horizontal cursor is valid or 0 if it is invalid or the horizontal cursor does not exist.

See also:
`HCursorActive(), HCursorNew(), CursorSearch(), ChanMeasure()`

HCursorVisible()

Horizontal cursors can be hidden without deleting them. A hidden cursor can be manipulated by the script language as if it were visible. Be aware that leaving a cursor hidden can be confusing for a user!

```
Func HCursorVisible(num%{, show%});
```

`num%` The cursor number in the range 1-9 or -1 for all horizontal cursors.

`show%` If present set this to 0 to hide the cursor and non-zero (1 preferred) to show it.

Returns If `num%` is -1, the result is the number of horizontal cursors. Otherwise, the state of the cursor at the time of the call (0=hidden or does not exist, 1=visible).

See also:
`HCursorExists(), HCursorNew(), HCursorValid()`

HCursorX()

This function returns the position of a horizontal cursor in a Time, Result or XY view before the move that took it to the current position. It is particularly useful with Active cursors. The same command is available as a Dialog Expression. This command was added at Spike2 version [8.09].

```
Func HCursorX(num%);
```

`num%` The cursor to use. It is an error to attempt this operation on an unknown cursor.

Returns The function returns the position of the cursor before the move (however caused) to the current position. Use `HCursor(num%)` to get the current position.

See also:
`HCursor(), HCursorChan(), HCursorDelete(), HCursorLabel(), HCursorLabelPos(), HCursorNew(), HCursorRenumber()`

Help()

This command lets you display help information from the standard Spike2 help file or from a help file of your choosing. You can also use `Help(0)` to close any open help file.

```
Func Help(topic%|topic$ {,file$});
```

`topic%` A numeric code for the help topic. These codes are assigned by the help system author. If `topic%` is zero, this closes any open help file and changes the default help file to `file$` or sets it back to the standard help file if `file$` is omitted or empty.

`topic$` When used with a `.chm` help file: a string holding a help topic keyword or phrase to look-up. From the on-line help show the Index pane (click Show at the top if it is not visible, then click the Index tab). You can show the help for any top level index entry that has an associated page by using the index entry. For example, this help entry (as are all script commands) is indexed by its name, so using `Help("Help()");` will open this page.

When used with other help formats (in the future as we anticipate moving away from the `.chm` format) this will open a specific help page, but you must know the name of the page. For example, all script command have lower-case names of the form `"scr_"+command_name`, so using `Help("scr_help");` will open this page.

`file$` If this is omitted, or the string is empty, the standard Spike2 help file is used. If this holds a file name, it is used as the help file. If `topic%` is present and zero, this sets the name of the help file to use for future help requests until the script terminates.

Returns 1 if the help topic was found, 0 if it was not found, -1 if no help file was found. `Help(0, file$)` returns 0 if the new file exists and -1 if it does not.

The Windows SDK has some help-authoring tools, and third-party tools are available if you wish to generate your own `.chm` files. We use the Help&Manual tool to generate our online help.

I

Get IIR filter information
Apply IIR filter to an array
`IIRApply()`
`IIRBp()`
`IIRBs()`
`IIRComment$()`
`IIRCreate()`
`IIRHp()`
`IIRInfo()`
`IIRLp()`
`IIRName$()`
`IIRNotch()`
`IIRReson()`
`InfoCloseAll()`
`InfoImage()`
`InfoOpen()`
`InfoSettings()`
`InfoSpeak()`
`Inkey()`
`Input()`
`Input$()`
`InStr()`
`InStrRE()`
Grammars
Regular expressions
`Interact()`

IIR commands

The `IIRxxxx()` script commands make it easy for you to generate and apply IIR (Infinite Impulse Response) filters to data held in arrays of real numbers. The data values are assumed to be a sampled sequence, spaced at equal intervals. You can create digital filters that are modelled on Butterworth, Bessel, Chebyshev type 1 and Chebyshev type 2 high pass, low pass, bandstop and bandpass filters. You can also create digital resonators and notch filters. The commands are:

`IIRBp()`	Bandpass filter	`IIRHp()`	High pass filter	`IIRNotch()`	Notch filter
`IIRBs()`	Bandstop filter	`IIRLp()`	Low pass filter	`IIRReson()`	Resonator

The algorithms used to create the filters are based on the `mkfilter` program, written by Tony Fisher of York University. The basic idea is to position the s-plane poles and zeros for a normalised low-pass filter of the desired characteristic and order, then to transform the filter to the desired type. The poles of a filter must lie outside the unit circle for a filter to be stable. The filter commands can return how close the poles are to the unit circle and this value limits the stability of the generated filters; this distance is termed the *stability value* in the command descriptions and experience shows that values less than 1e-12 tend to produce noisy or unstable filters.

Filter expression

The theory of IIR filters is beyond the scope of this manual; a classic reference work is *Theory and Application of Digital Signal Processing* by Rabiner and Gold, published in 1975 or see this Wikipedia article. The IIR filters generated by these commands can be modelled by:

$$y[n] = \sum_{i=0,N} (A_i * x[n-i]/G) + \sum_{i=1,M} (B_i * y[n-i])$$

where the x[n] are the sequence of input data values, the y[n] are the sequence of output values, the A_i and the B_i are the filter coefficients (some of which may be zero) and G is the filter gain. Although G could be incorporated into the A_i, for computational reasons we keep it separate. In the filters designed by the `IIRxxxx()` commands, N=M and is the order of the filter for low pass and high pass designs, twice the order for bandpass and bandstop designs and is 2 for resonators and notch filters. The order of the low pass, high pass, band pass and band stop filters determines the sharpness of the filter cut-off: the higher the order, the sharper the cut-off.

IIR and FIR filters

When compared to FIR filters, IIR filters have advantages:

- They can generate much steeper edges and narrower notches for the same computational effort.
- The filters are causal, which means that the filter output is only affected by current and previous data. If you run a step change through FIR filters you typically see ringing before the step as well as after it.

However, they also have disadvantages:

- IIR filters are prone to stability problems, FIR filters are unconditionally stable. IIR filters generated by these commands will normally be stable unless very narrow features (<0.0001 of the sample rate) are used. Stability gets worse as the filter order increases. You can get an indication of stability after generating a filter.
- They impose a group delay on the data that varies with frequency. This means that they do not preserve the shape of a waveform, in particular, the positions of peaks and troughs will change.

You can remedy the group delay problem by running a filter forwards, then backwards, through the data. However, this makes the filter non-causal, removing one of the advantages of using an IIR filter. The commands allow you to check the impulse, step, frequency and phase response of the filters, and we recommend that you do so before using a generated filter for a critical purpose.

The lowpass, highpass, bandpass and bandstop filters generate digital filters modelled on four types of analogue filter: Butterworth, Bessel, Chebyshev type 1 and Chebyshev type 2. The resulting digital filters are not identical to the analogue filters as the mapping from the analogue to the digital domain distorts the frequency scale. In many cases, this improves the performance of the digital filter over the analogue counterpart.

Filter types
You can generate notch and resonator filters plus lowpass, highpass, bandpass and bandstop filters modelled on Butterworth, Bessel and Chebyshev analogue filters.

Butterworth
These have a maximally flat pass band, but pay for this by not having the steepest possible transition between the pass band and the stop band.

Bessel
An analogue Bessel filter has the property that the group delay is maximally flat, which means that it tends to preserve the shape of a signal passed through it. This leads to filters with a gentle cut-off. When digitised, the constant group delay property is compromised; the higher the filter order, the worse the group delay.

Chebyshev type 1
Filters of this type are based on Chebyshev polynomials and have the fastest transition between the pass band and the stop band for a given ripple in the pass band and no ripples in the stop band.

Chebyshev type 2
Filters of this type are defined by the start of the stop band and the stop band ripple. The filter has the fastest transition between the pass and stop bands given the stop band ripple and no ripple in the pass band.

Notch
Notch filters are defined by a centre frequency and a `q` factor. `q` is the width of the stop band at the –3 dB point divided by the centre frequency: the higher the `q`, the narrower the notch. Notch filters are sometimes used to remove mains hum, but if you do this you will likely need to set notches at the first few odd harmonics of the mains frequency. If you have a fairly constant coupling of mains noise into your signal, there are other ways to remove it that may cause much less signal degradation than notch filtering. There is a Hum Remove script on the CED web site that implements one method (select Spike2 scripts).

Resonator
A resonator is the inverse of a notch. It is defined in terms of a centre frequency and a `q` factor. `q` is the width of the pass band at the –3 dB point divided by the centre frequency: the higher the `q`, the narrower the resonance. Resonators are sometimes used as alternatives to a narrow bandpass filter.

Non-filter bank commands
The `IIRBp()`, `IIRBs()`, `IIRLp()`, `IIRHp()`, `IIRReson()` and `IIRNotch()` commands are all independent and they each remember the last filter you created and the filter state after the last filtering operation. They have common operations to read back data and apply the filter to an array.

Filter bank commands
The `IIRApply()`, `IIRComment$()`, `IIRCreate()`, `IIRInfo()` and `IIRName$()` commands use the IIR filter bank to generate and apply digital filters to channels of data.

Command timings
To give you an idea of the speed of these commands, we measured the time taken in milliseconds to apply the filters to a vector of 10,000,000 points. The timings were done on an Intel Core i7-4770K CPU @ 3.50GHz.

Type/order	1	2	3	4	5	6	7	8	9	10
Low pass	37	37	38	88	111	146	144	177	161	192
High pass	37	39	38	90	106	144	138	167	147	173
Band pass	38	37	65	173	174	191	214	238	503	494
Band Stop	88	88	125	172	181	190	218	234	505	492
Resonator		39								

Notch	40

The Low pass, High pass, and Band pass filters have hand-tuned code for orders 1 to 3 which makes them run a bit faster. The Resonator and Notch filters are second order filters and also have hand-tuned code. All the other filter cases use a more generic implementation that is not quite as efficient.

See also:
`FIRMake()`, `IIRApply()`, `IIRBp()`, `IIRBs()`, `IIRComment$()`, `IIRCreate()`, `IIRHp()`, `IIRInfo()`, `IIRLp()`, `IIRName$()`, `IIRNotch()`, `IIRReson()`

Get IIR filter information

For the `IIRBp()`, `IIRBs()`, `IIRLp()`, `IIRHp()`, `IIRReson()` and `IIRNotch()` commands, you use a variant of the following form to return filter information:

```
Func IIRxxxx(get%{, arr[]{[]}});
```

`get%` The form of the command using this argument is used to read back information about the last filter created with this command. The argument can be:

1 `arr[]` is set to the impulse response of the filter. This is the response to an input of 1 followed by an infinite number of zeros assuming that all previous inputs were also zero. It is up to you to set a suitable length of the array. Each array data points corresponds with one sample interval. The return value is the magnitude of the largest value in `arr[]`. For example, if the impulse response ranged in values from –0.5 to 0.3, 0.5 would be returned.

2 `arr[]` is set to the step response of the filter. The input to the filter is assumed to be an infinite number of zeros, followed by an infinite number of ones, and the response is taken from the moment where the input changes to a 1. It is up to you to set a suitable array length. The return value is the magnitude of the largest value in `arr[]`.

3 `arr[][]` is a matrix with r rows and 2 columns. The frequency response is returned as complex numbers in the columns; `arr[][0]` holds the real part and `arr[][1]` the imaginary part. The first row corresponds to a frequency of 0; the final row corresponds to a frequency of half the sampling rate. The frequencies are spaced as `0.5/(r-1)`. The more rows you set, the finer the frequency response. The return value is the maximum magnitude of the returned frequency response.

4 The same as 3 except that the results are returned as the amplitude response in column 0 and the phase response in column 1. If the real and imaginary parts of the response are `r` and `i`, `arr[][0]` holds `sqrt(r*r+i*i)` and `arr[][1]` holds `atan(i, r)`. The return value is the maximum returned amplitude response.

5 Returns the number of filter coefficients (N+1) to apply to the filter input values and fills in `arr[]` with these values. These correspond to the a_i in the filter expression. However, we return the values in reverse order as this makes them easier to use as a dot product with old values. N is the filter order for low pass and high pass, twice the order for band pass and band stop, and 2 for resonators and notch filters.

6 Returns the number of filter coefficients (N+1) to apply to the filter output values and fills in `arr[]` with these values. These correspond to the b_i in the filter expression. However, we return the values in reverse order to match the a_i. The final value is always –1.0 (corresponding to b_0, which is not used when implementing the filter). N is the filter order for low pass and high pass, twice the order for band pass and band stop, and 2 for resonators and notch filters.

7 Returns the filter gain G as defined in the filter expression.

8 `arr[][]` is a matrix with N rows and 2 columns. N is the filter order for low pass and high pass, twice the order for band pass and band stop, and 2 for resonators and notch filters. The return value is the number of poles in the s-plane and the matrix is filled in with the poles as complex numbers with the real part in column 0 and the imaginary part in column 1; `arr[][0]` holds the real part and `arr[][1]` the imaginary part. If you do not know what the s-plane and z-plane are, you can find a reference in the IIR commands section.

9 The same as 8, but returning the s-plane zeros.

10 The same as 8, but returning the z-plane poles.

11 The same as 8, but returning the z-plane zeros.

12 Returns a measure of filter stability, being the distance of the nearest pole to the unit circle. It is our experience that values greater than 1e-12 generate plausible filters. As the filter depends on a function of this distance, which is of the form (pole position-1), as the pole position approaches 1, the numerical accuracy of the result become significantly compromised (floating point numbers have around 15 significant digits of accuracy, all else being equal). You can improve the stability by reducing the order of the filter. Reporting stability was added at Spike2 version [7.09].

To read back information about a filter in the filter bank, use the `IIRInfo()` command.

See also:
`FIRMake()`, `IIRApply()`, `IIRBp()`, `IIRBs()`, `IIRHp()`, `IIRInfo()`, `IIRLp()`, `IIRNotch()`, `IIRReson()`

Apply IIR filter to an array

This command applies the current filter set by one of the `IIRBp()`, `IIRBs()`, `IIRLp()`, `IIRHp()`, `IIRReson()` and `IIRNotch()` commands to an array of equally spaced data.

```
Func IIRBp(data[]{, flags%{, save[]}});
Func IIRBs(data[]{, flags%{, save[]}});
Func IIRLp(data[]{, flags%{, save[]}});
Func IIRHp(data[]{, flags%{, save[]}});
Func IIRReson(data[]{, flags%{, save[]}});
Func IIRNotch(data[]{, flags%{, save[]}});
```

`data` An array of data to filter.

`flags%` Optional, taken as 0 if omitted. It is the sum of:

1 To apply the filter backwards through the array. Applying a filter introduces a phase shift; running a filter forwards then backwards cancels the phase shift at the expense of a non-causal filter.

2 To treat `data[]` as a continuation of the last filtering operation (only do this if it really is a continuation, otherwise the results are nonsense). The filter state is saved separately for each filter type, but if you want to interleave use of the same command between multiple data streams, you must use the `save[]` argument. If you apply the filter backwards, you must present the data blocks backwards.

`save[]` Optional. This is a real vector that preserves the state of the IIR filter for a data stream so that you can interleave continuous filtering with the same filter between multiple streams. The minimum size of the array depends on the filter type and order. It is 2*order+2 for `IIRLp()` and `IIRHp()`, 6 for `IIRReson()` and `IIRNotch()` and 4*order+2 for `IIRBp()` and `IIRBs()`.

If `save` is present and 2 is added to `flags%`, the filter state is loaded from it before filtering; it is always updated after filtering. You can simplify code by always having the continuous data flag set if you make sure that `save[]` is initialized to zeros before it is used for the first time.

When would you use save[]?

IIR filters are recursive and causal, which means that the output of the filter at a particular time depends on all the data values up to that time. If you apply the filter as a one-off operation, this is not a problem. However, if you want to apply a filter in real time, or you want to filter a huge file and display your progress, you will need to apply the filter incrementally. For example, you might process from time 0 up to 1 second, then from 1 second to 2 and so on.

To allow you to do this you can set a flag to say that the operation is a continuation of the previous operation. Each of the six `IIRxxx()` commands saves the filter "state" internally, so if you are working on a single data stream you can just set the continuous flag and the command will take care of this for you.

However, if you want to apply the same filter to multiple data streams incrementally, you must supply storage space to save the internal state for each data stream. You cannot share the `save[]` vector between streams; either have different vector for each stream or use an array of two dimensions, one for the streams and one for saving the state. For this operation to make sense, each stream must have the same sample rate.

The following example applies the identical resonator filter to two data streams incrementally. You are expected to call `FilterTwoStreams()` with `isFirst%` set to 1 for the first call and then with it set to 0 for subsequent calls with contiguous data. The `saveReson` variable holds the state for each stream.

```
var saveReson[6][2];  'Space for two channels of Resonator state
...
Proc FilterTwoStreams(stream0[], stream1[], isFirst%);
if isFirst% then ArrConst(saveReson, 0) endif;
IIRReson(stream0[], 2, saveReson[][0]);
IIRReson(stream1[], 2, saveReson[][1]);
end;
```

Each `IIRXxxx()` command has its own internal state storage, so if you were processing two streams with different filter commands there would be no need to save the state as the commands would do it for you.

What you cannot do incrementally

Each `IIRXxxx()` command holds the filter state and the set of filter coefficients required to implement the filter. You cannot apply two filters of the same type (for example two low pass filters) with different corner frequencies at the same time without recalculating the filter before each use.

See also:

`FIRMake()`, `IIRApply()`, `IIRBp()`, `IIRBs()`, `IIRHp()`, `IIRLp()`, `IIRNotch()`, `IIRReson()`

IIRApply()

Applies a filter in the IIR filter bank to a waveform or RealWave channel in the current time view and places the result in a destination memory buffer or a disk-based channel. The output can be written as either a waveform or as a RealWave channel.

```
Func IIRApply(index%, dest%, srce%, sTime, eTime {,flags%});
```

`index%` Index of the filter in the filter bank to apply in the range -1 to 11. Use -1 for a temporary filter (you must have created it first, see `IIRCreate()`).

`dest%` The channel to hold the filtered waveform: either an unused disk channel, a memory channel with the same sampling frequency as `srce%` or 0 to create a compatible memory channel and place the filtered waveform in it. When a new channel is created, the channel settings are copied from the old channel.

`srce%` The source waveform or RealWave channel. There must be at least half the number of sampling coefficients worth of data points before `sTime` if the output is to start at `sTime`. Similarly, the channel must extend for the same number of data points beyond `eTime` if the output is to extend to `eTime`.

`sTime` Time to start the output of filtered data. There is no output for areas where there is no input data. If the filter has an even number of coefficients, the output is shifted by half a sample relative to the input.

`eTime` The end of the time range for filtered data.

`flags%` Optional, default value 0. The sum of the following flag values:

- 1 Optimises the destination channel scale and offset values to give the best possible representation of the output as 16-bit integers. For Waveform output channels, this doubles the processing time and existing data in the output channel that is not overwritten is also rescaled. If you do not re-scale, the channel's scale and offset are unchanged. However, you run the risk of the output being clipped to the 16-bit range allowed for a waveform channel
- 2 Create a RealWave destination channel in place of a Waveform channel.
- 4 Display a progress dialog (allowing the user to cancel the filter) if the filter takes more than a second or so. This option was added at [10.03]. You can set this value in previous versions of Spike2 without error, however it has no effect. It records as set (the interactive dialog allows a progress bar).

Returns The channel number that the output was written to or a negative error code. A negative error code is also returned if the user clicks Cancel from the progress bar that may appear during a long filtering operation or if `dest%` is a disk channel that is in use. Delete an existing channel with `ChanDelete(dest%)`.

See also:
The filter bank, `ChanDelete()`, `IIRComment$()`, `IIRCreate()`, `IIRBp()`, `IIRInfo()`, `IIRName$()`

IIRBp()

This function creates and applies IIR (Infinite Impulse Response) band pass filters to arrays of data. You can run the filter forwards or backwards through the data.

```
Func IIRBp(data[]|0, lo, hi, order%{, type%{, ripple}});
Func IIRBp(data[]{, flags%{, save[]}});
Func IIRBp(get%{, arr[]{[]}});
```

`data` An array of data to filter. If there are only 1 or 2 arguments, the last created band pass filter is used. Otherwise, the filter defined by the remaining arguments is used forwards. Replace `data` with 0 to create a filter without applying it; this filter survives until you use this command to define another filter.

`flags%` Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat `data[]` as a continuation of the last filtering operation.

`save[]` Optional. If present it must have a size of at least `4*order%+2`. It is used to save the filter state to allow use with the continued data flag when applying the same filter to multiple data streams.

`lo` The low corner frequency of the band stop filter as a fraction of the sample rate in the range 0.000001 to 0.499998. For Chebyshev type 2 filters, this is the point at which the attenuation reaches the `ripple` value, for all other filters this sets the –3 dB point.

`hi` The high corner frequency of the band pass filter as a fraction of the sampling rate in the range `lo`+0.000001 to 0.499999. For Chebyshev type 2 filters, this is the point at which the attenuation reaches the `ripple` value, for all other filters this sets the –3 dB point.

`order%` The order of the low pass filter used as the basis of the design, in the range 1 to 10. The order of the filter implemented is `order%*2`. High orders (`order%` > 7) and narrow bands may cause inaccuracy in the filter. Narrow means that `(hi-lo)/sqrt(lo*hi)` is less than 0.2, for example.

`type%` Set 0 for Butterworth, 1 for Bessel and 2 for Chebyshev type 1, 3 for Chebyshev type 2. The default value is 0 for a Butterworth filter.

`ripple` The desired pass band ripple in dB for Chebyshev type 1 filters (default 3 dB) or the desired minimum cut in the stop bands for Chebyshev type 2 filters (default 40 dB). The ripple value must be in the range 0.01 to 1000 dB.

`get%` The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.

`arr` An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.

Returns All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for success or 1 if a created filter has stability less than 1e-12 or -1 if the filter could not be created. The other command forms have their return values included in the description of the `get%` argument or return 0.

See also:
More about IIR filters and commands, Get filter information, `FIRMake()`, `IIRBs()`, `IIRHp()`, `IIRLp()`, `IIRNotch()`, `IIRReson()`

IIRBs()

This function creates and applies IIR (Infinite Impulse Response) band stop filters to arrays of data. You can run the filter forwards or backwards through the data.

```
Func IIRBs(data[]|0, lo, hi, order%{, type%{, ripple}});
Func IIRBs(data[]{, flags%{, save[]}});
Func IIRBs(get%{, arr[]{[]}});
```

`data` An array of data to filter. If there are only 1 or 2 arguments, the last created band stop filter is used. Otherwise, the filter defined by the remaining arguments is applied forwards. Replace `data` with 0 to create a filter without applying it; this filter survives until you use this command to define another filter.

`flags%` Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat `data[]` as a continuation of the last filtering operation.

`save[]` Optional. If present it must have a size of at least 4`*order%+2`. It is used to save the filter state to allow use with the continued data flag with multiple data streams.

`lo` The low corner frequency of the band pass filter as a fraction of the sample rate in the range 0.000001 to 0.499998. For Chebyshev filters, this is the point at which the attenuation reaches the `ripple` value, for other filters this sets the –3 dB point.

`hi` The high corner frequency of the band pass filter as a fraction of the sampling rate in the range `lo`+0.000001 to 0.499999. For Chebyshev type 2 filters, this is the point at which the attenuation reaches the `ripple` value, for all other filters this sets the –3 dB point.

`order%` The order of the low pass filter used as the basis of the design, in the range 1 to 10. The order of the filter implemented is `order%*2`. High orders and narrow pass bands may lose numerical accuracy in the filter output.

`type%` Set 0 for Butterworth, 1 for Bessel and 2 for Chebyshev type 1, 3 for Chebyshev type 2. The default value is 0 for a Butterworth filter.

`ripple` The desired pass band ripple in dB for Chebyshev type 1 filters (default 3 dB) or the desired minimum cut in the stop band for Chebyshev type 2 filters (default 40 dB). The ripple value must be in the range 0.01 to 1000 dB.

`get%` The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.

`arr` An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.

Returns All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for successor 1 if a created filter has stability less than 1e-12 or -1 if the filter could not be created. The other command forms have their return values included in the description of the `get%` argument or return 0.

See also:
More about IIR filters and commands, Get filter information, `FIRMake(), IIRBp(), IIRHp(), IIRLp(), IIRNotch(), IIRReson()`

IIRComment$()

This function gets and sets the comment associated with an IIR filter in the filter bank.

```
Func IIRComment$(index% {, new$});
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`new$` If present, sets the new comment.

Returns The previous comment for the filter at the index.

See also:
Filter banks, `IIRApply(), IIRInfo(), IIRName$()`

IIRCreate()

This creates an IIR filter description and adds it to the filter bank.

```
Func IIRCreate(index%, type%, model%, order%, fr1{,fr2{,extra}});
```

`index%` Index of the filter in the filter bank in the range -1 to 11.

`type%` Sets the filter type as: 0=Low pass, 1=High pass, 2=Band pass, 3=Band stop.

`model%` Sets the filter model: 0=Butterworth, 1=Bessel, 2=Chebyshev type 1, 3=Chebyshev type 2, 4=Resonator.

`order%` Sets the filter order in the range 1-10. Resonators always set an order of 2.

`fr1` Sets the corner frequency in Hz for low pass, high pass filters, the centre frequency for resonators, and the low corner frequency for band pass and band stop filters.

`fr2` Sets the upper corner frequency in Hz for band pass/stop filters, otherwise ignored.

`extra` Sets the ripple for Chebyshev filters in dB in the range 0.01 to 1000 and the Q factor for resonators in the range 1 to 10000. If you are unsure of what is a suitable value, use the interactive IIR digital filter dialog to design a suitable filter and copy (or record) your settings.

Returns Always returns 0 as all errors (setting impossible values) stop the script.

See also:
The filter bank, `IIRApply(), IIRComment$(), IIRCreate(), IIRName$()`

IIRHp()

This function creates and applies IIR (Infinite Impulse Response) high pass filters to arrays of data. You can run the filter forwards or backwards through the data.

```
Func IIRHp(data[]|0, edge, order%{, type%{, ripple}});
Func IIRHp(data[]{, flags%{, save[]}});
Func IIRHp(get%{, arr[]{[]}});
```

`data` An array of data to filter. If there are only 1 or 2 arguments, the last created high pass filter is used. Otherwise, the filter defined by the remaining arguments is applied forwards. Replace `data` with 0 to create a filter without applying it; this filter survives until you use this command to define another filter.

`flags%` Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat `data[]` as a continuation of the last filtering operation.

`save[]` Optional. If present it must have a size of at least `2*order%+2`. It is used to save the filter state to allow use with the continued data flag with multiple data streams.

`edge` The corner frequency of the high pass filter as a fraction of the sample rate in the range 0.000001 to 0.499999. For Chebyshev filters, this is the point at which the attenuation reaches the `ripple` value, for other filters this sets the –3 dB point.

`order%` The order of the filter in the range 1 to 10.

`type%` Set 0 for Butterworth, 1 for Bessel and 2 for Chebyshev type 1, 3 for Chebyshev type 2. The default value is 0 for a Butterworth filter.

`ripple` The desired pass band ripple in dB for Chebyshev type 1 filters (default 3 dB) or the desired minimum cut in the stop bands for Chebyshev type 2 filters (default 40 dB). The ripple value must be in the range 0.01 to 1000 dB.

`get%` The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.

`arr` An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.

Returns All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for successor 1 if a created filter has stability less than 1e-12 or -1 if the filter could not be created. The other command forms have their return values included in the description of the `get%` argument or return 0.

See also:
More about IIR filters and commands, Get filter information, `FIRMake(), IIRBp(), IIRBs(), IIRLp(), IIRNotch(), IIRReson()`

IIRInfo()

Retrieves information about an IIR filter in the bank.

```
Func IIRInfo(index%, &model%, &order%, &fr1{, &fr2{, &extra}});
```

`index%` Index of the filter in the filter bank in the range -1 to 11.

`model%` Returned as the filter model: 0=Butterworth, 1=Bessel, 2=Chebyshev type 1, 3=Chebyshev type 2, 4=Resonator.

`order%` Returned as the filter order in the range 1-10. Resonators always return 2.

`fr1` Returned as the corner frequency for low and high pass filters, as the low corner for band pass and band stop filters and as the centre frequency for resonators.

`fr2` Returned as the upper corner frequency for band pass and band stop filters, otherwise set the same as `fr1`.

`extra` Returned as the ripple for Chebyshev filters and as the Q factor for resonators.

Returns The type of the filter as 0=Low pass, 1=High pass, 2=Band pass, 3=Band stop.

See also:
The filter bank, `IIRApply(), IIRComment$(), IIRCreate(), IIRName$()`

IIRLp()

This function creates and applies IIR (Infinite Impulse Response) low pass filters to arrays of data. You can run the filter forwards or backwards through the data.

```
Func IIRLp(data[]|0, edge, order%{, type%{, ripple}});
Func IIRLp(data[]{, flags%{, save[]}});
Func IIRLp(get%{, arr[]{[]}});
```

`data` An array of data to filter. If there are only 1 or 2 arguments, the last created low pass filter is used. Otherwise, the filter defined by the remaining arguments is applied forwards. Replace `data` with 0 to create a filter without applying it; this filter survives until you use this command to define another filter.

`flags%` Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat `data[]` as a continuation of the last filtering operation.

`save[]` Optional. If present it must have a size of at least `2*order%+2`. It is used to save the filter state to allow use with the continued data flag with multiple data streams.

`edge` The corner frequency of the low pass filter as a fraction of the sample rate in the range 0.000001 to 0.499999. For Chebyshev filters, this is the point at which the attenuation reaches the `ripple` value, for other filters this sets the –3 dB point.

`order%` The order of the filter in the range 1 to 10.

`type%` Set 0 for Butterworth, 1 for Bessel and 2 for Chebyshev type 1, 3 for Chebyshev type 2. The default value is 0 for a Butterworth filter.

`ripple` The desired pass band ripple in dB for Chebyshev type 1 filters (default 3 dB) or the desired minimum cut in the stop bands for Chebyshev type 2 filters (default 40 dB). The ripple value must be in the range 0.01 to 1000 dB.

`get%` The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.

`arr` An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.

Returns All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for success or 1 if a created filter has stability less than 1e-12 or -1 if the filter could not be created. The other command forms have their return values included in the description of the `get%` argument or return 0.

See also:
More about IIR filters and commands, Get filter information, Backwards example, `FIRMake(), IIRBp(), IIRBs(), IIRHp(), IIRNotch(), IIRReson()`

Backwards filter example

IIR filters are not linear phase, that is they delay different frequencies by different times. This can be an issue as it may significantly move the positions of peaks. You can reduce this effect by running the filter twice, once forwards and once backwards. This is not something that can be done in real time!

Here is an example script to apply a low-pass Butterworth filter to forwards and then backwards to a data channel, writing the result to a memory channel. This was tested with the `demo.smr` file that is shipped with Spike2. This script will run on Spike2 version 7 onwards.

```
'Apply a low pass butterworth IIR filter forwards and backwards, creating a memory channel
'wc%    The input waveform or realwave channel, Data assumed contiguous in time range.
'tFrom  The start time to process from
'tUpto  The end time to process up to
'fEdge  The corner frequence (must be >0, <= sample rate/2)
'rw%    Set 0 for Waveform output, 1 for RealWave
'Return The channel number of the memory channel or a negative error code
Func IIRLpForBack(wc%, tFrom, tUpto, fEdge, rw%)
const type% := 0;               'low pass
const model% := 0;              'Butterworth
const order% := 4;              'Fourth order
var kind% := ChanKind(wc%);     'Get the type of the input channel
if (kind%<>1) and (kind%<>9) then return -1; endif;  'Must be a waveform
var bSz := BinSize(wc%);        'Time increment per bin
if IIRCreate(-1, type%, model%, order%, fEdge) < 0 then return -1 endif;  'temporary filter
if IIRLp(0, fEdge*bSz, order%) <> 0 then return -1 endif;     'Create same for backwards
var mc% := IIRApply(-1, 0, wc%, tFrom, tUpto, rw% ? 2 : 1);   'Apply filter to create channel
if (mc% < 0) then return mc% endif;  'Stop if we failed

const kN% := 10000;             'Buffer size for read
var wave[kN%];                  'Buffer space used for read
var nGot%;                      'Points read by ChanData

var t := tUpto, tBack, tFirst;
var flags% := 1;                'run backwards, first block not contiguous
while t > tFrom do
    tBack := t - bSz * kN%;     'Maximum backwards start position
    if (tBack < tFrom) then tBack := tFrom endif; 'limit it
    nGot% := ChanData(mc%, wave, tBack, t, tFirst);
    if (nGot% > 0) then
        flags% := (tFirst - t >= bSz) ? 1 : 3; 'detect gaps
        IIRLp(wave[:nGot%], flags%); 'Apply backwards
        ChanWriteWave(mc%, wave[:nGot%], tFirst);
        t := tFirst - bSz;      'New last point time
```

```
    else
        t := -1;                'Time to stop
    endif;
wend;
return mc%;
end;

'Open demo.smr file to run 10 Hz Lp forwards and backwards
var mc% := IIRLpForBack(1, 0, Maxtime(), 10.0, 0);
ChanShow(mc%);
```

IIRName$()

This function gets and/or sets the name of an IIR filter in the filter bank.

```
Func IIRName$(index% {, new$});
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`new$` If present, sets the new name.

Returns The previous name of the filter at that index.

See also:
Filter banks, `IIRApply(), IIRComment$(), IIRInfo()`

IIRNotch()

This function creates and applies IIR (Infinite Impulse Response) notch filters to arrays of data. You can run the filter forwards or backwards through the data. The gain of the notch filter is zero at the notch frequency and 1 at low and high frequencies.

```
Func IIRNotch(data[]|0, fr, q);
Func IIRNotch(data[]{, flags%{, save[]}});
Func IIRNotch(get%{, arr[]{[]}});
```

`data` An array of data to filter. If there are only 1 or 2 arguments, the last created notch filter is used. Otherwise, the filter defined by the remaining arguments is applied forwards. Replace `data` with 0 to create a filter without applying it; this filter survives until you use this command to define another filter.

`flags%` Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat `data[]` as a continuation of the last filtering operation.

`save[]` Optional. If present it must have a size of at least 6. It is used to save the filter state to allow use with the continued data flag with multiple data streams.

`fr` The frequency of the notch as a fraction of the sample rate in the range 0.000001 to 0.499999. However, the frequency range for which a notch can be calculated depends on the `q` factor. If the commands returns -1, it could not generate the desired filter; you *must* always check the returned value.

`q` The `q` factor for the notch in the range 1 to 10000; the higher the q, the narrower the notch. If `Flo` and `Fhi` are the frequencies of the –3 dB points either side of the notch, `q` is `fr/(Fhi-Flo)`. Try 100 as a starting point.

`get%` The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.

`arr` An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.

Returns All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for success or 1 if a created filter has stability less than 1e-12 or -1 if we could not generate a filter. The other command forms have their return values included in the description of the `get%` argument or return 0.

See also:
More about IIR filters and commands, Get filter information, `FIRMake(), IIRBp(), IIRBs(), IIRHp(), IIRLp(), IIRReson()`

IIRReson()

This function creates and applies IIR (Infinite Impulse Response) resonator filters to arrays of data. You can run the filter forwards or backwards through the data. The gain of the filter is 1 at the resonator frequency and zero at low and high frequencies.

```
Func IIRReson(data[]|0, fr, q);
Func IIRReson(data[]{, flags%{, save[]}});
Func IIRReson(get%{, arr[]{[]}});
```

`data` An array of data to filter. If there are only 1 or 2 arguments, the last created resonator filter is used. Otherwise, the filter defined by the remaining arguments is applied forwards. Replace `data` with 0 to create a filter without applying it; this filter survives until you use this command to define another filter.

`flags%` Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat `data[]` as a continuation of the last filtering operation.

`save[]` Optional. If present it must have a size of at least 6. It is used to save the filter state to allow use with the continued data flag with multiple data streams.

`fr` The centre frequency of the resonator as a fraction of the sample rate in the range 0.000001 to 0.499999. However, the frequency range for which a notch can be calculated depends on the `q` factor. If the commands returns -1, it could not generate the desired filter; you *must* always check the returned value.

`q` The `q` factor for the resonator in the range 1 to 10000; the higher the q, the narrower the resonance. If `Flo` and `Fhi` are the frequencies of the –3 dB points either side of the resonance, `q` is `fr/(Fhi-Flo)`. Try 100 as a starting point.

`get%` The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.

`arr` An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.

Returns All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for successor 1 if a created filter has stability less than 1e-12 and -1 if the filter could not be created. The other command forms have their return values included in the description of the `get%` argument or return 0.

See also:
More about IIR filters and commands, Get filter information, `FIRMake(), IIRBp(), IIRBs(), IIRHp(), IIRLp(), IIRNotch()`

Info...()

The `Info...()` family of commands support the Info windows, used to display easily-read information in a large font size to users. These windows are typically used during data capture, but are also useful in other situations.Info windows were added to Spike2 at version [10.00].

See also:
`InfoOpen(), InfoCloseAll(), InfoSettings(), InfoImage(), InfoRun(), FileClose()`

InfoCloseAll()

This function closes all the information windows attached to the current view.

```
Proc InfoCloseAll();
```

See also:
`InfoOpen(), InfoSettings(), InfoImage(), InfoRun(), FileClose()`

InfoImage()

This function sets and retrieves information about the image that is displayed in the current information window. This is equivalent to the Info Image dialog.

Set or clear image

```
Func InfoImage(path$);
```

`path$` A string holding the full path and name of the image file to display in the information window. This can be set to `"<CB>"` to display the image currently on the clipboard. A non-blank string referring to a non-existent file displays a question mark as a place holder. An empty string clears the image.

Returns The function result is 1 if an image is set and found, 0 if not.

Set image state and position

```
Func InfoImage(hide%, opa{, left, top, width, height});
```

`hide%` This sets the new image hidden state: 0 for visible, 1 for hidden, -1 for no change.

`opa` This sets the image opacity from 0.0 for fully transparent to 1.0 for completely opaque, or -1 for no change.

`left` This sets the X position of the top left corner of the image area as a percentage of the info window width, from 0 to 100. A value of zero is used if this is omitted.

`top` This sets the Y position of the top left corner of the image area as a percentage of the info window height, from 0 to 100. A value of zero is used if this is omitted.

`width` This sets the width of the image area as a percentage of the info window width, from 0 to 100. A value of 100 is used if this is omitted.

`height` This sets the height of the image area as a percentage of the info window height, from 0 to 100. A value of 100 is used if this is omitted.

Returns The function result is 0.

Read back image information

This form of the command returns information about the displayed image.

```
Func InfoImage(opt%{, path$});
```

`opt%` A value controlling the information that is returned:

`opt%`	Function result
-1	The image hidden state: 1 if the image is hidden, 0 if it is shown
-2	The image opacity from 0 for fully transparent to 1 for completely opaque
-3	The X position of the top left corner of the image area as a percentage.
-4	The Y position of the top left corner of the image area as a percentage.
-5	The width of the image area as a percentage.
-6	The height of the image area as a percentage.

`path$` If present, set to the stored image name.

Returns The function result is the image information value as selected by opt%.

See also:
InfoOpen(), InfoSettings(), InfoCloseAll(), InfoRun()

InfoOpen()

This function creates a new information window with the current Time, Result or XY view as its parent. See the InfoSettings() command for detailed descriptions of the arguments from text$ onwards. Note that once the window is open you can use FontSet() and FontGet() commands to manipulate the font.

```
Func InfoOpen({n%{, mode%{,
              text${, flags%{, reset{, stop{, step%{, show%}}}}}}}});
```

n% A value in the range -1 to 10 or omitted. Set to 0 to open a new Info window. Omit or set to -1 to query the count of information windows owned by the current view. If greater than 0, get the handle of the n^{th} information window associated with the current view. The remaining arguments are valid only when n% is 0.

mode% This optional argument has the default value 0 and is the sum of the following:

1 Make new windows visible. Omitting this does not hide an existing window.

text$ The text to generate the new information window. If the text includes a timer (%t), the timer will start in a running state.

flags% Set of flags to control the Timer and display. 0 if omitted.

reset The time value, in seconds to set on a timer reset. 0 if omitted.

stop The time at which the timer should stop, when stop is enabled, in seconds. 0 if omitted.

step% Sets the timer update rate: 0=0.1 seconds, 1=1 second, 2=1 minute, 3=1 hour. 0 if omitted.

show% Sets the timer format: 0=0.1 seconds, 1=seconds, 2=MM:SS. 3=HH:MM:SS. 3 if omitted.

Returns For n% = 0, the handle of the new view, or 0 if we failed to create one. For n% = -1 or omitted, the returned value is the count of information windows attached to the current view. For n% > 0, the returned value is the view handle of info window n%, or 0 if the window does not exist.

Example

The following example displays a message for 10 seconds with a timer that counts down from 00:00:10.0 to zero and closes the info window when the time reaches 0 (requires [10.15]):

```
InfoOpen(0, 1, "Count down %t", 1+2+8+32+128, 10, 0);
```

Note that the current view must be a Time, Result or XY view. This can be used to display a temporary message without the script needing to take action to clear it. Note that the script must release idle time for the display to update and to detect that the counter has reached the stop time.

See also:
InfoImage(), InfoSettings(), InfoCloseAll(), InfoRun(), ViewLink()

InfoRun()

This function starts, stops and resets the timer in the current Info window or all associated information windows if the current window owns Info windows. The command can also read back the timer run state and run time.

```
Func InfoRun({go%});
```

go% Positive values in the range 0-3 control the run and reset state of the timer. These values are the sum of:

1 Run. The timer continues from its current value to count up or down. If 1 is not added, the timer stops.

2 Reset. If 2 is added, the timer is set to its Reset value, otherwise it is not changed.

Omitting this argument (default value is -1) or negative values read back timer information. -1 returns the current timer run state, -2 returns the current timer time. If you send this command to a window that owns Info windows with a negative value of `go%`, the result is for the first Info window.

Return If there is no timer associated with the Info window, this command always returns -1. Otherwise, if `go%` is -2, it returns the current timer value in seconds. In all other cases (including `go% = -1`), it returns 1 if the timer was running at the time of the call, 0 if it was not. It does not report the new run state.

See also:
`InfoOpen(), InfoSettings(), InfoCloseAll(), InfoImage(), ViewLink()`

InfoSettings()

This function changes and reports the settings of the current information window. It has similar functionality to the information window Settings dialog.

```
Func InfoSettings(text$${, flags%{, reset{, stop{, step%{, show%}}}}});
```

`text$` The text that will be displayed by the new information window, with any special fields appropriately interpreted.

`flags%` This parameter controls various information window options, it is the sum of the following values (taken as zero if omitted):

1	Timer counts down, otherwise it counts upwards.
2	Timer stops when the stop value is reached, otherwise stop is ignored.
4	Timer starts when it is reset in the interactive window.
8	Hide all the buttons at the top of the information window (including the Settings button).
16	Lock the text font size. Prevents the text resizing to fill the window.
32	Hide the window title bar.
64	Hide the Settings button in the button bar.
128	Close the Info window when the timer stops (so requires flag value 2 to be set). Added at [10.15]. Note that the window will not close if any Info window dialog is open (Settings, Font, Image, Speak), so this would usually be used with the flag value 8 also set.

`reset` The time in seconds to which the timer is set when it is reset. Set to 0 if omitted. The time is set, and stored to an accuracy of 0.1 seconds (the value is multiplied by 10, then converted to an integer).

`stop` The time in seconds at which the timer will automatically stop if the `flags%` value has enabled this option. For timers configured to count upwards this value (if used) must be greater than `reset`, for timers counting down it must be less than `reset`. Set to zero if omitted. If you supply the value and it is inconsistent with the reset value and the counter direction, the return value is -1. The time is set, and stored to an accuracy of 0.1 seconds.

`step%` Sets how often the displayed time updates and the smallest unit of time shown; 0 for 0.1 seconds, 1 for 1 second, 2 for 1 minute and 3 for 1 hour. Set to 0 if omitted. This sets the fastest update rate and relies on Spike2 having idle time.

`show%` Selects how the time is shown; 0 for a count of 100 millisecond intervals, 1 for a count of seconds, 2 for MM:SS and 3 for HH:MM:SS. Set to 3 if omitted.

Returns -1 if the stop time and reset time are set such that the timer would stop if told to start, otherwise 0.

```
Func InfoSettings(opt%{, &text$});
```

`opt%` To read back the settings, `opt%` must be a negative value that selects what value is returned.

`text$` If present, returns the text that is processed to generated the display.

Returns The function result is a setting value as selected by `opt%`:

opt%	Function result
-1	The `flags%` value.
-2	The `reset` time, in seconds. This is a multiple of 0.1 seconds.

-3 The `stop` time, in seconds. This is a multiple of 0.1 seconds.

-4 The `step%` value in the range 0 to 3.

-5 The `show%` value in the range 0 to 3.

To read back the currently displayed text in the Info window, make the window current and use `EditCopy()` to place the text on the clipboard and `EditPaste()` to collect it.

See also:
`InfoImage(), InfoOpen(), InfoCloseAll(), InfoRun()`

InfoSpeak()

This is the script language equivalent of the Info Speak settings dialog. The current view must be an Info window to use this command. The command has two forms; the first is to set the speak values:

```
Func InfoSpeak(mode%{, gap{, vol%{, rate%}}});
```

`mode%` Set 0 for Off, 1 for On change, 2 for Continuous and 3 for On change, repeat mode.

`gap` The silent gap after an utterance in the range 0.0 to 60.0 seconds, or -1 to not wait for the previous utterance to finish before starting the next. If omitted, the gap is unchanged..

`vol%` The volume of the speech in the range 0 to 100. The values map the full range using an approximately logarithmic scale so that the perceived volume for values in the range 1-100 is linear and 0 is silent. If omitted, the volume is unchanged.

`rate%` The speech speed, in the range -10 to +10. The normal speed is 0, the extreme values are approximately 4 times faster or slower. If omitted, the speech speed is unchanged.

Return -1 if there is no speech support detected. 0 if the command was accepted.

```
Func InfoSpeak({get%});
```

`get%` Determines what information to return. If omitted, it is taken as -1.

Return The value selected by the `get%` argument:

-1 Detect the presence of speech support. -1 for none, 0 for detected.

-2 The current speech mode.

-3 The current gap, in seconds or a negative value if not waiting.

-4 The current speech volume in the range 0 to 100.

-5 The current speech rate in the range -10 to 10.

It is possible for the displayed text to include embedded SAPI XML commands, which will be honoured, but this is unlikely to be useful as they will look somewhat strange when displayed.

See also:
Embedded SAPI XML, `InfoOpen()`, `InfoRun()`, `InfoSettings()`, `Speak()`

Inkey()

This function was provided for compatibility with the MS-DOS version of Spike2. Do not use it in new scripts. It returns the ASCII code for the key pressed, or -1 if no key was pressed. In some cases Spike2 absorbs keystrokes, for example if you are sampling and the current window is the sampling time window all keystrokes are taken as markers.

```
Func Inkey();
```

Returns The key code or -1 if there is no pending key. The codes are:

```
 1-31  Ctrl+ABC...XYZ[\]^_       65-95  ABC...XYZ[\]^_
  32   space                       96    `
33-47  !"#$%&'()*+,-./           97-126 abc...xyz{|}~
```

```
        48-64 0123456789:;<=>?@         127   Rubout
```

See also:
`Interact(), Keypress(), Toolbar(), ToolbarSet()`

Input()

This function reads a number from the user. It opens a window with a message, and displays the initial value of a variable. You can limit the range of the result.

```
Func Input(text$, value{, low{, high{, pre%}}});
```

`text$` A string holding a prompt for the user. If the string includes a vertical bar, the text before the vertical bar is used as the window title.

`value` The initial value to be displayed for editing. If limits are given, and the initial value is outside the limits, it is set to the nearer limit.

`low` An optional low limit for the result. If `low>=high`, the limits are ignored.

`high` An optional high limit for the result.

`pre%` If present, this sets the number of significant figures to use to represent the number in the range 6 (the default) to 15.

Returns The value typed in. The function always returns a value. If an out-of-range value is entered, the function warns the user and a correct value must be given. When parsing the input, leading white space is ignored and the number interpretation stops at the first non-numeric character or the end of the string.

See also:
`DlgReal(), DlgInteger(), DlgFont(), Input$(), Inkey()`

Input$()

This function reads user input into a string variable. It opens a window with a message, and displays a string. You can also limit the range of acceptable characters.

```
Func Input$(text$, edit${, maxSz%{, legal$}});
```

`text$` A string holding a prompt for the user. If the string includes a vertical bar, the text before the vertical bar is used as the window title.

`edit$` The starting value for the text to edit. If `legal$` is supplied, `text$` is filtered by the character list in `legal$`.

`maxSz%` Optional, maximum size of the response string.

`legal$` An string holding acceptable characters. To allow only lower-case English vowels you would use: `"aeiou"`. As it would be tedious and space consuming to list all acceptable characters, you can use character ranges. For example, to accept the upper and lower case English letters plus the digits you could use: `"a-zA-Z0-9"`. A hyphen indicates a range of characters with increasing character codes. To include a hyphen in the list, place it first or last. To allow positive and negative integer numbers you could use: `"-0-9"`. To allow Japanese Hiragana only you could use: `"\u3041-\u3096\u3099-\u309f"`.

If this string is omitted, all characters are acceptable. The order of characters is determined by the Unicode character code-point. The code points start by matching the ASCII sequence (Basic Latin) for characters with code points below 128. Excluding control characters, the sequence is:

```
space !"#$%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
abcdefghijklmnopqrstuvwxyz{|}~
```

Please consult a Unicode table for all other characters.

Returns The result is the edited string. A blank string is a possible result.

See also:
DlgString(), DlgFont(), Input(), Inkey()

InStr()

This function searches for a string within another string. This function is case sensitive. For more complex searches you could consider using InStrRE() which can perform a variety of regular expression searches.

```
Func InStr(text$, find${, index%});
```

text$ The string to be searched.

find$ The string to look for.

index% If present, the start character index into text$ for the search. The first character is index 1.

Returns The index of the first matched character, or 0 if the string is not found.

Unicode

In a Unicode build of Spike2 the underlying characters are stored in a vector of 16-bit values and index% is the index into these values of the start of the search. Most common Unicode characters fit in one 16-bit value, but some require two consecutive values, leading to the possibility that you might attempt to search half an extended character. The rule we apply is that if the start or end of any string section falls between the lead and trail codes of a Unicode surrogate pair, we move the start or end on by one position, to the start of the next character or the end of the string. In this case, the effect is to increase index% by 1.

See also:
Chr$(), DelStr$(), InStrRE(), LCase$(), Left$(), Len(), Mid$(), Print$(), ReadStr(), Replace$(), Right$(), Str$(), UCase$(), Val()

InStrRE()

This function searched for a string within another string using regular expressions. This code is implemented though the regular expression engine that is part of the C++ standard. There are a lot of flag values and error codes that are defined by name in the standard, but not by value. Because of this, we have given them all symbolic names, and the associated values are defined in the script file regex.s2s in the system include folder. Should you need to refer to any of these symbolic names, #include <regex.s2s> at the start of your script. There are 3 command variants:

Set the regular expression

Call this function first to set up for matching. The regular expression and syntax you set here persist until you change it. If you need to search for several different regular expressions you may find the all in one command variant more useful.

```
Func InStrRE(syntx%, re$);
```

syntx% Sets the syntax to use when parsing the regular expression (both the grammar and options). You should only set one of the flags that set the regular expression grammar, or set none for ECMAScript. The call with two arguments sets the default values to use if match% and syntx% are not supplied in the other command variants.

resECMAScript	Use the script grammar as described for ECMAScript. This is the default and is set if you do not specify a grammar.
resBasic	Use the basic POSIX script grammar.
resExtended	Use the extended POSIX script grammar.
resAwk	Use the awk grammar.
resGrep	Use the POSIX grep grammar.
resEgrep	Use the POSIX egrep grammar.
resIcase	Ignore case when matching.
resNosubs	Do not return information about any sub-matches.

resOptimise	Spend more time optimising the search. Maybe useful when searching a long string or if you will apply the same regular expression to multiple strings. This is not guaranteed to have any effect.
resCollate	Make character ranges, for example [a-d], sensitive to the locale. This can be important if your search text is not in English.

re$ The regular expression text to use for the search. This text can contain capture groups.

Returns 0 if the regular expression was compiled without error or a negative error code (see below).

Search text for a match

```
Func InStrRE(text${, index%{, match%{, ms%}}});
```

text$ The text to search. You can move the start point with index%.

index% If present, the start character index into text$ for the search. The first character is index 1. That is, characters before this index are not searched (or even visible to the search engine).

match% A set of flags that can be used to control the matching process. Although these flags ought to have stable values, we cannot guarantee that they will not be changed, so you must use the constants defined in regex.s2s.

remDefault	The default state, 0. You should use this unless you really know you need something else.
remNotBol	Do not take the first character as matching beginning of line (^).
remNotEol	Do not take the end of the string as matching end of line ($).
remNotBow	Do not take the start of the string as matching beginning of word (\b).
remNotEow	Do not take the end of the string as matching the end of a word (\b).
remAny	If more than 1 match is possible, then any match is acceptable.
remNotNull	If text$ is empty, then no match is possible (prevents "x*" matching an empty string, for example).
remContinuous	The search expression must match with the first character to succeed.

ms% An optional time out in milliseconds. If you omit this it defaults to 1000 milliseconds. If you are certain that your regular expression is well-formed and will terminate quickly you can set this to 0, which removes the time out and the search will run faster. **Beware**: it is possible to write expressions that will search forever. If you set ms% to 0 and set such a search, you will not be able to stop it. Spike2 will hang.

Returns 0 if no match was found. If a match is found, the return value is 1 plus the number of capture groups in the regular expression. The return code is negative for an error (see below). If you are timed out, the return code is rerrComplexity.

Get information about the matches and captured groups

You can get the matched text as positions and sizes in the original string. Positions and sizes are very useful if you want to replace the found text.

```
Func InStrRE(pos%[]{, sz%[]});
```

pos%[] This array is filled in with the 1-based start indices into text$ of the match, followed by the start indices of any captured groups. If a capture group is part of an alternative and is not matched, for example "(dog)|(cat)", the non-matching capture group has a start index of 0 and a size of 0. It is not an error for the array to be too small or too large.

sz%[] Optional. If there is a match, this is filled in with the size of the complete match, followed by the sizes of each captured group. It is not an error for the array to be too small or too large.

Returns The number of available matches.

If find$ holds the searched for text, the entire matched text is Mid$(find$, pos%[0], sz%[0]) and the n^{th} capture group is Mid$(find$, pos%[n%], sz%[n%]).

All in one call

This call sets up a search, performs it and reports the start index of any match in a single call. You can follow it with the previous variant to get more detailed match information.

```
Func InStrRE(text$, re${, index%{, match%{, syntax%{, ms%}}}});
```

`text$` The text to search. You can move the start point with `index%`.

`index%` If present, the 1-based start character index into `text$` for the search.

`match%` An optional set of flags that can be used to control the matching process as described above.

`syntx%` Sets the syntax to use when parsing the regular expression (both the grammar and options) as described above. If omitted, the default syntax is set.

`ms%` An optional time out in milliseconds as described above.

Returns The position of the start of the match or 0 if no match or a negative error code. If you need the positions and sizes of matches you can follow this with the **`InStrRE(pos%[]{, sz%[]})`** call.

Error codes

All of the function variants return a positive code for normal operation and a negative code for an error. The `regex.s2s` file holds definitions of the error code constants described below and also `Func regexError$(err%)` that converts a negative error code into a text string. Currently defined error codes are:

`rerrCollate`	Use of an invalid collating element name.
`rerrCtype`	Use of invalid character class name `:lower:` is an example of a valid name.
`rerrEscape`	Use of an invalid escaped character, or a trailing escape.
`rerrBackref`	Invalid back reference, for example `\3` when no such parenthesized group.
`rerrBrack`	Mismatched square brackets [and].
`rerrParen`	Mismatched parentheses (and).
`rerrBrace`	Mismatched braces { and }.
`rerrBadbrace`	The contents of the braces { and } was invalid.
`rerrRange`	There was an invalid character range.
`rerrSpace`	Ran out of memory when converting the regular expression to a state machine.
`rerrBadrepeat`	One of the repeat specifiers (+*? or {...}) did not have a valid expression in front of it.
`rerrComplexity`	The regular expression is too complicated or the search was timed out.
`rerrStack`	Ran out of space when matching the expression.
`rerrParse`	Some other error when parsing the regular expression.
`rerrSyntax`	A syntax error when parsing that is not covered above.

Example

The following very simple example locates a numeric data with the format `dd/mm/yy` in text. We allow the day and month to be specified by 1 or 2 decimal digits and the year by 2 decimal digits. We capture the day, month and year as separate strings:

```
var text$ := "There is a date 1/12/14 in here";
var find$ := "(\\d{1,2})/(\\d{1,2})/(\\d{2,2})";
InStrRE(0, find$);          'Setup the search
var n% := InStrRE(text$);   'Search for a match
PrintLog("%d\n", n%);       'Print the matches
var pos%[4],sz%[4], i%;
InStrRE(pos%, sz%);         'get positions and sizes
for i% := 0 to n%-1 do
    PrintLog("%s\n", Mid$(text$, pos%[i%], sz%[i%]))
    next;
```

Note that you have to double up the `\` characters when they are supplied as part of a literal string (as they are used by the script language as escapes). The output from this script is:

```
4
1/12/14
```

```
1
12
14
```

Grammars

The C++ specified regular expression engine can cope with several different grammars. We suggest that you stick with the ECMAScript grammar as this is pretty much a superset of the available features. However, you may be more familiar with one of the other grammars and prefer to use it. Note that `(...)` is used for capture groups (so you must use `\(` and `\)` to match parentheses) except for the basic and `grep` grammars. The available grammars are (together with links to web-based descriptions that worked in 2024):

`ECMAScript`	Use the script grammar as described for ECMAScript (JavaScript). This has the most supported grammar features of all the supported grammars.
`Basic`	Use the basic POSIX script grammar. This uses `\(...\)` to capture a group.
`Extended`	Use the extended POSIX script grammar.
`Awk`	Use the `awk` grammar.
`Grep`	Use the POSIX `grep` grammar. This is the same as `Basic`, but accepts `\n` as well as `|` for alternation.
`Egrep`	Use the POSIX `egrep` grammar. This is the same as `Extended`, but accepts `\n` as well as `|` for alternation.

We are using the Microsoft supplied regular expression library and you can find their description of the differences between supported grammars at http://msdn.microsoft.com/en-us/library/bb982727.aspx. This reference also has the Microsoft description of regular expressions.

The editor Find dialog can use a simple grammar that is a cut down version of the Basic syntax or the ECMAScript grammar.

Regular expressions

Using regular expressions is a huge topic, well beyond the reach of the Spike2 online help. This section attempts to get you started with the basics of the ECMAScript regular expression grammar. If you need more than this you can look it up online; there are many sites that will tell you more than you ever wanted to know, for example this one. If you are really serious about regular expressions, there are programs that will analyse and debug your regular expressions for you (for example, `RegexBuddy`). The other supported regular expression grammars are basically subsets of this one with some syntax differences.

What does a regular expression do?

Basically, it works through target text and either matches some part (or maybe all) of the text, or it does not. If it matches the text, you have the option of saving sub-matches that were found along the way. You are most likely to use regular expressions for verifying that text matches some particular specification, or when transforming input text. Both the regular expression and the target text are processed from left to right, but there are circumstances where the search will backtrack (which can cause problems). A lot of the skill in writing a good regular expression is to avoid backtracking as much as possible.

Special characters

The characters `^ $ \ . * + ? ( ) [ ] { ` and `}` are special. All other characters just stand for themselves, so a regular expression of `"rat"` will match the same letters in `"scratch"`.

Single character matches

Any character except one of the special characters listed above, matches itself. A dot matches any single character except an end of line character.

Escapes

If you want to match any of the special characters you must put a \ in front of them. Beware: if the regular expression is being provided by a script literal, you will need to double the \ to \\ to get it past the Script compiler. For example if you want to match the parenthesised text in "cat (tiger) and" you would need to use a regular expression of "\\(tiger\\)", which becomes \(tiger\) after being processed by the script compiler.

You can also use escapes for the file format escape characters: form feed \f, new line \n, carriage return \r, tab \t and vertical tab \v. Don't forget to double the \ in literal strings: tab is "\\t", for example. There is also an escape \0 ("\\0" in literal strings) for the character with code 0, which you should not need in Spike2.

Ranges

You can state that instead of matching a single, defined character, you can match any one of a whole range of characters. This is done using square braces holding the list of allowed matching characters. You can also provide a list of non-matching characters:

[0-9a-zABC-**]** Matches the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, the lower case letters z through z, the upper case letters A, B and C and a hyphen. A minus sign (-) at the start or end of the character list stands for itself. Otherwise, it stands for an inclusive character range. The range a-c is the characters a, b, c. The range c-a is an error because c comes after a in the collation order.

[^-1357A-Z**]** Matches any character that is not in the list. So this will not match the minus sign, the odd digits, or any upper-case letter (unless ignore case is set, when it will not match a-z and A-Z.

You could match both "scare" and "score" with "sc[ao]re". You do not need to escape the special characters except] and \ inside the range brackets as there is no possibility of using them for other purposes.

Character classes

You can refer to classes of characters by class names. These mostly use up more space than typing the range equivalent, but are useful in the Unicode context as they cover more than the ASCII characters. A class is specified by [: followed by a name followed by :]. You can use these constructs within a range. The names recognised are:

alnum	Lower case and upper case letters and decimal digits. Underline is considered a letter.
alpha	Lower case and upper case letters
blank	Space or tab
cntrl	The file format escape characters
digits	The decimal digits
d	Same as digits
graph	Lower case and upper case letters, decimal digits and punctuation
lower	Lower case letters
print	Lower case and upper case letters, decimal digits, punctuation and space or tab
punct	Punctuation
space	Space characters (not tab)
s	Same as space
upper	Upper case letters
w	Same as alnum
xdigit	Hexadecimal digits [0-9a-fA-f]

To use these classes, include them within a range or a not-range. For example to include hexadecimal characters and space and the letter z: [[:xdigit:][:space:]z]. To exclude all digits you could use: [^[:d:]]. If you forget the range brackets [:d:] will match the letter d and a colon.

dsw character escapes

These provide a useful shorthand for ranges (but remember that you will have to double the \ in a string literal). You can use these both in a range, where they stand for the equivalent character class, and outside a range, where they are equivalent to a range of the character class.

Escape	Same as	Use
\d	[[:d:]]	Matches decimal digit.
\D	[^[:d]]	Not decimal digit.
\s	[[:s]]	Matches space.
\S	[^[:s]]	Not space.
\w	[[:w]]	lower and upper case letters and decimal digits.
\W	[^[:w]]	Not lower and upper case letters and decimal digits.

Capturing and non-capturing groups and back references

(stuff**)**	This groups the stuff within the parentheses and saves it as capture group *n*, where *n* is the count of the left parentheses from the start of the search. The first group is numbered 1. The group then behaves as a single element. You can then use \1, \2 and so on ("\\1" in a literal string) to refer back to a previously matched group. So the literal string "(cat)\\1" will match "catcat". Note that the Basic and grep grammars treat (and) as normal characters and you must escape them as \(and \) for use as syntax elements.
(?:stuff**)**	This groups the stuff within the parentheses for lexical purposes, but does not capture it.

It is possible to have captured text within a repeated group. In this case, the captured text is the last capture.

Alternation

The vertical bar separates alternatives. Thus "dog|cat" will match either domestic pet. You can use groups to limit the scope of the alternation. "gr(a|e)y" will match "grey" or "gray" and capture the character, "gr(?:a|e)y" will match both and not capture the character.

Beware: If your search text uses | as a separator (for example, see the ScriptBar() command), you can be caught out searching for "match|", expecting this to match the text "match|". In fact, this matches the text "match" *or* an empty string (which matches everything as it has no constraint). In this case you must escape the | so that the regular expression engine sees: "match\|". If you provide this text as a string literal, you must use:

```
const match$ := "match\\|";
```

because the script compiler also uses backslash as the escape character.

Quantifiers (repeats)

A quantifier refers to the regular expression element immediately to its left and specifies how many repeats of it we want to match:

{n}	Matches the previous entity repeated exactly n times (n is composed of decimal digits). For example "x{10}" matches "xxxxxxxxxx".
{min,max}	Matches the previous entry repeated between min and max times (min and max are composed of decimal digits). min can be 0, max must be the same as or more than min.
{min,}	Matches the previous entry repeated at least min times.
*****	Matches the previous entry 0 or more times. Equivalent to {0,}.
+	Matches the previous entry 1 or more times. Equivalent to {1,}.
?	Matches the previous entry 0 or 1 times. Equivalent to {0,1}.

Greedy/non-greedy matching

Normally, a match will consume as much as possible of the text while still matching the data. For example, the expression ".*end" will match all of "it depends on what you spend". This is called *greedy* matching as the match consumes as much as it can.

You can follow any quantifier (except ?) with a ? to convert it from a greedy match to a non-greedy match. With the previous example, ".*?end" will match "it depend".

Greedy matches can be dangerous, particularly using * or {min,}. Consider searching for ".*a.*b". This means "find the last a that is followed by a b". A simple-minded implementation will start by finding the last a, then find the last b after it. If there is no last b it has to find the second last a and try again, then the third last a, and so on. This repeated searching due to a later mismatch is called backtracking. In a bad case, backtracking can cause the search to take so long that it never ends, or it can use up so much memory that the system crashes.

To avoid hanging up the script with a never-ending search, by default, an InStrRE() search will time out after 1 second.

Tip
Whenever you use a greedy match, such as .* or .+, consider if you really intended .*? or .+? for a non-greedy match.

Anchors and boundaries

These items are a little different from the matches we have seen so far as they do not consume characters in the target text, but they state a condition that must be true for a match to continue. You can think of these as asserting that a condition must be true.

^ Assert that the current position is the start of the text or start of a line. "^T" will only match text/lines that begin with a capital T. You can use remNotBol to state that the start of the text is not at the start of a line.

**$** Assert that the current position is the end of the text or a line. "end.$" will match the final "end.", not the one in "spend." in "Do not spend. That is the end." You can use remNotEol to state that the end of the text is not the end of a line.

\b Asserts that the current position is a word boundary. That is, the previous character is not part of the alnum class and the current one is part of the alnum class or vice versa. The start of the text is also considered a word boundary unless you use the remNotBow flag when it is not a word boundary. The end of the text is also considered a word boundary unless you use the remNotEow flag.

\B Asserts that the current position is not a word boundary, that is both the previous and current characters are part of the alnum class, or they are both not part of the class.

(?=expr) Assert that the following text matches the regular expression expr but does not move the search position past expr.

(?!expr) Assert that the following text does not match expr.

Interact()

This function provides a quick and easy way to interact with a user (but also consider the Toolbar() command family). It displays the interact toolbar at the top of the Spike2 window and pauses the script until a button or a key linked to a button is pressed. While the Interact() command runs, Spike2 releases Idle time to the system. Cursors can always be dragged as we assume that they are one of the main ways of interacting with the data. You can limit the user actions when the bar is active. Since version [10.06], we save the view with the input focus at the time of the click and attempt to restore it before returning.

```
Func Interact(msg$, allow% {,help {, lb1$ {,lb2$ {,lb3$...}}}});
```

msg$ A message to display in the message area of the toolbar. The message area competes with the button area. With many buttons, the text may not be visible.

allow% A number that specifies the actions that the user can and cannot take while interacting with Spike2. 0 allows the user to inspect data and position cursors in a single, unmovable window. The codes are shown in both decimal and hexadecimal format. The number is the sum of possible activity codes.

1	0x0001	Can change application
2	0x0002	Can change the current window
4	0x0004	Can move and resize windows
8	0x0008	Can use File menu

16	0x0010	Can use Edit menu
32	0x0020	Can use View menu but not ReRun
64	0x0040	Can use Analysis menu
128	0x0080	Can use Cursor menu and add cursors
256	0x0100	Can use Window menu
512	0x0200	Can use Sample menu
1024	0x0400	No changes to y axis
2048	0x0800	No changes to x axis
4096	0x1000	No horizontal cursor channel change

`help` This is either the number of a help item (CED internal use) or it is a help context string. This is used to set the help information that is presented when the user presses the F1 key. Set 0 to accept the default help. Set a string as displayed in the Help Index to select a help topic, for example `"Cursors"`.

`lb1$` These label strings create buttons, from right to left, in the tool bar. If no labels are given, one label is displayed with the text `"OK"`. The maximum number of buttons is 17. Buttons can be linked to the keyboard using `&` and by adding a vertical bar followed by a virtual key code to the end of the label. You can also set a tool tip. The format is `"Label|code|tip"`. See the documentation for `label$` in the `ToolbarSet()` command for details.

Returns The number of the button that was pressed. Buttons are numbered in order, so `lb1$` is button 1, `lb2$` is button 2 and so on.

If a toolbar created by `Toolbar()` is present, it is hidden during the `Interact()` command and restored after `Interact()` returns.

See also:
`Toolbar(), ToolbarSet()`

K

`Keypress()`

Keypress()

This function returns 1 if the `Inkey()` function would return a character, or 0 if it would not. This function and `Inkey()` are provided for compatibility with the MS-DOS version of Spike2 and are not recommended for new scripts.

```
Func Keypress();
```

Returns 1 if a key is ready to read, 0 if there is no key.

See also:
`Inkey(), Interact(), Toolbar(), ToolbarSet()`

L

`LastTime()`
`LCase$()`
`Left$()`
`Len()`
`LinPred()`
`Ln()`
`LnGamma()`
`Log()`
`LogHandle()`

LastTime()

This function finds the first item on a channel before a time. If a marker filter is applied to the channel, only data in the filter is visible. This function is for time views only.

```
Func LastTime(chan%, time{,&value|code%[]{,data[]|data%[]|&data$}});
```

`chan%` The channel number in the view to use.

`time` The time to search before. Items at the time are ignored. To start a backward search that guarantees to iterate through all items, start at `Maxtime(chan%)+1`.

`value` Optional: for waveform channels it returns the waveform value. For event level channels, it is returned 0 if the transition is low to high, and 1 if the transition is high to low. If there is no event it returns the level at `time`; 0 for low, 1 for high.

`code%` This optional parameter is only used if the channel is a marker type (marker, RealMark, TextMark, WaveMark). From version [10.12] the first 4 elements (if they exist) are set to the marker codes associated with the item. The fifth element (`code%[4]`), if present, is set to the extra 32-bit value associated with each marker code in a 64-bit `.smrx` file.

Prior to version [10.12] this is an array with at least four elements that is filled in with the marker codes.

`data` Filled with data from RealMark and WaveMark channels. If there is insufficient data to fill it, unused entries are unchanged. An integer array can be used with WaveMark data to collect a copy of the 16-bit data that holds the waveform. If WaveMark data has multiple traces, use `data[points%][traces%]` to get real data and `data%[points%][traces%]` to get the integer data. If you use a vector with multiple traces, the first trace is returned unless the `MarkTrace()` command has been used to set a different trace as the default.

`data$` A string returned holding the text from a TextMark channel.

Returns The function returns either the time of the next item, or -1 if there are no more items to be found or a negative error code.

This function can be slow if run on a channel with a marker filter set and with the majority of events filtered out as Spike2 has to search for events that are in the filter. If you are using this to get Marker and extended Marker data, you could consider using `ChanData()`.

See also:
`ChanData(), MaxTime(), MarkTrace(), NextTime()`

LCase$()

This function converts a string into lower case.

```
Func LCase$(text$);
```

`text$` The string to convert.

Returns A lower cased version of the original string.

See also:
`Asc(), Chr$(), DelStr$(), InStr(), Left$(), Len(), Mid$(), Print$(), ReadStr(), Replace$(), Reverse$(), Right$(), Str$(), UCase$(), Val()`

Left$()

This function returns the first `n` characters of a string.

```
Func Left$(text$, n%);
```

`text$` A string of text.

`n%` The maximum number of characters to return.

Returns The first `n%` characters, or all the string if it is less than `n%` characters long.

Unicode

In a Unicode build of Spike2 the underlying characters are stored in a vector of 16-bit values and `n%` is the number of these values to use. Most common Unicode characters fit in one 16-bit value, but some require two consecutive values, leading to the possibility that you might attempt to return half an extended character. The rule we apply is that if the start or end of any string section falls between the lead and trail codes of a Unicode surrogate pair, we move the start or end on by one position, which must put it at the start of a character. In the case of `Left$()`, we increase `n%` by 1.

See also:

`Asc(), Chr$(), DelStr$(), InStr(), LCase$(), Len(), Mid$(), Print$(), ReadStr(), Replace$(), Right$(), Str$(), UCase$(), Val()`

Len()

This function returns the length of a string or the size of a one dimensional array or the size of the nth dimension of an array.

```
Func Len(text$);
Func Len(const arr[]);
Func Len(const nd[]{[]...}{, n%}) added at version [8.08], extended at [10.12]
```

`text$` The text string.

`arr[]` A one dimensional array. It is an error to pass in a two dimensional array.

`nd[]` A one or more dimensional array.

`n%` The 0-based index of the dimension to return the size of. From [10.12] set -1 for total elements (product of all dimensions).

Returns The length of the string or the array, as an integer.

You can find out the size of each dimension of a 2 dimensional array as follows:

```
proc something(arr[][])   'function passed a 2-d array
var n% := Len(arr[][0]);  'get size of first dimension
var m% := Len(arr[0][]);  'get size of second dimension
```

The largest multi-dimensional array we support is 5-d, so here is the code for finding the sizes of all the dimensions of a 5-d array.

```
proc arrayMax(a[][][][][])
var d1% := Len(a[][0][0][0][0]);
var d2% := Len(a[0][][0][0][0]);
var d3% := Len(a[0][0][][0][0]);
var d4% := Len(a[0][0][0][][0]);
var d5% := Len(a[0][0][0][0][]);
```

From version 7.11c you can pass an array with a 0 length to `Len()`, and the result is 0 rather than a fatal script error.

From version 8.08 you can use the third variant, so the previous example becomes:

```
proc arrayMax(a[][][][][])
var d1% := Len(a, 0);
var d2% := Len(a, 1);
var d3% := Len(a, 2);
var d4% := Len(a, 3);
var d5% := Len(a, 4);
```

This also allows you to detect a zero length dimension as you are not allowed to create a sub-array of a zero-sized array.

```
var a[2][0];
if Len(a[0][]) = 0 then 'Syntax error as a has zero size
```

but

```
if Len(a, 1) = 0 then 'OK as not creating a sub-array
```

From [10.12, 9.15, 8.23] you can set `n%` to -1 to get the total element count of the array. This allows you to detect that any dimension is zero with:

```
if Len(a, -1) = 0 then ... 'detect any zero dimension
```

Unicode

Versions of Spike2 from [8.03] onwards are Unicode enabled. The underlying storage used for strings is a vector of 16-bit values. Most characters fit in one 16-bit value, but some require two. The `Len()` function returns the number of 16-bit values required to hold the string, not the number of characters.

If you are running the Unicode version of Spike2, you may wish to know the length of a string if it were written as UTF-8 (for example for use by `BWriteSize()`). The following function will return the length of the UTF-8 equivalent of a string `str$` in bytes.

```
func LenUTF8%(str$)
var i% := 1, bytes%, c%; ' bytes% initialised to 0
var n% := Len(str$);
while i% <= n% do
    c% := Asc(Mid$(str$, i%));  'get code of next char or surrogate pair
    docase
    case c% < 0x80 then bytes% += 1;     'ASCII code
    case c% < 0x800 then bytes% += 2;    'needs 2 bytes
    case c% < 0x10000 then bytes% += 3; 'needs 3 bytes
    else bytes% += 4; i% +=1;            'Surrogate pair, skip second part
    endcase;
    i% += 1;
wend;
return bytes%;
end;
```

See also:

`Asc(), Chr$(), DelStr$(), InStr(), LCase$(), Left$(), Mid$(), Print$(), Replace$(), Reverse$(), Right$(), Str$(), UCase$(), Val()`

LinPred()

Linear prediction can be used to predict future (or past) data values based on a sequence of data values on the assumption that the data is statistically stationary. It can also be used to estimate power spectra using the Maximum Entropy or All Poles method. The command generates a set of coefficients that when applied to the previous `m` points, generate the next predicted point. Some of the explanation for this command relies on technical knowledge; see the references below for more information. If you want to use linear prediction to fix artifacts in a time View waveform or a Result view channel, see the `ChanLinPred()` command. The `LinPred` command has the following variants:

```
Func LinPred(const data[], mMax%{, limit{, out[]{, dir%{, stab%}}}});
Func LinPred(0, out[]{, data[]});     'Predict forward
Func LinPred(1, out[]{, data[]});     'Predict backwards
Func LinPred(2, stab%);               'Check stability
Func LinPred(3, coef[]);              'Get the coefficients
Func LinPred(4, refl[]);              'Get reflection coefficients
Func LinPred(5, power[], frLo, frHi); 'Get estimated power spectrum
Func LinPred(6, poles[][]{, fr[]});   'get poles and frequencies
```

`data` An array holding the data to be used to form the linear prediction coefficients or to be used to initialise a prediction based on existing coefficients.

`mMax%` The maximum order of the prediction, which is the number of previous points to use to predict each future point. The actual number may be less than this, depending on the value of the `limit` argument. You will generally want to use the smallest value of `mMax%` that you can; values in the range 5-50 are common, but this does depend on the data and the application. `mMax%` must be less than the number of data points and is generally much less. From [10.07, 9.12] we limit `mMax%` to 1024 (this is higher than you are likely to need); previously it was an error to exceed 1024.

`limit` The algorithm to calculate the poles is an iterative procedure. The command holds an array of residual values that is initialised to the raw data. Each iteration subtracts data from the residual based on a normalised autocorrelation of the residual (in the range -1 to 1) at increasing lags, aiming to reduce the residual array to a list of zeros. We track a number that models the significance of the remaining data. This number is 1.0 at the start, and is multiplied by ($1-ac^2$) at each iteration, where `ac` is the autocorrelation, so the value decreases at each iteration unless `ac` is 0. If this value becomes less than `limit`, the iteration will stop. So set limit to 0 for no early stopping, setting a small value above `1.0e-31` may stop it early. The actual value of `m` used is returned by the first command variant.

`out` An array of output values predicted by the command. When predicting forwards, the first item in the `out` array is the first predicted point. When predicting backwards, the data values are written into the `out` array so that the first data item in the array is the last predicted point.

`dir%` Optional argument to the set up command that sets the prediction direction. Set 0 or omit for a forward prediction, 1 for reverse.

`stab%` The linear prediction coefficients that are generated form a characteristic polynomial, and the roots of this polynomial are the positions of the poles in the z-plane. For the resulting filter to be stable (have an output that does not increase exponentially), the poles must lie inside (corresponding to decaying sinusoids) or on (corresponding to constant amplitude sinusoids) the unit circle. This algorithm should not produce poles outside the unit circle unless there are numerical accuracy problems (usually when `mMax%` is large). You can use this argument to check the stability of the result and to adjust the pole positions and recompute the coefficients by setting the values:

- 0 Do nothing to the poles.
- 1 If a pole lies outside the unit circle, reflect it across the unit circle so that a growing sinusoid becomes a decaying sinusoid.
- 2 If a pole lies outside the unit circle, move it onto the unit circle, corresponding to a constant sinusoid.
- 3 Move all poles onto the unit circle. This produces an output that neither grows nor decays with time. However, this is very prone to numerical stability problems and is likely to be interesting only when `mMax%` is small.

`coef` An array to be filled in with coefficients. The array can be any size, but only the points corresponding with coefficients will be set.

`refl` An array to be filled with reflection coefficients. These values represent the proportion of the residual that was removed at each iteration. These are the values that are modified by the Levinson recursion to form the coefficients.

`power` An array to be filled in with estimated power spectrum components.

`frLo` A value in the range -0.5 to 0.5, being the fraction of the sampling frequency that the first bin of `power` will hold the estimated power spectrum for.

`frHi` A value in the range -0.5 to 0.5, being the fraction of the sampling frequency that the last bin of `power` will hold the estimated power spectrum for.

`poles` A matrix with the second dimension of size at least 2. `poles[n][0]` is returned holding the real component of the n^{th} pole, and `poles[n][1]` holds the imaginary component.

`fr` An array to be filled in with the frequencies that correspond to the pole positions, in the range -0.5 to 0.5 (fraction of the sampling rate).

Setup

```
Func LinPred(data[], mMax%{, limit{, out[]{, dir%{, stab%}}}});
```

This command must be used before any other as it calculates the initial set of coefficients. The command returns the number of coefficients that have been generated (this will be the same as `mMax%` if `limit` is 0). The command can be used to generate predicted data, or to set up the system for further `LinPred()` commands. The following example takes 1000 data points from near the start of a waveform channel, predicts the next 100 points and writes them to a memory channel.

```
const chan%:=1;                              'a waveform channel
var data[1000], out[100], sTime, bsz := BinSize(chan%);
var n% := ChanData(chan%, data, 100*bsz, MaxTime(), sTime);
var m% := LinPred(data, 20, 0, out, 0, 0); 'predict forwards
```

```
var mc% := MemChan(9, 0, bsz);              'new RealWave channel
MemSetItem(mc%, 0, sTime + n%*bsz, out[]); 'save the data
ChanShow(mc%);                              'display it
```

The predicted data will not be the same as the actual data that follows the first 1000 points unless the first 1100 points are composed of the sum of constant amplitude sinusoids. The command forms a mathematical model of the data held in the `data[]` array based on the assumption that the spectral components are not changing with time (the signal is stationary) and that the data is modelled by a set of resonances. Unless you set `stab%` to 3, the result will usually decay with time.

Predict forwards and backwards

```
Func LinPred(0, out[]{, data[]});     'Predict forward
Func LinPred(1, out[]{, data[]});     'Predict backwards
```

These two command versions fill the `out[]` array with data predicted by the coefficients established by the Setup version of the command. The set up command `dir%` argument will have set up the command to generate output that joins up with the original data array, or with any output if the set up command generated output. You can choose to continue generating output in the same direction, in which case you must NOT supply the `data[]` array, or you can supply a `data[]` array of at least the size of the number of coefficients to reset the prediction and you can then run forwards or backwards. By supplying a `data[]` array, you are not recalculating the coefficients, these remain unchanged; you are reloading the data points that are used with the coefficients to predict values. If you have `m` coefficients, when going forwards, the last m data points of `data[]` are used; when going backwards, the first `m` data points. For example, if we wanted to extend the previous example to predict the 100 data points that might have led up to the original 1000 points we could add the lines:

```
var back[100];
LinPred(1, back, data);                    ' predict back from start
MemSetItem(mc%, 0, sTime-100*bsz, back[]); ' save the data
```

If we had just used `LinPred(1, back);` this would have caused an error as the previous use of the command was to go forwards. These command variants return 0. See here for an example of use for prediction.

Check stability

```
Func LinPred(2, stab%);                'Check stability
```

This command variant is used to check that the poles of the characteristic polynomial lie within the unit circle. You should use it after **`LinPred(data[], mMax%{, limit});`** and before generating any output. The command returns the distance of the pole furthest from the origin of the z-plane. This should be less than or equal to 1.0 for a stable set of coefficients.

Stable, in this context, means that the predicted data does not grow exponentially. The algorithms we use should generate stable solutions, but poles can be generated outside the unit circle due to loss of numerical precision in the calculations. The `stab%` argument can be used to adjust the pole positions, as described above. If you are using this command to replace a short stretch of damaged data (like fixing a scratch in a record), you may want to predict both forwards and backwards across the damaged data, then mix the two predictions together. If the data used to generate the coefficients is not stationary, it will decay across the gap, in which case using `stab%` set to 3 will generate a result that maintains its amplitude (but is very prone to instability), which may be what you require.

Get coefficients

```
Func LinPred(3, coef[]);               'Get the coefficients
```

This function returns the number of coefficients and returns them in the `coef[]` array. Note that the first coefficient is the one that is multiplied with the most recent data point (when going forward), that is the coefficients run backwards compared to the data. Given an array `x[]` of data and coefficient `coef[]`, both of length `m%` points, the next forward and backward predictions are given by:

```
var i%, fwd:=0, rev:=0;
for i% := 0 to m%-1 do
   fwd += x[m%-i%-1] * coef[i%];
   rev += x[i%] * coef[i%];
next;
```

It is usually much simpler to use the forward and backward prediction versions of the command to do this. If you cannot do this, for example when you need to process several waveforms simultaneously, save the

coefficients for each waveform (and reverse the order if predicting forwards), then you can use `ArrDot()` to predict each new point.

Get reflection coefficients

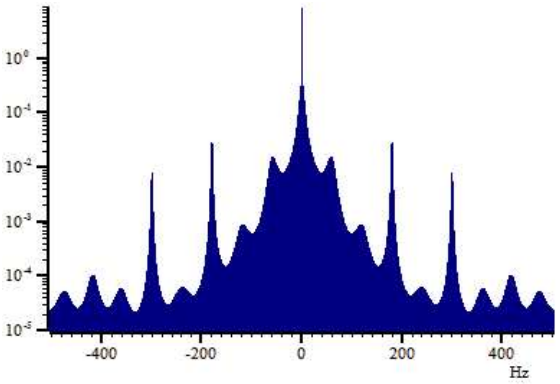
```
Func LinPred(4, refl[]);                    'Get reflection coefficients
```

This function returns the number of reflection values that are available (the same value as the number of coefficients) and sets `refl` to the results of the auto-correlations done at each lag as the original data was analysed. The results are normalized such that the results lie in the range -1 to 1. The first value is the auto-correlation at a lag of 1 point, the next is for a lag of 2 points, and so on. The term reflection comes from the use of this technique in seismology.

Power spectrum

```
Func LinPred(5, power[], frLo, frHi); 'Get estimated power spectrum
```

The variant generates an estimate of the power spectrum based on the representation of the original as the sum of a set of resonances. You should leave the `stab%` argument at 0 when using the command for this purpose. This method is particularly effective if you know that the data contains a small number of constant frequency sinusoids and can allow you to separate peaks that would merge into a single peak using the FFT. However, if your spectrum is not representable as the sum of a set of resonances, the result may be misleading unless `mMax%` is large enough so that the spectrum can be approximated. Unlike the FFT, where the resolution of the result depends on the number of points in the transform, here the resolution of a particular peak is determined by the number of bins and the frequency range that you set. Make sure you set the bin width small enough so you do not miss a very narrow resonance. The frequencies are defined in terms of a fraction of the sampling rates from -0.5 to 0.5 (but note that -0.5 is the same frequency as 0.5); you will find that the power at frequency *f* is equal to the power at frequency *-f*.



Typical Power spectrum

The result is scaled so that the integral of the power from -0.5 up to (but not including) 0.5 is equal to the mean square of the values in the original `data[]` array used in the set up call. For example, we could extend the previous examples to display the power with:

```
const bins% := 10001;                       'Bins in spectrum
var power[bins%];                           'Array to hold the power
LinPred(5, power, -0.5, 0.5);               'NB: -0.5 is same as 0.5
var rv% := SetResult(bins%, 1/(bsz*(bins%-1)), -0.5/bsz, "Power", "Hz");
ArrConst([], power);Optimise();WindowVisible(1); ' display the result
var MeanSq := ArrDot(data,data)/Len(data);  'Mean square of data
var MemSum := ArrSum(power[:bins%-1])/(bins%-1); 'Integral of power
Message("Mean sumSq = %g, SumPower = %g", MeanSq, MemSum);
```

The example image shows the power spectrum obtained by data with 60 Hz mains interference, leading to peaks at odd multiples of 60. If you try this with data constructed from pure sinusoids you will find that `MemSum < MeanSq`. The reason is that the purer the sinusoid, the higher and narrower the peak in the power spectrum; simple schemes for adding up equal width bins will not give an accurate result in this case. In the limiting case where the roots of the characteristic polynomial lie on the unit circle, the resonance has infinite amplitude, but zero width.

You might be puzzled about how this method can get much better frequency resolution than the FFT. The basic reason is that this method makes the assumption that the data continues in *a sensible way* outside the range of input points so as to preserve the auto-correlation between the data values. The FFT assumes that the data repeats exactly outside the initial range of data, leading to a limit on the frequency resolution.

Get Poles and frequencies

```
Func LinPred(6, poles[][]{, fr[]});     'get poles and frequencies
```

In the z-plane, the coefficients are represented by a set of poles. You can use this variant to read back the positions of the poles as complex numbers. The poles are sorted in order by their real components, so the pole

pairs for each imaginary root should be adjacent. You can also read back the frequencies (as a fraction of the sample rate in the range -0.5 to 0.5) at which the poles are located.

Predicting data across a gap (example)

If you are predicting data across a gap (a very common use of this function), you will find that your choice of `mMax%` is critical. If your data is periodic, for example a blood pressure signal and there are around *p* points in the period, then you will find that setting `mMax%` to *p* (the exact value is not critical) will usually work. If your data is the sum of a small number of sinusoids, you will find that a value of around twice the number of sinusoids may work. In other cases, if you are predicting across a gap of *g* points, then setting `mMax%` to *g* (again the exact value is not critical) may work.

It is a good idea to predict forwards and backwards across the gap, then merge the two predictions using some sort of weighting function that starts with all forwards prediction at the start and all backwards prediction at the end. There is example code to do this, here.

Remember that the mathematics makes the assumption that the data used for the prediction is stationary; that is that the statistical properties of the data (power spectra) remain constant with time. In many cases this will not be true; this leads to predicted data that decays away to the mean value.

The stability parameter should usually be 0. If you find that your predicted values are growing exponentially, then you can try setting the `stab%` argument to 1 or 2, which should stop this. However, the more normal problem is for the data to decay away. Setting `stab%` to 3 is a rather drastic act as it moves all the sinusoidal resonances that are used to model the power spectrum of the data onto the unit circle (converts them all into resonances with infinite Q and zero width). This may be useful if you have a small value of `mMax%` and you know that the data consists of constant sinusoids.

References

Claerbout, Jon F. (1976). "Chapter 7 - Waveform Applications of Least-Squares." Fundamentals of Geophysical Data Processing. Palo Alto: Blackwell Scientific Publications. This has an explanation of the John P Burg algorithm that we implement.

Levinson recursion is a method for inverting a Toeplitz matrix in $O(n^2)$ time, taking advantage of the symmetry of the matrix, and improving on the $O(n^3)$ time of a general inversion (*n* is the order of the prediction). Symmetrical Toeplitz matrices come about as a natural consequence of the linear prediction equations. However, implementing the obvious equations often results in unsatisfactory solutions, and the Burg algorithm is a better approach that incorporates the iterative idea of the Levinson recursion, but calculates the autocorrelation in a different way that leads to stable solutions.

There is an overview of linear prediction and the Maximum Entropy (All Poles) method of power spectrum estimation in Numerical Recipes, The Art of Scientific Computing, by Press, Flannery, Teukolsky and Vettering.

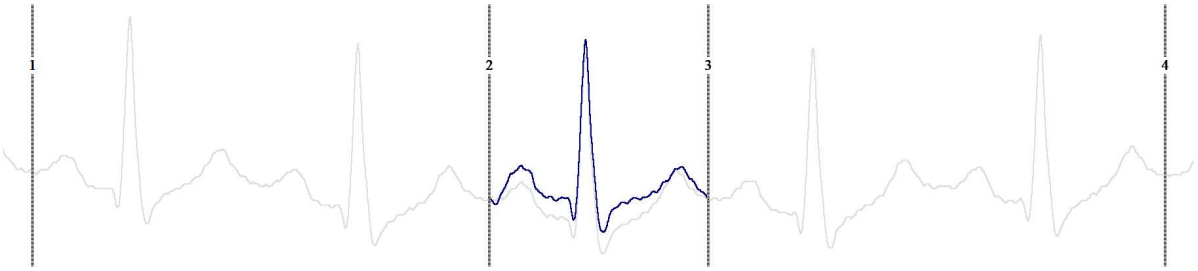
See also:

`ChanLinPred()`

Predict across a gap example

This example demonstrates the use of the `LinPred()` script command. From Spike2 [10.08], the `ChanLinPred()` command provides the functionality of this example and can be applied to a Result view channel in addition to Time view waveforms.

The following example code is intended to predict missing or corrupt data across a 'short' gap in a situation where the missing data can be assumed to continue is the same manner as the preceding data and lead in a natural way to the following data. We do it by predicting forwards from the preceding data and backwards from the following data. The example below shows the code in action:



The grey waveform is the original. Cursors 2-3 mark an area that we will assume is missing or corrupted. Cursors 1-2 and 3-4 mark the areas to use for forward prediction and backwards prediction. The dark trace is the predicted waveform. It is a mix of the forward predicted area from the Cursor 1-2 area and the backwards predicted area from the Cursor 3-4 area. You can see that it is not the same as the waveform that actually happened; it is a best guess at the waveform that continues the *before* and *after* waveform.

The gap will typically be significantly shorter than the pre and post data used for prediction. As we limit the number of points used for prediction to 1024 (currently), attempting to predict across gaps or using cycle lengths greater than this is likely to yield unreliable results. If the data has a cyclical pattern, you will want to use at least 1 cycle before and after. Using more cycles gives estimates that are more representative of the 'average' cycle. In this case we have predicted a whole cycle, but there is no need to do this and we would expect that in most cases we are filling in a few points due to brief artefacts or data drop out.

If you use this with data that is not obviously cyclical, the generated points will continue the sequence of data points in a manner that generates data auto-correlations that match those of the data used for prediction.

```
'Interpolate a waveform channel over a gap
'ch%    The channel (Waveform or RealWave) to interpolate
't1-t2  The time range before the gap
't3-t4  The time range after the gap
'Return The memory channel holding the result or -1 if impossible
Func Interpolate%(ch%, t1, t2, t3, t4)
const kind% := ChanKind(ch%);
if (kind% <> 1) && (kind%<>9) then return -1 endif;
const bSize := BinSize(ch%);    'Channel sample interval
if (t1<0) || (t2<t1) || (t3<=t2) || (t4<t3) then return -1 endif;
var n1% := (t2-t1) / bSize + 1; ' maximum first set of points
var n2% := (t4-t3) / bSize + 1; ' maximum second set of points
var y1[n1%], ta, tb;          ' space for data and start times
n1% := ChanData(ch%, y1, t1, t2, ta);
if n1% <= 0 then return -1 endif;
var y2[n2%];
n2% := ChanData(ch%, y2, t3, t4, tb);
if n2% <= 0 then return -1 endif;

var nFill% := (tb - ta + bSize/2) / bSize - n1%; 'fill points
if nFill% <= 0 then return -1 endif;
var yFwd[nFill%], yBack[nFill%], i%;   'Space for the fill points
LinPred(y1[:n1%], n1%, 0, yFwd,  0, 0); 'Predict forwards
LinPred(y2[:n2%], n2%, 0, yBack, 1, 0); 'Predict backwards

var yFill[nFill%];                     'array for result
Mix(yFill, yFwd, yBack, nFill%);       'Blend two predictions
var t := ta + n1% * bSize;             'First predicted time
var mc% := OutChan%(yFill, t, ch%);    'Display as memory channel
return mc%;
end;

'Linear mix of two waves
'out   The output mix of the two arrays
'fwd   Forward array, 100% at front, 0% at back
'bck   Backwards array 0% at front, 100% at back
Proc Mix(out[], fwd[], bck[], n%)
var i%;
for i% := 0 to n%-1 do
    out[i%] := (fwd[i%] * (n% - 1 - i%)) / (n%-1) + (bck[i%] * i%) / (n% - 1);
    next;
end;
```

```
'Create matching memory channel to hold data
'y      Array of values to add to a memory channel
't      Time of the first point
'ch%    The channel to match (type, sample rate and scaling)
Func OutChan%(y[], t, ch%)
var mc% := MemChan(0, ch%);     'Copy the channel type
MemSetItem(mc%, 0, t, y);
ChanShow(mc%);
return mc%;
end
```

This code generates a memory channel that holds the data predicted to fill the gap between times `t2` and `t3`. The data output to a memory channel is done by the `OutChan%()` code.

The following example of use expects you have opened a data file with data in channel 1 and have set cursors 1-4:

```
const ch% := 1;                 'The channel (assumed a waveform)
CursorRenumber();               ' We need cursors 1-4 in order
if (Cursor(1) < 0) || (Cursor(2) < 0) || (Cursor(3) < 0) || (Cursor(4) < 0) then
    Message("4 cursors needed"); halt
endif
var mc% := Interpolate%(ch%, Cursor(1), Cursor(2), Cursor(3), Cursor(4));
```

If you want to use this code to fix a file with no gaps but with artefacts, you could replace the `OutChan%()` function call with:

```
ChanWriteWave(ch%, y[], t);   'Overwrite the channel with predicted data
```

However, it is usually a good idea to visualise the data first with a memory channel to be sure that the replacement is reasonable.

Listener()

This command returns the number of Listener devices currently attached to Spike2. Currently, the only listener devices are generated by the s2video multimedia recording application. You could use this command after launching the `s2video` program using `ProgRun()` to discover if it is ready to record data. This command was added at version [10.03].

```
Func Listener()
```

Returns The count of listener devices that are registered with Spike2 and that will respond to sampling.

Ln()

This function calculates the natural logarithm (inverse of `Exp()`) of an expression, or replaces the elements of an array with their natural logarithms.

```
Func Ln(x|x[]{[]...});
```

`x` A real number or a real array. Zero or negative numbers cause the script to halt with an error unless the argument is an array, when an error code is returned.

Returns When used with an array, it returns 0 if all was well, or a negative error code. When used with an expression, it returns the natural logarithm of the argument.

See also:
`Abs()`, `ATan()`, `Cos()`, `Exp()`, `Frac()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`

LnGamma()

This returns the natural logarithm of the gamma function *Gamma(x)*, which exists for real values of $x > 0.0$ and for negative non-integral values. For integral values of x less than or equal to 0, *Gamma(x)* is infinite. *Gamma(n+1)* is the same as *n*! (*n* factorial) for integral values of $n > 0$. However, it increases very rapidly with *x*, reaching floating-point infinity when *x* is 172.62. To avoid this problem, the script returns the natural logarithm of the gamma function. The definition of the gamma function is:

$$\Gamma(a) = \int_0^\infty t^{a-1}e^{-t}dt$$

If you require factorials to computer binomial coefficients you should use `BinomialC()` instead.

```
Func LnGamma(a);
```

`a` A positive value. The script stops with a fatal error if this is negative.

Returns The natural logarithm of the Gamma function of `a`.

See also:
`BinomialC(), GammaP()`

Log()

Takes the logarithm to the base 10 of the argument.

```
Func Log(x|x[]{[]...});
```

`x` A real number or a real array. Zero or negative numbers cause the script to halt with an error unless the argument is an array, when an error code is returned.

Returns With an array, this returns 0 if all was well or a negative error code. With an expression, this returns the logarithm of the number to the base 10.

See also:
`Abs(), ATan(), Cos(), Exp(), Frac(), Ln(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan(), Trunc()`

LogHandle()

This returns the view handle of the Log view (which always exists). You need this if you are to size the log window, or make it the current or front window, or to use the `EditSelectAll();EditClear();` commands to clear it. The Log view is created by the application and is the destination for `PrintLog()`. The Log view is a simple text window. You can hide it by clicking in the go away box.

```
Func LogHandle();
```

See also:
`Print(), PrintLog(), View()`

M

`MarkEdit()`
`MarkInfo()`
`MarkMask()`
`MarkSet()`
`MarkShow()`
`MarkTrace()`
`MATDet()`
`MATInv()`

MatLab Script Support

`MatLabOpen()`
`MatLabClose()`
`MatLabPut()`

MatLabGet()
MatLabEval()
MatLabShow()
MATMul()
MATSolve()
MATTrace()
MATTrans()
Max()
Maxtime()
MeasureChan()
MeasureToChan()
MeasureToXY()
MeasureX()
MeasureY()
MemChan()
MemDeleteItem()
MemDeleteTime()
MemGetItem()
MemImport()
MemSave()
MemSetItem()
Message()
Mid$()
Min()
MinMax()
MM (Multimedia) commands
MMAudio()
MMFrame()
MMImage()
MMOffset()
MMOpen()
MMPosition()
MMRate()
MMVideo()
Modified()
MousePointer()
MoveBy()
MoveTo()

MarkEdit()

This changes the data stored in a Marker, RealMark, TextMark or WaveMark item at a particular time. You can get the data using LastTime() and NextTime(). From Spike2 version [10.12] you can use this on a Level event channel in a 64-bit .smrx file to change the marker codes (Level event data is stored in the same format as a Marker channel.)

```
Func MarkEdit(chan%, time, const code%[]{, data$|const data[]|const data%
[]});
```

chan% The marker, TextMark, WaveMark or RealMark channel to edit.

time The time of the marker (must match exactly).

code% From version [10.12] this is an array of up to 4 marker codes (bottom 8 bits used) to replace markers in the channel. If the array is longer than 4 items, the lower 32-bits of the fifth item (code%[4]) set the extra 32-bit value associated with each marker. If the array is shorter than 4 elements, missing elements are not changed. Level event channels in .smrx files use the first marker code to set the level; 0 is low, 1 (or not zero) is high.

Prior to version [10.12], this was an array of 4 marker codes.

data The data to replace the marker data. If you use integer data with a WaveMark, the bottom 16 bits of each integer replace the data. To update a WaveMark with multiple traces use a two-dimensional array, for example with 32 points and 4 traces use var data[32][4]; to declare the array. If you use a one-dimensional array, the first trace is modified.

Returns 0 if a marker was edited, or -1 if no marker exists at time.

See also:
LastTime(), MarkInfo(), MarkMask(), MarkSet(), MarkShow(), NextTime()

MarkInfo()

This function is used to get information on the extended marker types (TextMark, WaveMark and RealMark).

```
Func MarkInfo(chan% {,&pre% {,&trace%}});
```

`chan%` The channel to get the information from.

`pre%` If present, returned as the number of pre-peak points for WaveMark data.

`trace%` If present, returned as the number of traces (electrodes) in the WaveMark data.

Returns For WaveMark data it returns the number of waveform points, for TextMark data it returns the maximum string length and for RealMark data it returns the number of reals attached to each marker. For all other channel types it returns 0.

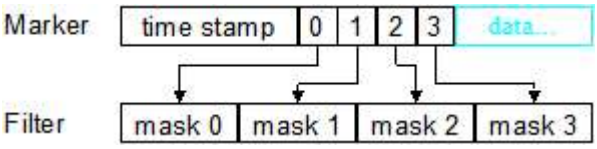
See also:
`LastTime(), MarkEdit(), MarkMask(), MarkSet(), MarkShow(), NextTime()`

MarkMask()

This function sets the mask for a marker, WaveMark, TextMark or RealMark channel. Each data item in one of these channels has four marker codes. Each code has a value from 0 to 255. Each data channel (and duplicated channel) has its own marker filter that determines the visible data items. A marker filter has four masks, one for each of the four marker codes. For each mask, you can specify which codes are wanted. There are two marker filter modes. In the diagrams, a marker data item is represented as a time stamp, four marker codes and data values that depend on the marker type.

Mode 0 (AND)

A marker data item is allowed through the mask if each of the four codes in the data item is present in the corresponding mask. We think of this as *and* mode because for the filter to pass the marker, marker code 0 must be in mask 0 *and* marker code 1 in mask 1 *and* marker code 2 in mask 2 *and* marker code 3 in mask 3. In this mode, masks 1, 2 and 3 are usually set to accept all codes and masking is used for layer 0.



Mode 1 (OR)

A marker data item is allowed through the mask if any of the four marker codes is present in mask 0. A code 00 is only accepted for the first of the four marker codes. Masks 1 to 3 are ignored. We think of this as *or* mode because marker code 0 *or* 1 *or* 2 *or* 3 must be present in mask 0 for the filter to pass the marker for display or analysis. This mode can be used with spike shape data (WaveMark) where two spikes have collided and the marker represents a spike of two different templates.

There are four command variants, the first reports if the mask is active, the second sets the filter codes, the third sets the filter mode and the fourth allows you to read and set the entire mask:

```
Func MarkMask(chan%);
Func MarkMask(chan%, layer%, set%, code%|code$ {,code%|code$...});
Func MarkMask(chan%, mode%);
Func MarkMask(chan%, mask%[]{[]}{, write%});
Func MarkMask(chan%, &fSpc$|fSpc${, write%});
```

`chan%` The channel number to work on. If unsuitable, `MarkMask()` returns a negative error code.

`layer%` The layer of the mask in the range 0 to 3 or -1 for all layers.

`set%` 1 to include codes in the mask, 0 to exclude codes or -1 to invert the mask. Inverting a mask changes all included codes to excluded and vice versa.

`code%` A number in the range 0 to 255 setting a code to include or exclude. You can specify more than one code at a time. -1 is also allowed, meaning all codes.

`code$` Each character in the string is converted to its ASCII value, and used as a code.

`mode%` The variant with two arguments returns the current mode of the marker filter and optionally sets the mode of matching as 0 or 1 or -1 for no change.

`mask%` This is an array that is usually declared as `mask%[256][4]` or `mask%[256]` that maps onto the mask.

`write%` If this optional argument is omitted or zero, read mask data. If it is 1, write data to the channel filter mask.

`fSpc$` From [10.06], this argument is used to write or read the marker filter as text. When reading, this argument must be a string variable.

Returns A negative error code or 0 except: the first variant returns 1 if the current mask is active, 0 if not active. The last variant returns 0 for success and 1 if you attempt to write with a bad text specification.

Report active state

```
Func MarkMask(chan%);
```

The first variant was added at Spike2 version 8.01 and reports if the current mask is active. An active mask is one that could prevent some items being displayed. However, being active does NOT mean that items are filtered out as this depends on the data in the channel.

Set filter codes

```
Func MarkMask(chan%, layer%, set%, code%|code$ {,code%|code$...});
```

The second variant allows you to set codes, either individually or by layer. A common requirement is to allow all markers to be used. This is achieved by:

```
MarkMask(chan%, -1, 1, -1);  'Set all layers to 1 for all codes
```

To fill or empty or invert a complete layer use:

```
MarkMask(chan%, layer%, set%, -1); 'Apply set% to entire layer
```

This example sets the keyboard marker channel mask (channel 31) to show only markers 0 and 1 (start and stop recording markers) and key presses for A, B, C and D:

```
MarkMask(31, 0);                  'set mode 0
MarkMask(31,-1, 1,-1);            'include everything (reset)
MarkMask(31, 0, 0, -1);           'exclude everything in layer 0
MarkMask(31, 0, 1, 0, 1, "ABCD"); 'include the codes we want
```

You can use this command together with `ChanDuplicate()` to split a marker channel into several channels based on marker codes.

Set or get the filter mode

```
Func MarkMask(chan%, mode%);
```

This sets the filter mode to AND (`mode%`=0) or OR (`mode%`=1) and returns the original mode. To get the mode without a change use `MarkMask(chan%, -1)`.

Set or get the mask

```
Func MarkMask(chan%, mask%[]{[]}{, write%});
```

This command copies an array of mask values to the filter or reads the filter into an array. `mask%` is array that is usually declared as `mask%[256][4]` or `mask%[256]` that maps onto the mask. The second dimension sets the layer. A one-dimensional array maps onto layer 0. When reading, elements are set to 0 or 1. When writing, zero element clear and non-zero elements set the corresponding item in the mask. If the array size does not match 256 by 4, data is transferred between array items that map onto the mask.

Set or get the mask as a text specification [10.06]

```
Func MarkMask(chan%, &fSpc$|fSpc${, write%});
```

This command either sets or gets the filter marker using a text specification. If `write%` is present and non-zero (1), the string is parsed, and if legal, it is used to set the channel marker filter. Otherwise, the `fSpc$` argument (which must be a variable), is updated with a text representation of the marker filter. When generating a string from a filter, we generate lists of codes in ascending order, which may not match how they were set. For example, a list set as `"[a-z,A-Z,0-9]"` will be returned as `"[0-9,A-Z,a-z]"`. Further, we use a simple-minded algorithm to generate the list, so a list specified as `"[!a-z,e]"` is returned as `"![a-d,f-z]"`, which is equivalent, but not as concise.

See also:
`ChanDuplicate(), LastTime(), MarkEdit(), MarkInfo(), MarkSet(), MarkShow(), NextTime()`

MarkSet()

This sets the marker codes of a marker, WaveMark, TextMark or RealMark channel in a time range. If a marker filter mask is set, only data that is passed by the mask is changed.

```
Func MarkSet(chan%, sT, eT, const code%[]);
Func MarkSet(chan%, sT, eT, c0%{, c1%{, c2%{, c3%}}});
```

`chan%` The channel to process.

`sT,eT` The time range to process.

`code%` An array of 4 integers holding the new marker codes in the range 0 to 255 or -1 for a code that is unchanged.

`c0-c3%` One to four marker codes as an alternative to `code%[]`. Use values 0 to 255 to change a code and -1 (or omit the value) to leave a code unchanged.

Returns The number of markers that were changed.

See also:
`LastTime(), MarkEdit(), MarkMask(), MarkShow(), NextTime()`

MarkShow()

This sets which of the four marker codes to display for one or more marker channels and the format used when displaying the marker code in a time view. Setting the default draw mode with `DrawMode(chan%,0)` resets the displayed code and drawing mode to 0. This is equivalent to the Marker code and Hex only fields in the channel Draw Mode dialog.

```
Func MarkShow(cSpc{, code%{, mode%}});
```

`cSpc` A channel specifier for the channels to modify.

`code%` This optional argument sets which of the 4 marker codes to display in the range 0 to 3, or -1 or -2 for no change (-2 returns the current drawing mode).

`mode%` Omit or set to -1 for no change. Otherwise set 1 to display all codes as two hexadecimal digits, or 0 to show codes `0x20` to `0x7e` as the character equivalent.

Returns The original marker code to display (0-3) or, if `code%` is -2, the drawing mode.

See also:
`DrawMode(),LastTime(),MarkEdit(),MarkMask(),NextTime(), MarkTrace()`

MarkTrace()

This command is used with WaveMark channels with multiple traces to set or get the trace to use when the channel needs to be treated as a waveform. The trace is saved together with any Marker filter set for the channel. If you duplicate a channel, each duplicate can have a different trace set. The selected trace is saved in the resources associated with the data file.

```
Func MarkTrace(chan%{, trace%});
```

`chan%` The channel number of a WaveMark channel.

`trace%` If present it sets the trace to use in the range 0 to the number of traces-1. You can also set the trace to -1 to clear any trace set (in which case the first trace is used in situations where a single trace is wanted). If you set a trace that is outside the allowed range for the channel, the trace is set to -1.

Return The trace set before any change made by this call. This is -1 if the trace is cleared (this is the default).

See also:

`MarkMask()`, `MarkInfo()`, `MarkShow()`, Marker Filter, `DrawMode(-17)`

MATDet()

This calculates the determinant of a matrix (a two dimensional array).

```
Func MATDet(const mat[][]);
```

`mat` A two dimensional array with the same number of rows and columns.

Returns The determinant of `mat` or 0.0 if the matrix is singular.

See also:

`ArrAdd()`, `MATInv()`, `MATMul()`, `MATTrans()`

MATInv()

This inverts a square matrix (a two dimensional square array) and optionally returns the determinant. If you need to solve the matrix equation **A x** = **y** for **x**, given **A** and **y** see the `MATSolve()` command rather than inverting the matrix and multiplying by **y**.

```
Func MATInv(inv[][] {,const src[][]{, &det}});
```

`inv` A two dimensional array to hold the result. If `src` is omitted, `inv` is replaced by its own inverse. The number of rows and columns of `inv` must be the same.

`src` If present, the matrix to invert. The numbers of rows and columns of this two dimensional array must be at least as large as `inv`.

`det` If present, returned holding the determinant of the inverted matrix.

Returns 0 if all was OK, -1 if the matrix was singular or very close to singular.

Timings

The time taken to invert a matrix depends on the speed of the computer, but this table, measured on an Intel Core(TM) i7-4770K running at 3.5 GHz should give some idea of what you can expect. In the table, **n** is the number of rows and columns; the matrix size is **n** x **n**. Note the use of s, ms and us. In theory, for the algorithm we use, times should be proportional to $\mathbf{n}^3$, but in fact for small **n** the times increase at less than this (while the overheads dominate the times and the data fits in the cache) and for large **n** they increase faster as the size of the data exceeds the cache.

n	time	n	time	n	time
2	3 us	32	137 us	512	155 ms
4	6 us	64	680 us	1024	2.00 s
8	9 us	128	4.56 ms	2048	25.0 s
16	28 us	256	25.3 ms	4096	242 s

See also:

`MATDet()`, `MATMul()`, `MATSolve()`

MatLab Script Support

If you have MATLAB™ installed on your computer and selected the MATLAB script option when installing Spike2, the `MatLabXxx()` script commands let you start a MATLAB process to use as a computational engine. This process can have a visible window allowing some user interaction; it is not a full MATLAB workspace and is separate from any normally opened MATLAB workspaces that you use. You can transfer script variables and arrays (but not arrays of strings) to it, command it to process your data, then move results back into the script language.

If you need access to the full MATLAB environment, consider exporting your data as a `.mat` file. It is also possible to read (and write) both old format `.smr` and new format `.smrx` files from MATLAB using the SON64 interface for MATLAB that is downloadable from our web site.

Example

This example can be used to check that the script support is working. It illustrates use of all the functions and most of the data transfer options.

```
var f%, a, b, c, m%, n%;

f% := MatLabOpen(); ' Open MATLAB
if (f% < 0) then Message("Open failed %d", f%); halt endif;

f% := MatLabShow(); ' Use MatLabShow
if (f% <> 0) then Message("Show initial state %d", f%) endif;
f% := MatLabShow(1);
if (f% <> 0) then Message("Show 1: state %d", f%) endif;
f% := MatLabShow(0);
if (f% <> 1) then Message("Show 0: state %d", f%) endif;
f% := MatLabShow();
if (f% <> 0) then Message("Show final state %d", f%) endif;
MatLabShow(1);

' Test put and get of a simple variable
a := 2.5;
f% := MatLabPut("var_a", a);
if (f% <> 0) then Message("MatLabPut var_a result %d", f%) endif;
f% := MatLabGet("var_a", c);
if (f% <> 0) then Message("MatLabGet var_a result %d", f%) endif;
if (c <> 2.5) then Message("get value after put 2.5 is %g", c) endif;

' Test put of variable, eval then get the result
b := 1.5;
f% := MatLabPut("var_b", b);
if (f% <> 0) then Message("MatLabPut var_b result %d", f%) endif;
f% := MatLabEval("var_b = var_b + var_a");
if (f% <> 0) then Message("MatLabEval result %d", f%) endif;
f% := MatLabGet("var_b", c);
if (f% <> 0) then Message("MatLabGet result %d", f%) endif;
if (c <> 4) then Message("MatLabGet of eval(1.5 + 2.5): %g", c) endif;

' Test put and get of 1D arrays.
var ar1%[4],ar2d%[4][1],ar2dt%[1][4],ar2%[4],ar3%[6]; ' Test with 1D arrays
ar1%[0] := 12; ar1%[1] := 11; ar1%[2] := 120; ar1%[3] := 1200;
f% := MatLabPut("tar",ar1%[]);      ' Trivial put & get
if (f% <> 0) then Message("Put ar1 result %d", f%) endif;
f% := MatLabGet("tar",ar2%[]);
if (f% <> 4) then Message("Get ar2 result %d", f%) endif;
Compare1D("transfer 1D", ar1%, ar2%);

f% := MatLabGet("tar", ar2d%); ' Now get into nx1 array
if (f% <> 1) then Message("Get ar2d result %d, should be 1", f%) endif;
Compare1D("Transfer to nx1", ar1%, ar2d%[][0]);

var r$;
```

```
MatLabEval("tart = tar'", r$);          ' Next get the transposed data
f% := MatLabGet("tart",ar2%[]);
if (f% <> 4) then Message("Get tart to ar2 result %d", f%) endif;
Compare1D("1D transposed", ar1%, ar2%);

f% := MatLabGet("tart",ar2dt%[][]);         ' Now get into 1xn array
if (f% <> 4) then Message("Get ar2dt result %d", f%) endif;
Compare1D("Transfer to 1xn", ar1%, ar2dt%[0][]);
f% := MatLabGet("tart",ar3%[]);         ' get into larger 1D array
if (f% <> 4) then Message("Get tart to ar3 result %d", f%) endif;
Compare1D("transfer 1D to larger", ar3%[:4], ar2%);

var stri$;                      ' Test with a string variable
f% := MatLabPut("string","abc");
if (f% <> 0) then Message("Put string result %d", f%) endif;
f% := MatLabGet("string",stri$);
if (f% <> 0) then Message("Get string result %d", f%) endif;
if (stri$ <> "abc") then Message("sent abc, got %s", stri$) endif;

' Now tests on nxn arrays. We check put, simple get, get into
' array with last dimension bigger and get after transpose
var arr[3][3];
arr[0][0] := 1; arr[0][1] := 2; arr[0][2] := 3;
arr[1][0] := 4; arr[1][1] := 5; arr[1][2] := 6;
arr[2][0] := 7; arr[2][1] := 8; arr[2][2] := 9;
f% := MatLabPut("var_arr", arr);    ' Copyto MATLAB
if (f% <> 0) then Message("MatLabPut arr result %d", f%) endif;
var arrg[3][3];                     ' Simple get test
f% := MatLabGet("var_arr", arrg);
if (f% <> 3) then Message("Get arrg result %d", f%) endif;
Compare2D("Get 3x3 to 3x3", arr, arrg);

var arrb[3][6];                     ' Get to larger array test
f% := MatLabGet("var_arr", arrb);
if (f% <> 3) then Message("Get arrb result %d", f%) endif;
Compare2D("Get 3x3 to 3x6", arr, arrb);

MatLabEval("var_arrt = var_arr'", r$);          ' Now get after transpose
f% := MatLabGet("var_arrt", arrg);
if (f% <> 3) then Message("Get arrg result %d", f%) endif;
MatTrans(arr);                                  ' Transpose data to match
Compare2D("3x3 transpose", arr, arrg);          ' Check

MatLabClose();
halt;

Proc Compare1D(test$, a%[], b%[])
var bad% := 0, m%, n1%;
n1% := Len(a%);
for m% := 0 to n1%-1 do
    if (a%[m%] <> b%[m%]) then
        PrintLog("%s: 1D mismatch at %d, got %g, expected %d\n",
                 test$, m%, b%[m%], a%[m%]);
        bad% += 1;
    endif;
next;
if (bad% > 0) then Message("%s: %d bad values", test$, bad%) endif;
end;

Proc Compare2D(test$, a[][], b[][])
var m%, n%, n1%,n2%, bad% := 0;
n1% := Len(a[][0]); n2% := Len(a[0][]);
for m% := 0 to n2%-1 do
    for n% := 0 to n1%-1 do
        if (a[n%][m%] <> b[n%][m%]) then
            PrintLog("%s Mismatch at %d %d, got %g, expected %g\n",
                     test$, n%, m%, b[n%][m%], a[n%][m%]);
```

```
            bad% += 1;
        endif;
    next;
next;
if (bad% > 0) then Message("%s: %d bad values", test$, bad%) endif;
end;
```

See also:
`MatLabOpen()`, `MatLabClose()`, `MatLabPut()`, `MatLabGet()`, `MatLabEval()`, `MatLabShow()`, connection problems

MatLabOpen()

This opens a connection to an invisible MATLAB command window. Other `MatLabXxx()` commands fail if there is no connection. Making a connection takes a detectable time; it is inadvisable to open and close the connection many times. This command will fail with error -1518 if MATLAB support is not installed (see below).

```
Func MatLabOpen({mode%});
```

`mode%` If `mode%` is absent or zero, the connection is shared, allowing other applications to use the same command window. Set `mode%` to 1 for unshared, for exclusive use by the script.

Returns 0 if the connection was opened successfully, or a negative error code if the function failed. If the function fails, there may be additional information about the problem in the Log view.

The most likely error code is -1518, "The file does not exist". This occurs for one of three reasons:

1. The MATLAB support is not installed. The Log view message includes: "(is MatLab support installed?)" . Reinstall Spike2 (if in Custom mode, select MATLAB support).
2. The MATLAB support is damaged. The Log view message includes: "Problem loading libeng.dll". Reinstall Spike2 and try again.
3. MATLAB is not available. The Log view message includes: "Failed to access MatLab, use Matlab /regserver command?". Spike2 MATLAB support is installed and working, but the attempt to start the MATLAB engine failed. The most likely explanation is that MATLAB is not installed. This MathWorks web site link may help.

There is more information on connection problems if you follow the link, below.

See also:
`MatLabClose()`, `MatLabPut()`, `MatLabGet()`, `MatLabEval()`, `MatLabShow()`, script example, connection problems

MatLabClose()

This function closes a connection that was created using `MatLabOpen()`.

```
Proc MatLabClose();
```

See also:
`MatLabOpen()`, `MatLabPut()`, `MatLabGet()`, `MatLabEval()`, `MatLabShow()`, script example

MatLabPut()

This function takes a script variable and puts it into the connected MATLAB command window.

```
Func MatLabPut(name$, v%|v|v$|const v%[]{[]...}|const v[]{[]...}{, as%});
```

`name$` the name of a MATLAB variable to create or overwrite. Names may use only alphanumeric characters plus underscore '_' and must start with an alphabetic character.

`v` An expression or variable holding data to move to MATLAB. The created variable type is set by the type of `v` and the `as%` argument. All types except string arrays are supported. You may pass `const` arrays.

`as%` Optional. Sets the type of the MATLAB variable. If `v` is a real value or array, `as%` can be 4 or 8 to specify single- or double-precision real data. If `v` is an integer value or array, `as%` can be 1, 2, 4 or 8 for 1, 2, 4 and 8-byte signed integer data and -1, -2, -4 and -8 for the corresponding unsigned integer data. For 1, 2 and 4-byte integer data only the lower 1, 2 or 4 bytes are used. If `as%` is omitted, it is taken as 8 for a real value and 4 for an integer value.

For string data, a string variable is created in the workspace and `as%` is ignored.

For integer and real data, either a single variable or a vector is created in the MATLAB workspace. If an array is used then the dimensions of the variable created in the workspace match the script array: a 1-dimensional script array length n creates an n x 1 vector, a 2 dimensional script array `arr[m][n]` creates an `m` x `n` matrix and so forth.

Returns 0 if the data was successfully transferred and -1 if it was not.

See also:
`MatLabOpen(), MatLabClose(), MatLabGet(), MatLabEval(), MatLabShow()`, script example

MatLabGet()

This function copies a variable from the MATLAB command window into a script variable.

```
Func MatLabGet(name$, &v%|&v|&v$|v%[]{[]...]|v[]{[]...});
```

`name$` the name of the MATLAB variable to copy. If the variable does not exist the function returns -1 and script execution continues. MATLAB variables that are structures, cell arrays, functions or logical (`BOOLEAN`) data are not supported. Of the remaining data types (character and numeric values), the allowed types vary with the type of script language variable supplied.

`v$` For a string variable, the MATLAB variable must be a simple string (a 1-dimensional array of characters). String arrays are not supported.

`v% v` For non-array integer and real variables, the MATLAB variable must be a numeric value. Integer and floating-point data are automatically converted, floating point values put into an integer variable are truncated.

`v[]` For real and integer script arrays, the MATLAB variable must match the number of dimensions and type (integer or real). All dimension sizes except that last must also match. The last dimension size must be less than or equal to the size of the last script dimension. We make an exception for a script vector which matches both `n x 1` and `1 x n` MATLAB variables. MATLAB floating-point types are converted to script reals and MATLAB integer types are converted to script integers. The return value is the number of items in the last dimension of the script array that was filled.

Returns -1 if the variable was not found in MATLAB or a type match error, otherwise it returns 0 for a simple variable or the size of the last dimension of the script array that was filled.

Command changes

At Spike2 revision 7.09, the command was changed to be more tolerant when returning arrays. The table shows which Spike2 variables are compatible with which MATLAB variables as long as `n<=m`.

Spike2	MATLAB
`s1[m]`	`m1(n), m1(n,1), m1(1,n)`
`s2[a][m]`	`m2(a,n)`
`s3[a][b][m]`	`m3(a,b,n)`
`s4[a][b][c][m]`	`m4(a,b,c,n)`
`s5[a][b][c][d][m]`	`m5(a,b,c,d,n)`

The return value was also changed. An error is still flagged by a -1 return, but success is no longer indicated by a return value of 0. The return value is now 0 for simple variables and n (as in the table) for vectors and multi-dimensional arrays. If you have a script that used to work before version 7.09, look for code like:

```
if (MatLabGet(...) = 0 then
```

and change = 0 to >= 0 (so it works with old and new versions of the command).

See also:
MatLabOpen(), MatLabClose(), MatLabPut(), MatLabEval(), MatLabShow(), script example

MatLabEval()

Requests the MATLAB command window to execute an arbitrary command and gets the result as a string. Typical use: set data with MatLabPut(), process with MatLabEval(), get the result with MatLabGet().

```
Func MatLabEval(cmd$ {,&resp$});
```

cmd$ This is the command string that is sent to the command window, for example "x = a + b".

resp$ Optional. If present, it is set to the command window response. For example, it is set to an error message if the command fails. We limit the response length to 511 characters.

Returns 0 if cmd$ was successfully passed to the command window and -1 if it was not.

See also:
MatLabOpen(), MatLabClose(), MatLabGet(), MatLabPut(), MatLabShow(), script example

MatLabShow()

This command retrieves the visible or hidden status of the MATLAB command window opened by MatLabOpen() and optionally changes it.

```
Func MatLabShow({show%});
```

show% Optional. If provided, sets the new visible state. Set 1 to show the window and 0 to hide it.

Returns the command window visibility at the time the call was made.

See also:
MatLabOpen(), MatLabClose(), MatLabGet(), MatLabPut(), MatLabEval(), script example

Matlab problems

There can be problems setting up the MATLAB connection.

If you upgrade your copy of MATLAB you must follow these instructions again. The path to MATLAB will change.

Get the correct MATLAB directory into the Windows PATH

The Windows PATH is a list of directories that are searched to find the MATLAB DLLs (Dynamic Load Libraries) needed for the connection. If the directory holding the required DLLs is not in the path then they will not be found and opening the connection will fail, normally with error code 126. This is by far the most common issue, as modern MATLAB installers do not set the path up in the way that we need.

To find the directory where your copy of the MATLAB DLLs are stored, search your hard disk for libeng.dll; it will normally be somewhere like C:\Program Files\MATLAB\R2016a\bin\win64. Ignore any copies of this file in the directories where Spike2 or Signal is installed. This directory must be in the path; often you will find that only the directory one level up (C:\Program Files\MATLAB\R2016a\bin) is there (do NOT remove this entry).

WARNING! Incorrect changes to the Windows path can damage your system and, at worst, stop it working altogether. DO NOT make any changes unless you are sure that you understand these instructions and can avoid mistakes. If you are not sure, get an experienced user or administrator to help you.

To examine and change the Windows path (this is based on a Win10 system, others will be similar):

- Open the system dialog to Edit the Environment variables. In Windows 10 type *Environment* into the desktop search box and follow the link to *Edit the system Environment variables*. In earlier version of Windows:
  1. Open the system properties by right-clicking on This PC, either as a desktop shortcut or where it appears on the left-hand section of *File Explorer*. Select *Properties* in the menu that appears.
  2. Select *Advanced system settings* from the options on the left.
- In the *System Properties* dialog, at the bottom of the *Advanced* tab, click the *Environment Variables* button.
- Find and select `Path` in the lower *System variables* section and click the *Edit* button to open the *Edit environment variable* window.
- If `C:\Program Files\MATLAB\R2016a\bin\win64` (or wherever) directory where the MATLAB DLLs are located is not in the list you must add it:
  1. Click *New* and enter the complete path to the MATLAB DLLs, press `Enter` to complete the entry.
  2. Use the *Move Up* button to move the new entry up to immediately after the existing entry for MATLAB. Press *OK* to accept the changes.
- On older systems, the dialog that shows the individual path entries is not there, instead you get a much simpler edit variable window with the entire path shown as one line of text, with semicolons separating the individual entries. Editing the path with this dialog is a bit more complicated, but the idea is much the same - search through the text for the main MATLAB directory and, if the directory we want is not there, add it after the main MATLAB directory.

If anything goes wrong while you are adding the new entry press *Cancel* to exit from the *Edit environment variable* window without any changes being made and start the process again.

A rare pathing problem

MATLAB uses a very large number of libraries in DLL files, several of which are third-party libraries from other sources. If another application is installed which uses the same DLLs but installs different versions from MATLAB, and the path to these other DLLs appears in the Windows path above the paths to the MATLAB directories, then these other DLLs will be loaded instead of the MATLAB versions, which can cause MATLAB to fail.

The solution to this problem is to move both MATLAB directories up to the top of the Windows path, which will ensure that the MATLAB DLLs are found first and loaded. This situation is quite rare, so we would recommend only trying this fix after all other attempts have failed.

Register the MATLAB server

In order to make the connection to MATLAB, MATLAB must be registered as a suitable server. This is normally done as part of the MATLAB installation process but it can fail, or the registration can be lost, causing the connection to fail with error code 0 or 3. To fix this we need to re-register MATLAB:

- Open a command prompt window. In Windows 10:
  1. Type *Command* into the desktop search box
  2. Right-click on the offered *Command prompt* and then *Run as administrator*. If the administrator option is not present, just Run it, but registration may not work.
- In older system use *Start* menu button->*Windows system* (or *Accessories*)->*Command prompt*.
- Type the command `matlab /regserver` and press `Enter`.

There may be more up to date information in this MathWorks web site link.

Possible effects on MatLab file export

Various users have reported that exporting data to MATLAB files works normally but fails after the `MatLabOpen()` script command has been used. This appears to be caused by DLL version incompatibilities between the MATLAB DLLs installed with Spike2 for the purposes of file export (which should be available even if MATLAB is not installed) and the DLLs installed with MATLAB (which are loaded by `MatLabOpen()`). It appears that the problem can be fixed, at least in some cases, by hiding the MATLAB DLLs installed with Spike2 so that file export uses the DLLs installed with MATLAB.

The DLLs in question are all in the Export folder inside the Spike2 installation folder, they are:

boost_date_time-vc100-mt-1_49.dll, boost_filesystem-vc100-mt-1_49.dll, boost_log-vc100-mt-1_49.dll, boost_regex-vc100-mt-1_49.dll, boost_signals-vc100-mt-1_49.dll, boost_system-vc100-mt-1_49.dll, boost_thread-vc100-mt-1_49.dll, hdf5dll.dll, icudt49.dll, icuin49.dll, icuio49.dll, icuuc49.dll, libeng.dll, libexpat.dll, libmat.dll, libmwfl.dll, libmwi18n.dll, libmwMATLAB_res.dll, libmwresource_core.dll, libmx.dll, libut.dll, msvcp100.dll, msvcr100.dll, tbb.dll, tbbmalloc.dll and zlib1.dll.

The best way to hide these DLLs is to put them, but not the matexp.sxl, msvcp100.dll and msvcr100.dll files that are also in the export directory, into a separate sub-directory inside the Export directory. That way you will be able to get the files back easily if necessary. If this solves your problem you will have to repeat this each time you update Spike2.

MATMul()

This function multiplies matrices and/or vectors. In matrix terms, this evaluates A = BC where A is an m rows by n columns matrix, B is an m by p matrix and C is a p by n matrix. Vectors of length v are treated as a v by 1 matrix.

```
Proc MATMul(a[]{[]}, const b[]{[]}, const c[]{[]});
```

a A m by n matrix of real values or a vector of length m (n is 1) to hold the result.

b A m by p matrix or a vector of length m (p is 1).

c A p by n matrix or a vector of length p (n must be 1).

If you pass any of a, b or c as a vector, they are treated as a n by 1 matrix, where n is the length of the vector. Use the trans() operator to convert a vector to a 1 by n matrix.

See also:
trans() operator, ArrMul(), MATInv()

MATSolve()

This function solves the matrix equation A x = y for x, given A and y. Both x and y are vectors of length n and A is an n by n matrix. The solution is done by LU decomposition.

```
Func MATSolve(x[], const a[][], const y[]);
```

x A one dimensional real array of length n to hold the result.

a A two dimensional (n by n) array of real values holding the matrix.

y A one dimensional real array of length n.

Returns The functions returns 0 if all is OK or -1 if a is a singular matrix.

See also:
ArrMul(), MATInv()

MATTrace()

This evaluates the trace of a matrix, that is the sum of the diagonal of the matrix. If the matrix is not square, for example arr[i][j] and the smaller dimension is of size n, the trace is of the arr[:n][:n].

```
Func MATTrace(const arr[][])
```

arr A matrix (two-dimensional array) that is either square, or that is treated as square using the smaller dimension. The array can be either real or integer. The result is always a real value.

Returns The trace of the matrix, that is the sum of the diagonal elements.

This is equivalent to ArrSum(diag(arr));

MATTrans()

This transposes a matrix (a two dimensional array), swapping the rows and columns. This procedure physically moves the data, unlike the `trans()` or ` operator, which remaps the matrix without moving any data. It is usually much more efficient to use `trans()`.

```
Proc MATTrans(mat[][]{, const src[][]});
```

`mat` A `m` by `n` matrix returned holding the transpose of `src`. If `src` is omitted, `m` must be equal to `n` and the rows and columns of `mat` are swapped in place.

`src` Optional, a `n` by `m` matrix to transpose.

See also:
`trans()` operator, `ArrAdd(), MATMul()`

Max()

This function returns the index of the maximum value in an array, or the maximum of several real and/or integer variables or sets the elements of an array to the maximum of itself and the corresponding elements of a second array. The three variants are:

Find maximum of a list of values

```
Func Max(val1 {,val2 {,val3...}});
```

`varN` The values to compare. The values are treated as real (even if they are integer variables).

Returns The maximum value. Note that if you use this to find the maximum of integers greater than 9007199254740992, the result may be less than you anticipate as doubles have a resolution of some 53 bits whereas integers have a resolution of 64 bits.

Find index of maximum value

Search all elements of an array for the maximum value.

```
Func Max(const arr[]);
```

`arr` A real or integer array.

Returns The index of the maximum value. If there are multiple indices with the same value, this finds the first.

Set maximum of corresponding array elements

This variant was added at Spike2 version [10.17].

```
Func Max(arr[]{[]...}, const src[]{[]...});
```

`arr` A real or integer array with 1 to 5 dimensions. If any of the corresponding elements of the `src` array are larger, the `arr` value is replaced. If `arr` is integer and `src` is real, the values are limited to integer range. If `arr` is real and `src` is integer, the values are limited by the resolution of real values.

`src` A real or integer array with the same number of dimensions as `arr`.

Returns 0

See also:
`Abs(), ATan(), Clamp(), Cos(), Exp(), Frac(), Ln(), Log(), Min(), MinMax(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan(), Trunc(), XYRange()`

MaxTime()

In a time view, this returns the maximum time in seconds in the file or in a channel, or the current sample time during sampling, or re-evaluates the maximum time in the data file on disk. In a result view, it returns the number of bins.

```
Func MaxTime({chan%});
```

`chan%` Optional channel number for time views, ignored in result views. If present, the function gets the maximum time in the channel ignoring any marker filter. If there is no data in the channel, the return value is -1. WaveMark channels return the time of the start of the last item.

Spike2 data files store the maximum time on any channel in the data file in the data file header. If you delete a disk channel that holds the maximum time in the file, the file still remembers the original maximum time. From Spike2 version [11.00] you can set `chan%` to -1 to scan all the channels in the file to find the maximum time and set this in the data file header. This rescan does not include memory channels or virtual channels, only channels on disk.

Returns The requested information. The value returned is negative if the channel does not exist. If there is no data in a time view or the channel is empty, or the view is the wrong type, the return value is -1. When called with no channel for a sampling time view that has not started to sample, the return value is once sampling clock tick less than 0.

To find the time of the last item in the marker filter on a channel with a marker filter set:

```
time := LastTime(chan%, MaxTime() + BinSize());
```

With a marker filter set, this will search the data backwards from the end to find a marker in the filter. `MaxTime()` returns the last valid time in the file on all channels. The `+ BinSize()` is because `LastTime()` finds data before the search time.

Dialog expression `MaxTime()` *anomaly*
In a time view with WaveMark data, the dialog expression version of `MaxTime()` increases the time of the last item in the file to include the maximum width of any WaveMark channel. So far, this has not caused a problem for anyone; we have no plans to change it.

Time views and sampling

From version [10.19] onwards, when a time view is created for sampling until sampling stops, the `MaxTime()` return value is the current sampling time, which will be 0.0 until sampling starts. When sampling stops it will be the maximum time of any item in the file.

Prior to [10.19], when the view was created, if you used a script to write data to a channel, the `MaxTime()` return value was the last time of any written data item until sampling started, at which point it became the time into sampling.

If you supply a channel number, the return value is the time of the last data item in the channel, regardless of sampling time.

The returned value is not modified by the `ViewExtraTime()` command.

See also:

`Len(), LastTime(), NextTime(), ViewExtraTime(), Seconds()`

MeasureChan()

This function adds or changes a measurement channel in an XY view created with `MeasureToXY()` with the XY as the current view, or adds or changes a measurement item in a time view RealMark channel created by `MeasureToChan()` with the time view as the current view. The measurement settings are the most recent defined by `MeasureX()` and `MeasureY()`. The XY view or time view must be the current view. When used with a time view, the channel targeted is the one most recently created by `MeasureToChan()`. This command implements some of the functionality of the Measure to XY and Measure to channel dialogs.

```
Func MeasureChan(chan%, name${, pts%});
```

`chan%` In an XY view, this is 0 to add a new channel or the number of an existing channel to change settings. `MeasureToXY()` creates an XY view with one channel, so you will usually call this function with `chan%` set to 1. You can have up to 32 measurement channels in the XY view.

In a Time view, this is 0 to add a new y measurement to the current `MeasureToChan()` process, or is the 1-based index of an already added measurement. Measurement 1 is the one created by `MeasureToChan()`. The first one you add is number 2. You cannot modify a measurement that has not yet been defined. This argument is usually 0 (to add a measurement).

`name$` This sets the name of the channel and can be up to 9 characters long.

`pts%` Ignored in time views. Sets the maximum number of XY view points for this channel. If omitted or set to zero then all points are used. When a points limit is set and more points are added, the oldest points are deleted.

Returns The channel number in the XY view or data index in the RealMark channel that these `settings` were applied to or a negative error code.

Changes at version [10.03]

Prior to 10.03, this command was not used with RealMark channels. When used with a RealMark channel the maximum number of measurement items is set by the `type%` argument of the `MeasureToChan()` command. If you use this command to define more measurements than set, only the set number are saved.

I generated the following code by:

1. Turning recording on in the Script menu.
2. Opening the `demo.smr` file.
3. Using the Analysis->Measurements->Data channel... command to open the Measure to channel dialog
4. Creating three measurements in the dialog and clicking New.
5. Processing all the data
6. Turning recording off

```
var v12%; 'View created by FileOpen()
var ch1%; 'Memory channel created by MeasureToChan
v12%:=FileOpen("C:\\Users\\Greg\\Documents\\Demo.smr",0, 3);
Window(3.46715, 0.824742,54.9531, 50.567);
MeasureX(102, 0, "Cursor(0)", "0"); ' The time of the item
MeasureY(100, 1, "Cursor(0)", "0"); ' First measurement
ch1% := MeasureToChan(0, "Measure 1", -3, 4, 1, 0, "0.0", 0, 1, "", 0);
MeasureY(100, 1, "Cursor(0)+1", "0"); ' Second measurement
MeasureChan(0,"Measure 2");           ' Add new measurement
MeasureY(3, 1, "Cursor(0)-0.1", "Cursor(0)+0.1"); 'Third measurement
MeasureChan(0,"Measure 3");           ' Add new measurement
Process(0.0, 120.921, 0, 1, ch1%);    ' Process the data
```

This generates a new RealMark memory channel with 3 items. To display all the values, duplicate the memory channel twice, then use the Draw mode dialog to set the Data index to display in each duplicate.

See also:

XY trend plot dialog documentation, `CursorActive()`, `MeasureToChan()`, `MeasureToXY()`, `MeasureX()`, `MeasureY()`

MeasureToChan()

This creates a new Event, Marker or RealMark channel with an associated measurement process and cursor 0 iteration method for the current time view. Before calling this function, use `MeasureX()` to set the measurement method to 102 (Time) and set `expr1$` to generate the time stamps of the new items. If you select any other method this command will return an error code. The iteration must produce times in ascending order unless the output is to a memory channel.

If you are creating a RealMark channel, call `MeasureY()` before this function to define the first measurement to attach to each data item added to the new channel. You can define additional measurements by following `MeasureToChan()` by pairs of `MeasureY()` and `MeasureChan()` calls; each pair defines an additional

measurement. You must reserve space for all additional measurements with the `type%` argument. These command implement the functionality of the measure to channel dialog.

Once the channel has been created and all measurements set, use the `Process()` command to iterate the active cursor measurements; each successful iteration adds a set of measurements to the channel. When working with a sampling document you can also use `ProcessAuto()`.

```
Func MeasureToChan(dest%, name$, type%, mode%, chan%, min|exp${, lv|
lv${ ,hw{, flgs%{, qu${, width{, lv2|lv2${, code$}}}}}});
```

`dest%` This is the output channel number or zero for the lowest numbered, unused memory channel. The channel must be unused and can be either a disk-based channel or a memory channel. From Spike2 8.11 you can use `Chan("m1")` to get the number of the first memory channel, `Chan("m2")` for the second, and so on, which will future-proof your code against changes in channel mapping in future releases of Spike2. It is a fatal error to set a channel that is in use unless you use -1, meaning online use of the keyboard channel.

From Spike2 [10.10], for online (during sampling) use only you can set the value -1, meaning write to the keyboard marker channel. This is an experimental feature. It allows you to detect features and write a marker code that can trigger (albeit with some delay), the output sequencer and arbitrary waveform output without the need to write any script code. As only 1 output code is possible (set by `MeasureX()`), this has limited utility; if you want anything more complicated you will need to use a script.

`name$` The output channel name. Channel units, if required, are inferred from `chan%` and the measurement method.

`type%` This sets the output channel type, as for `ChanKind()`. The allowed types are: 2 or 3 for Event, 5 for a Marker, and 7 for RealMark. If you wish to store data other than event times in the file, you should set `type%` to 7. From version [10.03] onwards you can set `type%` to `-n`, to generate a RealMark channel with space reserved for `n` attached items at each cursor 0 iteration; setting type 7 generates a channel with space for one data item. You can reserve space for more measurements than you define; the additional items have the value 0.0 stored to them.

`mode%` This is the cursor 0 iteration mode. Modes are the same as in `CursorActive()` but not all modes can be used. Valid modes are (modes 21 and 22 added at [10.20]):

4 Peak find	12 Slope peak	16 Expression	21 Gap start
5 Trough find	13 Slope trough	17 Turning point	22 Gap end
7 Rising threshold	14 Data points	19 Outside levels	
8 Falling threshold	16 Expression	20 Inside levels	

`chan%` This is the channel searched by the cursor 0 iterator. In expression mode (16), this is ignored and should be set to 0.

`min` This is the minimum allowed step for cursor 0 used in all modes except 16.

`exp$` This is the expression that is evaluated in mode 16.

`lv` This number or string expression sets the threshold level for threshold modes and the peak size for peak and trough modes. It is in the y axis units of channel `chan%` or y axis units per second for modes 12 and 13. In mode 14 it sets the points as a number and defaults to 1. This argument is ignored and should be 0 or omitted for modes that do not require it.

`hw` This sets the hysteresis in y axis units for threshold modes. This argument is ignored and should be 0 or omitted for modes that do not require it.

`flgs%` This is the sum of option flags. Add 1 to force a common x axis Add 2 for user checks on the cursor positions. The default value is zero.

`qu$` This sets the qualification expression for the iteration. If left blank then all iteration positions will be used. If not blank, and it evaluates to non-zero, then the iteration is skipped.

`width` Set this value in seconds; use 0 or omit it in modes that do not require it. For slopes it sets the time over which the slope is measured. For peaks and troughs, it sets the maximum peak width (use 0 for no maximum). For threshold modes 7, 8, 19 and 20 it sets the minimum crossing time (Delay in the dialog)

`lv2` This number or string expression is used with `lv` to set the two threshold levels for cursor iteration modes 19 and 20.

code$ Used when a Marker or RealMark channel is the target. This sets the code to label the marker with as either a single character or as two hexadecimal digits. This corresponds to the Code field of the Measurements to Channel dialog. This item was added at version [10.20].

Returns The function result is the number of the created channel.

Delete process (new at [10.13])

To stop the Measurement process associated with a time view channel you can delete the channel. You can also delete the process but leave the channel as it is with this command variant.

```
Func MeasureToChan(chan%{, what%});
```

chan% The channel number with the associated process.

what% A code for the required action; taken as 0 if omitted. Currently only 0 is accepted, meaning delete the process. More values may be added in future.

Returns 1 if there was a process associated with the channel, 0 if there was not.

Change at [8.03]

Prior to version 8.03, for backwards compatibility with Spike2 version 3, if width was omitted, the hw parameter set the width in seconds for slope measurements. We no longer support this; if you need a non-zero width you must set it. If this causes you a problem, set the hw argument to 0 and add a width argument with the value previously passed in hw.

Changes at [10.03]

Prior to 10.03, MeasureToChan() could generate RealMark channels with a single measurement. From 10.03 onwards you can generate multiple measurements from each cursor 0 iteration. To define multiple measurements set the type% argument to minus the number of items to reserve space for. The MeasureToChan() command creates the channel with the reserved space and defines the x and first y measurement. Follow this with the extra items to measure. For example, this creates a RealMark channel with three measurements:

```
MeasureX(102, 0, "Cursor(0)", "0");     ' Define the time of each measurement
MeasureY(100, 1, "Cursor(0)", "0");     ' The first y measurement
var mc% := MeasureToChan(0, "Channel 1", -3, 4, 1, 0, "0.0", 0, 1, "", 0);
MeasureY(100, 1, "Cursor(0)+.1", "0"); ' Define second measurement
MeasureChan(0,"Channel 2");             ' Add second measurement
MeasureY(100, 1, "Cursor(0)+.2", "0"); ' Define third measurement
MeasureChan(0,"Channel 3");             ' Add third measurement
Process(0.0, MaxTime(), 0, 1, mc%);     ' process all available data
```

The MeasureX() command determines the time of the data item that is saved. The most common value to set is "Cursor(0)", but any value that increments for each active cursor 0 iteration is acceptable.

The MeasureY() command sets the current measurement. The current measurement is used by the MeasureToChan() to set the first measurement and by MeasureChan() to add additional measurements.

When used with RealMark output, each measurement may have its own title and units. The title is set by MeasureToChan() and MeasureChan(). The units are taken from the source channel f each measurement (if any), and may not be what you require. You can manipulate the output channel titles and units with the ChanTitle$() and ChanUnits$() command variants that include the index% argument.

See also:

CursorActive(), MeasureChan(), MeasureX(), MeasureY(), Process()

MeasureToXY()

This creates a new XY view with a measurement process and cursor 0 iteration method for channels in the current time view. It creates one output channel with a default measurement method. Use MeasureX(), MeasureY() and MeasureChan() to edit the method and add channels. Use Process() to generate the plot. The new XY view is the current view and is invisible. Use WindowVisible(1) to make it visible. These commands implement the functionality of the Measurements to XY view dialog.

```
Func MeasureToXY(mode%, chan%, min|exp${, lv|lv${ ,hw{, flgs%{, qu${,
width{, lv2|lv2$}}}}}});
```

`mode%` This is the cursor 0 iteration mode. Modes are the same as in `CursorActive()` but not all modes can be used. Valid modes are:

4 Peak find	12 Slope peak	16 Expression	21 Gap start
5 Trough find	13 Slope trough	17 Turning point	22 Gap end
7 Rising threshold	14 Data points	19 Outside levels	
8 Falling threshold	16 Expression	20 Inside levels	

`chan%` This is the channel searched by the cursor 0 iterator. In expression mode (16), this is ignored and should be set to 0.

`min` This is the minimum allowed step for cursor 0 used in all modes except 16.

`exp$` This is the expression that is evaluated in mode 16.

`lv` This number or string expression sets the threshold level for threshold modes and the peak size for peak and trough modes. It is in the y axis units of channel `chan%` or y axis units per second for modes 12 and 13. In mode 14 it sets the points as a number and defaults to 1. This argument is ignored and should be 0 or omitted for modes that do not require it.

`hw` This sets the hysteresis in y axis units for threshold modes. This argument is ignored and should be 0 or omitted for modes that do not require it.

`flgs%` This is the sum of option flags. Add 1 to force a common x axis. Add 2 for user checks on the cursor positions. The default value is zero.

`qu$` This sets the qualification expression for the iteration. If left blank then all iteration positions will be used. If not blank, and it evaluates to non-zero, then the iteration is skipped.

`width` Set this value in seconds; use 0 or omit it in modes that do not require it. For slopes it sets the time over which the slope is measured. For peaks and troughs, it sets the maximum peak width (use 0 for no maximum). For threshold modes 7, 8, 19 and 20 it sets the minimum crossing time (Delay in the dialog)

`lv2` This number or string expression together with `lv` sets the two threshold levels for cursor iteration modes 19 and 20.

Returns The function result is an XY view handle or a negative error code.

Arguments passed as strings are not evaluated until data is processed. Invalid strings generate invalid measurements and no data points in the XY view.

Example

This generates a plot of peak values in channel 1 of the current time view. Peaks must be at least 0.1 seconds apart and the data must fall by at least 1 y axis unit after each peak.

```
var xy%;                          'Handle of new xy view
xy%:=MeasureToXY(4, 1, 0.1, 1); 'Peak, chan 1, min step 0.1, amp 1
WindowVisible(1);                 'Window is invisible, so show it
MeasureX(102, 0, "Cursor(0)");   'x = Time, no channel, at cursor 0
MeasureY(100, 1, "Cursor(0)");   'y = Value of chan 1 at cursor 0
MeasureChan(1, "Peaks", 0);      'Set the title, no point limit
Process(0.0, View(-1).MaxTime(),0,1); 'Process all the data
```

Change at 8.03

Prior to version 8.03, for backwards compatibility with Spike2 version 3, if `width` was omitted, the `hw` parameter set the width in seconds for slope measurements. We no longer support this; if you need a non-zero width you must set it. If this causes you a problem, set the `hw` argument to 0 and add a `width` argument with the value previously passed in `hw`.

See also:

XY trend plot dialog documentation, `CursorActive(), MeasureChan(), MeasureX(), MeasureY(), Process()`

MeasureX()

MeasureX() and MeasureY() set the x and y part of a measurement. The settings are saved but have no effect until MeasureChan() or MeasureToChan() are used to change or create a channel. This command implements some of the functionality of the XY plot setting dialog. The current view must be the target of the measurements.

```
Func MeasureX(type%, chan%, expr1|coef% {,expr2 {,width}});
Func MeasureY(type%, chan%, expr1|coef% {,expr2 {,width}});
```

type% This sets the x or y measurement type. Values less than 100 match those for the ChanMeasure() command (except types 5 and 1 are identical). When using MeasureX() with MeasureToChan(), the only valid type% is 102. Values 20 and 21 were added at version [10.13].

1 Area	7 Modulus	13 Abs max.	19 Mean of absolute values
2 Mean	8 Maximum	14 Peak	20 Standard Error of the Mean
3 Slope	9 Minimum	15 Trough	21 Median
4 Sum	10 Peak to Peak	16 RMS error	
5 Area (scaled)	11 RMS Amplitude	17 Mean in X	
6 Curve area	12 Standard deviation	18 SD in X	

Values from 100 up are:

100 Value	104 0-based fit coefficient	108 Value product
101 Value difference	105 User entered value	109 Value less baseline
102 Time	106 Expression	
103 Time difference	107 Value ratio	

chan% This is the channel number for the measurement. For time, user entered and expression measurements it is ignored and should be set to 0.

expr1 Either a real value or a string expression that sets the start time for measurements over a time range, the position for time (102) and value measurements and the expression used for measurement type 106.

coef% The zero-based coefficient number for measurement type 104.

expr2 Either a real value or a string expression that sets the end time for measurements over a time range and the reference time for single-point measurements and differences. Set an empty string when width is required and this is not.

From Spike2 [10.10], for the special case of MeasureToChan() with a Marker or RealMark target channel, this field sets the channel marker code as a string as one character or two hexadecimal digits.

width This is the measurement width for value and value difference measurements. The default value is zero.

Returns The function return value is zero or a negative error code.

See also:
XY trend plot dialog documentation, Measurement descriptions, CursorActive(), MeasureChan(), MeasureToChan(), MeasureToXY()

MeasureY()

This is identical to MeasureX() and sets the y part of a measurement for a measurement channel. The settings are saved but have no effect until MeasureChan() is used to change or create a channel. See the MeasureX() documentation for details.

```
Func MeasureY(type%, chan%, expr1$ {,expr2$ {,width}});
```

See also:
MeasureX()

MemChan()

This function creates a new channel in memory and attaches it to the file in the current time view. You can have up to 2000 memory channels per file (old versions of Spike2 have lower limits). There are two variants: the first lets you specify the channels settings, the second (added at version [6.03]) copies settings from an existing channel.

```
Func MemChan(type%{, szLev%{, binsz{, pre%{, trace%}}}});
Func MemChan(0, copy%);
```

`type%` The type of channel to create. Codes are:

1 Waveform	4 Level (Event+-)	7 RealMark
2 Event (Event-)	5 Marker	8 TextMark
3 Event (Event+)	6 WaveMark	9 Real wave

`szLev%` For TextMark, RealMark and WaveMark channels it sets the number of characters, reals or waveform points to attach to each item. In level channels 0 or omitted sets the initial level as low, 1 sets high. Set 0 or omit for other channels. Setting the initial level was added at [7.08] and had no effect before this version. When setting points, the range is 1-1024 except for WaveMark channels, where the range is 6-1024 (6 is the minimum number of points that allows templating to work).

`binsz` Used for waveform and WaveMark data to specify the time interval between the waveform points. This is rounded to the nearest multiple of the underlying time resolution. If you set this 0 or negative, the smallest bin size possible is set.

`pre%` This must be present for WaveMark data to set the number of pre-trigger points. The allowed range is 0 to `szLev%`. Note that sampled WaveMark data has a maximum of 126 points per trace and 1 to points-1 pre-trigger points.

`trace%` Optional, default 1, sets the number of WaveMark traces in range 1 to 4. Although setting 3 traces is not forbidden, sampling does not support it.

`copy%` A channel from which to copy settings, including the channel comment, title, units and scale factors. The channel must exist.

Returns The new memory channel number, or 0 if there is no free channel, or a negative error code.

Channels created by the first variant have default titles, units, comments and scale factors. You can set these with `ChanTitle$()`, `ChanUnits$()`, `ChanComment$()`, `ChanScale()` and `ChanOffset()`. See the `ChanNew()` command for an example of setting these. This example copies a channel of any type:

```
func CopyWave%(chan%)
var mc%;
mc% := MemChan(0, chan%);                'Create waveform channel
if mc%>0 then                            'Created OK?
   ChanComment$(mc%, "Copy of channel "+Str$(chan%));
   MemImport(mc%, chan%, 0, MaxTime());  'Copy data
   ChanShow(mc%);                        'display new channel
endif;
return mc%; 'Return the new memory channel number
end;
```

See also:
`ChanNew()`, `MemDeleteItem()`, `MemDeleteTime()`, `MemGetItem()`, `MemImport()`, `MemSave()`, `MemSetItem()`

MemDeleteItem()

This function deletes one or more items from a channel created by `MemChan()`. To delete the entire channel use `ChanDelete()`.

```
Func MemDeleteItem(chan% {,index% {,num%}});
```

`chan%` The channel number of a channel created by `MemChan()`.

index% The ordinal index into the channel to the item to delete. The first item is numbered 1. If you specify index -1 or omit this argument, all items are deleted.

num% The number of items to delete from index%. Default value is 1.

Returns The number of items deleted or a negative error code.

See also:
MemChan(), MemDeleteTime(), MemGetItem(), MemImport(), MemSave(), MemSetItem()

MemDeleteTime()

This function deletes one or more items from a memory channel based on a time range. If a marker filter is set, only items in the filter are deleted unless you add 4 to mode%.

```
Func MemDeleteTime(chan%, mode%, t1{, t2});
```

chan% The channel number of a channel created by MemChan().

mode% This sets the items to delete and how to interpret the time range t1 and t2:

- 0 A single item is deleted. t1 is the item time and t2 is a timing tolerance (0 if t2 is omitted). The nearest item to t1 in the time range t1-t2 to t1+t2 is deleted. If there are no items in the range, nothing is deleted.
- 1 Delete all items from time t1-t2 to t1+t2. If t2 is omitted, 0 is used.
- 2 Delete the first item from time t1 to t2. If t2 is omitted it is taken as t1.
- 3 Delete all items from time t1 to t2. If t2 is omitted, it is taken as t1.
- +4 If you add 4 to the mode, any marker filter for the channel is ignored.

t1, t2 Two times, in seconds, that set the time range for items to delete.

Returns The number of items deleted, or a negative error code.

See also:
MemChan(), MemDeleteItem(), MemGetItem(), MemImport(), MemSave(), MemSetItem()

MemGetItem()

This returns information about a memory channel item. The items are identified by their ordinal position in the channel, not by time, and any mask set for markers is ignored. See MemSetItem(chan%, -1, time); to find index corresponding to a time.

```
Func MemGetItem(chan% {,index% {,code%[] {,&data$|data[]}}});
Func MemGetItem(chan%, index%, &wave|&wave%|wave[]|wave%[]{,&n%});
```

chan% The channel number of a channel created by MemChan().

index% The ordinal index into the channel to the item required. The first item is numbered 1. If you omit the index, or specify index 0, the function returns the number of items in the channel.

The remaining fields are only allowed if the index is non-zero.

code% This optional parameter returns codes from a marker type (marker, RealMark, TextMark, WaveMark) channel. From version [10.12] the first 4 elements (if they exist) are set to the marker codes associated with the item. The fifth element (code%[4]), if present, is set to the extra 32-bit value associated with each marker code in a 64-bit .smrx file. From version [7.10], code% can also be an integer holding the 4 marker codes as: code0 + 256*(code1 + 256*(code2 + 256*code3)).

Prior to version [10.12] codes% is an array with at least four elements that is filled in with the marker codes.

If the channel is not a Marker or derived type, the codes are set to 0.

data This must be a string variable for TextMark data or a real array for RealMark or WaveMark data. It is returned holding the data from the item. If the data type is incorrect for the channel, it is not changed. For an array, the points returned is the smaller of the size of the array and the number of values in the

item. You can use a two dimensional array to collect WaveMark data with multiple traces. The second dimension is for the traces: `var data[points%][traces%];`

`wave` This argument collects waveforms from waveform channels. If a real variable or array is passed, the waveform data is in user units. If an integer variable or array is passed, the data is a copy of the 16-bit integer data used by Spike2 to store waveforms. The maximum number of elements copied is the array size.

When an array is used, only contiguous data is returned. A gap in the data (when the interval between two points is greater that the sample interval for the channel) terminates the data transfer.

You may find that `ChanData()` is easier to use when you want to collect waveform (or event) data based on a time range.

`n%` If the previous argument is an array this optional argument returns the number of data items copied into the array.

Returns For `index%` of 0 or omitted, the function returns the number of items in the channel. If an index is given that is outside the range of items present, the function returns -1. Otherwise it returns the time of the item.

See also:
`ChanData(), MemChan(), MemDeleteItem(), MemDeleteTime(), MemImport(), MemSave(), MemSetItem()`

MemImport()

This function imports data into a channel created by `MemChan()`. There are some restrictions on the type of data channel that you can import from, depending on the type of the destination channel.

Destination	Source	Restrictions
Waveform	Waveform WaveMark All others	Data copied, but must match sample interval. Copy channel set by `MarkTrace()` if the sample rate matches (new in version 9.00). Also see `VirtualChan()`. Not available, see `VirtualChan()` or `EventToWaveform()`.
Event	Waveform All others	Can extract event times. Times are extracted from the channel.
Level	Waveform All others	Can extract event times. Times are extracted from the channel. First time in destination is assumed to be a low to high transition.
Marker	Waveform Event Level All others	Extract event times, coded for peak/trough and so on. Marker codes all set to 0. Rising edges coded as 01, falling as 00. Marker codes are copied.
TextMark	Waveform Event Level TextMark All others	Extract event times, coded for peak/trough and so on. Copies times, marker codes set 0, empty strings. As Event but rising edges code as 01, falling as 00. Copies all data, strings may be truncated if too long. Copies marker information, empty strings.
RealMark	Waveform Event Level RealMark WaveMark All others	Extract event times, coded for peak/trough & values. Times copied, marker codes and reals set to 0. As Event but rising edges code as 01, falling as 00. Copied, real data truncated or zero padded as needed. Copied, waveform to real values, padded/truncated. Marker portion copied, real values filled with 0.
WaveMark	Waveform Event Level	Special option, event channel marks waveforms. Copies times, marker codes set to 0, waveform set 0. As Event but rising edges code as 01, falling as 00.

	WaveMark All others	Copies all data, waveforms aligned on trigger point. Copies marker, waveform filled with zeros.

You can extract events or markers from waveform data using peak search and level crossing techniques. You can convert waveform data to WaveMark data with a special option that chops sections of waveform data out based on event times on a third channel.

Import compatible channel

The first command variant imports data from a compatible channel. All event and marker channels can be copied to each other, but the information transferred is the lowest common denominator of the two channel types. Missing data is padded with zeros. Waveform data is compatible with itself or WaveMark data if both channels have the same waveform sampling rate; you can use a virtual channel or a channel process (for Waveform channels) to interpolate to a different rate.

```
Func MemImport(chan%, inCh%, start, end);
```

`chan%` The channel number of a channel created by `MemChan()`.

`inCh%` The channel number to import data from.

`start` The start time to collect data from.

`end` The end time to collect data up to (and including).

Returns The number of items imported or a negative error code.

Extracting events from waveforms

The `mode%`, `time`, `level` and `code%` arguments are used when extracting events from waveform data. The level crossing modes use linear interpolation between points to find the exact time of the waveform crossing. The peak and trough modes fit cubic splines to the points around the peak or trough to estimate the time more accurately.

When extracting events to a RealMark channel, the first RealMark value is set to the peak, trough or level associated with the event. Peak and trough levels are calculated from a cubic spline through the data values to match the event time.

When extracting events to a WaveMark channel from waveform data, the time saved is the time of the start of the waveform section, not the peak/trough or level crossing time. The saved waveform data starts the number of points before each event set by the `pre%` parameter to `MemChan()`. The WaveMark time is adjusted to match the waveform. If both channels do not have the same sampling rate, the copied waveform is set to zero.

```
Func MemImport(chan%,inCh%,start,end{,mode%,time,level{,code%}});
```

`mode%` The mode of data extraction. The event times are based on:

- 0 Peaks in the waveform. If the destination is a marker, these events are coded as 2 unless `code%` is set.
- 1 Troughs in the waveform. If the destination is a marker, the events are coded 3 unless `code%` is set.
- 2 Waveform rising through `level`. If the destination is a marker, the events are coded 4 unless `code%` is set.
- 3 Waveform falling through `level`. If the destination is a marker, the events are coded 5 unless `code%` is set.
- 4 Peak and trough times. Peaks are always coded as 2, troughs are always coded as 3.
- 5 Level crossing times. Rising levels are always coded as 4, falling as 5.

`time` The minimum time between detected events. Use this to filter noisy signals. In modes 4 and 5, the timing is between events of the same code.

`level` In modes 0 and 1, this is the distance that the waveform must fall after a peak or rise after a trough. In modes 2 and 3 it is the level to cross to detect an event.

`code%` If present and positive it overrides the codes based on `mode%` that are applied to the events. The low byte of `code%` sets the first marker code; the remaining marker codes are always 0.

Returns The number of items imported or a negative error code.

WaveMark from events and waveform data

The special mode to convert waveform to WaveMark data uses an extra channel to mark the waveform sections to be extracted. The waveform must have the same sampling rate as set for the WaveMark channel. Each trigger time in the event channel is used to generate one WaveMark. If a trigger time does not align with the waveform data, the time is adjusted by up to 1/2 of the waveform sample interval so that it does align. The saved waveform data starts the number of points before each trigger time set by the `pre%` parameter to `MemChan()`. Any missing waveform points are set to 0.

```
Func MemImport(chan%, inCh%, start, end, eCh%);
```

`eCh%` A channel holding event times to mark the waveform sections to extract. The time saved is the time of the first point in each waveform section. If the channel contains marker codes, these are copied to the memory channel.

Returns It returns the number of WaveMark items added to the channel, or a negative error code.

See also:
`MemChan(), MemDeleteItem(), MemDeleteTime(), MemGetItem(), MemSave(), MemSetItem()`

MemSave()

This writes a channel created by `MemChan()` to the data file associated with the current window, making the data permanent. The new channel is created with the default drawing mode for the channel type. If you want to copy the source channel draw mode use the `DrawModeCopy()` command. The memory channel is not changed; use `ChanDelete()` to remove it.

```
Func MemSave(chan%, dest%{, type%{, query%{, bufSz%}});
```

`chan%` A channel created by the `MemChan()` function.

`dest%` The destination channel in the file. This must be in the range 1 to the maximum number of channels allowed in the data file.

`type%` The type of data to save the data as. The type selected must be compatible with the data in the memory channel. Codes are:

0 Same type (default)	3 Event (Event+)	6 WaveMark	9 Real wave
1 Waveform	4 Level	7 RealMark	
2 Event (Event-)	5 Marker	8 TextMark	

The special code -1 means append the memory channel to an existing channel. The new data must occur after the last item in the `dest%` channel and the `dest%` channel must be of a compatible type to the memory channel.

`query%` If this is not present or zero, and the `dest%` channel is already in use, the user is queried about overwriting it. If this is non-zero, no query is made.

`bufSz%` No longer used. In old versions of Spike2 if this was present and non-zero it set the disk buffer size used for waveform and Real wave data. Please delete this argument from old code.

Returns The number of items written, or a negative error code.

See also:
`ChanDelete(), ChanWriteWave(), DrawModeCopy(), MemChan(), MemDeleteItem(), MemDeleteTime(), MemGetItem(), MemImport(), MemSetItem()`

MemSetItem()

This function edits or adds an item in a channel created by `MemChan()`. The item is identified by its ordinal position in the channel and any mask set for markers is ignored. The first item in a memory channel has an index of 1.

```
Func MemSetItem(chan%, index%, time{, const code%[]{, data$|const
data[]}});
```

```
Func MemSetItem(chan%, index%, time, wave|wave%|const wave[]|const wave%
[]);
```

`chan%` The channel number of a channel created by `MemChan()`.

`index%` The index of the item to edit. The first item is number 1. An index of 0 adds a new item to the buffer at a position set by `time` (which must be positive). From Spike2 version [10.12] you can set `index%` to -1 to get the index of the first item at or past `time`.

`time` The item time, or -1 for no change. If `index%` is 0, you must supply a time. For a waveform channel, it sets the time of the first data point but if there is already data in the channel, `time` is adjusted by up to half the sampling interval to be compatible with the sampling interval of the channel and the existing data.

`code%` This is an integer array of at least 4 elements that hold the marker codes for the channel. If the channel does not require marker codes, this argument is ignored. If this parameter is omitted for a channel with markers, the codes are set to 0. From version [7.10], `code%` can also be an integer holding the 4 marker codes as: `code0 + 256*(code1 + 256*(code2 + 256*code3))`.

`data` A string for TextMark data or a real array for RealMark or WaveMark, holding the item data. If the data type is incorrect it is ignored. The number of points or characters set is the smaller of the number passed and the number expected. For RealMark and WaveMark data, if the array is too short, the extra values are unchanged when `index%` > 0 (editing) and have the value 0 if `index%` is 0.

WaveMark and waveform real values are limited to the range -5*`scale`+`offs` to 4.99985*`scale`+`offs`. A two dimensional array is allowed for WaveMark data: `var data[points%][traces%];` passed in as `data[][]`.

For WaveMark data, from version [10.14] you can also use an integer array, in which case the values are limited to the range -32768 to 32767.

`wave` For a waveform you can set one value, or an array. Real values are limited as described above. For integers, the lower 16-bits of the 32-bit integer are copied to the channel (values greater than 32767 or less than -32768 will overflow).

Returns The function returns the index at which the data was stored. The first item has an index of 1. If an index is given that is outside the range of items present, the function returns -1.

Get index for time

This variant was added at version [10.12]. It returns the index into the memory buffer of the first item at or after a time:

```
Func MemSetItem(chan%, -1, time);
```

`chan%` The channel number of a channel created by `MemChan()`.

`time` The item time to find.

Returns The index of the first item at or after `time`. The return value is -1 if there is no such item or the channel does not exist or is not a memory channel.

You can find the number of items in a memory channel with the `MemGetItem(chan%);` command.

Example

This example creates an array of events at time 1, 2, 3...10 seconds. It assumes that the current view is a time view; if not the `MakeEventChan%()` function will return -1.

```
'Create a new memory event channel hoding the times passed in an array.
'times      is an array of times to add to a new memory event channel
'returns    event channel or 0 if no data or -ve if an error
func MakeEventChan%(times[])
var ec%, n%, i%;
n% := Len(times[]);                 'number of points to add
if n% <= 0 then return 0 endif;     'no data
ec% := MemChan(2);                  'create a new memory channel
if (ec% < 0) then return -1 endif;  'failed to create
for i% := 0 to n%-1 do
    MemSetItem(ec%, 0, times[i%]);
next;
```

```
return ec%;
end;

var times[10], memChan%;
ArrConst(times, 1);ArrIntgl(times); 'create 1,2,3..10
memChan% := MakeEventChan%(times);  'make hidden event channel
ChanShow(memChan%);                 'show hidden channel
```

See also:
`ChanWriteWave(), MemChan(), MemDeleteItem(), MemDeleteTime(), MemGetItem(), MemImport(), MemSave()`

MenuCommand()

This command is used to emulate the user selecting a command from the menu. It was added at Spike2 version [10.16]. We anticipate using this command in tutorial scripts. It can also allow you to use built-in dialogs in a script, rather than having to create a user-defined dialog.

```
Func MenuCommand(cmd$);
```

`cmd$` This identifies the menu command to run. To identify the command, type the menu entries as you see them using vertical bar as a separator and omitting trailing dots and exclamation marks. For example, to run the View menu Change Colours... menu command, this would be: `"View|Change Colours"`. The matching ignores case, so `"view|change colours"` would also work. It also ignores trailing dots (...) and trailing exclamation points. This allows us to show the TalkerEx information dialog with `"Sample|Talkers|TalkerEx|Info"` without needing to know if it is currently running or not (the menu has an exclamation mark if it is not).

Returns If the menu item was located and issued, the return value is the index of the item that ran in the last matching sub-menu. If the entry was not found, the return value is -1. The return value is -2 if an intermediate part of `cmd$` is not a sub-menu and -3 if the last element of `cmd$` identifies a sub-menu and not a command.

Menus and the active view

In Spike2, the commands available in the menu depend on the active (front) view. This is the Time, Result, XY, Grid or text-based view with a highlighted title bar, usually the last view that the user clicked on. If your selected menu command depends on the current view you must make sure you have activated the desired view before using `MenuCommand()`. You make a view active with the `FrontView()` script command. As an example, the View menu Font... command exists in all versions of the menu, but it has different contents in a text-based view from the others. To make sure you have the right version of the Font menu:

```
FrontView(textH%);          'textH% holds the text view handle
MenuCommand("View|Font");   'show the menu
```

If the return value of the function is -1 and the cmd$ string is correct, the most likely reason is that the wrong view is active. You must use `FrontView(viewHandle%)` to set the active view, not `View(viewHandle%)` which sets the current view, but not the active view.

Effect of running this command

This command enables all menu commands as we assume that the script is allowed to do anything. When `MenuCommand()` returns, the set of allowed actions is restored to the state it was in when the command was called.

If the menu command opens a dialog that prevents any other activity until you close the dialog (a *Modal* dialog), the call to `MenuCommand()` will not return until you close the dialog. If this is used in a situation where an Idle routine is active, the Idle routine *will not* be called while the dialog is active. The script will not regain control until the user closes the dialog.

If the menu command opens a dialog that allows other activities to continue, such as the View menu Channel Draw Mode... command in a Time or Result view (a *Modeless* dialog), the `MenuCommand()` call will return immediately and your script must allow idle time for the user to interact with the dialog. In this case, it is possible that the current set of allowed actions may restrict what happens (set by the `allow%` argument to the function you used to release idle time).

Message()

This function displays a message in a box with an OK button that the user must click to remove the message. Alternatively, the user can press the Enter key.

```
Proc Message(form$ {,arg1 {,arg2...}});
```

`form$` A string that defines the output format as for `Print()`. If the string includes a vertical bar, the text before the vertical bar is used as the window title. There is no limit on the length of the text string you use here, but there is a limit on the space it can occupy in the Message dialog. You are allowed up to 80 dialog units wide (about 80 wide characters) and up to 40 dialog units high (at least 40 lines). To make use of this you must split the text into lines using `\n` to insert a new line. Alternatively, you can allow Spike2 to split up the lines, but this may not achieve the best possible results.

`arg1,2` The arguments used to replace `%d`, `%f` and `%s` type formats. All array elements are written with the same format. Array arguments can be `const`.

You can split the message into multiple lines by including `\n` in the `form$` string. Long messages are truncated.

See also:
`Print(), Input(), Query(), DlgCreate()`

Mid$()

This function returns a sub-string of a string.

```
Func Mid$(text$, index% {,count%});
```

`text$` A text string.

`index%` The starting character in the string. The first character is index 1.

`count%` The maximum characters to return. If omitted, there is no limit on the number.

Returns The sub-string. If `index%` is larger than `Len(text$)`, the string is empty.

Unicode
In a Unicode build of Spike2 the underlying characters are stored in a vector of 16-bit values and `index%` is the index into these values and `count%` is the number of these values. Most common Unicode characters fit in one 16-bit value, but some require two consecutive values, leading to the possibility that you might attempt to search half an extended character. The rule we apply is that if the start or end of any string section falls between the lead and trail codes of a Unicode surrogate pair, we move the start or end on by one position, to the start of the next character or the end of the string.

See also:
`Asc(), Chr$(), DelStr$(), InStr(), LCase$(), Left$(), Len(), Print$(), ReadStr(), Replace$(), Reverse$(), Right$(), Str$(), UCase$(), Val()`

Min()

This function returns the minimum of several real and/or integer variables, or the index of the minimum value in an array or sets the elements of an array to the minimum of itself and the corresponding elements of a second array. The three variants are:

Find minimum of a list of values

```
Func Min(val1 {,val2 {,val3...}});
```

`varN` The values to compare. The values are treated as real (even if they are integer variables).

Returns The minimum value. Note that if you use this to find the minimum of integers less than -9007199254740992, the result may be greater than you anticipate as doubles have a resolution of some 53 bits whereas integers have a resolution of 64 bits.

Find index of minimum value

Search all elements of an array for the minimum value.

```
Func Max(const arr[]);
```

`arr` A real or integer array.

Returns The index of the minimum value. If there are multiple indices with the same value, this finds the first.

Set minimum of corresponding array elements

This variant was added at Spike2 version [10.17].

```
Func Max(arr[]{[]...}, const src[]{[]...});
```

`arr` A real or integer array with 1 to 5 dimensions. If any of the corresponding elements of the `src` array are lower, the `arr` value is replaced. If `arr` is integer and `src` is real, the values are limited to integer range. If `arr` is real and `src` is integer, the values are limited by the resolution of real values.

`src` A real or integer array with the same number of dimensions as `arr`.

Returns 0

Example

Find the minimum in a sub-array holding 10 items of the original data:

```
var data[70], minPos%, minVal;
...
minPos:=Min(data[40:10]);   ' returns a position between 0 and 9
minVal:=data[40+minPos];    ' value of minimum
```

See also:

`Abs()`, `ATan()`, `Clamp()`, `Cos()`, `Exp()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Minmax()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`, `XYRange()`

MinMax()

`Minmax()` finds the minimum and maximum values for result views and time view channels with a y axis, or the minimum and maximum intervals for an event or marker channel drawn as dots or lines. `Min()` and `Max()` are preferred in result views.

```
Func MinMax(chan%, start, finish, &min, &max,{,&minP{,&maxP{, mode%
{,binsz {,trig%|edge%{, as%}}}}}});
```

`chan%` The channel number in the time or result view.

`start` The start position in time for a time view, in bins for a result view.

`finish` The end position in time for a time view, in bins for a result view.

`min` The minimum value is returned in this variable.

`max` The maximum value is returned in this variable.

`minP` The position of the minimum is returned in this variable. If there are multiple positions with the same value, this returns the first.

`maxP` The position of the maximum is returned in this variable. If there are multiple positions with the same value, this returns the first.

`mode%` If present, this sets the drawing mode in which to find the minimum and maximum. If `mode%` is absent or inappropriate, the display mode is used. This parameter is ignored in a result view. See the `DrawMode()` command for full details of all the drawing modes. The modes in a time view are:

- 0 The standard mode for the channel.
- 1 Dots mode for events.
- 2 Lines mode.
- 3 Waveform mode. This is the only mode for waveform channels.

4 WaveMark mode.
5 Rate mode. The `binSz` argument sets the width of each bin.
6 Mean frequency mode. `binSz` sets the time period.
7 Instantaneous frequency mode.
8 Raster mode. `trig%` sets the trigger channel.

`binSz` This sets the width of the rate histogram bins and the smoothing period for mean frequency mode when specifying your own mode.

`trig%` The trigger channel for raster displays, level data raster displays are impossible.

`edge%` For level data event channels. It sets which edges of the level signal are used for mean frequency, instantaneous frequency and rate modes. The values are:
0 Use both edges (same as omitting the parameter).
1 Use rising edges.
2 Use falling edges

`as%` Used with instantaneous frequency mode to determine how the data is measured. 0=Default, 1=Dots, 2=Line, 3=Skyline.

Returns Zero if all was well or a negative error code.

See also:
`ArrRange(), ChanValue(), DrawMode(), Min(), Max(), XYRange()`

MM (Multimedia) commands

These commands give you control over multimedia windows in Spike2 and the `MMRate()` commands lets you control how often frames are saved when sampling through the s2video application.

`MMOpen()` Open a multimedia window associated with the current time view or get the handle of an open multimedia view.

`MMAudio()` Get the sample rate of any audio stream in the window.

`MMVideo()` Get the size of the video image and the average frame rate.

`MMPosition()` Set or get the position in the mutimedia window, link the position to cursor 0 and step forwards and backwards by frames.

`MMImage()` Copy the current image to an array.

`MMFrame()` Get a list of frame times within a time range. This list does not include dropped frames (frames with no data).

`MMOffset()` Get or set the time of the first frame in the file relative to the Spike2 time.

`EditCopy()` Copy the current video image to the clipboard.

`FileSaveAs()` File types 13-16 only, save video image as a bitmap.

See also:
The Spike2 Video recorder, `MMAudio(), MMFrame(), MMImage(), MMOffset(), MMOpen(), MMPosition(), MMRate(), MMVideo()`

MMAudio()

This returns information about the audio content of the current multimedia window.

```
Func MMAudio({&rate});
```

`rate` If present, this is returned as the sample rate per channel, in Hz.

Returns The number of audio channels or 0 if no audio or not a multimedia window.

See also:
MMFrame(), MMImage(), MMOffset(), MMOpen(), MMPosition(), MMRate(), MMVideo()

MMFrame()

This command returns a list of real frame times in the current multimedia view, if this is supported by the multimedia file format. Before [10.06] this was supported only for AVI format files (.avi).

If you used s2video to capture the video and you set a low frame rate, the low rate is achieved by dropping frames from the file. In an AVI file (which assumes a constant frame rate) dropping frames is achieved by setting no data for a frame. The frame is still counted (to allow the file to be timed correctly), but not saving data allows us to save disk space. AVI frames with no data are not returned by this command. MP4 files achieve slow frame rates by not saving the frames. This command lets you get the time positions where frames with image data can be found in AVI files and all saved frames for MP4 files.

```
Func MMFrame(tStart, tEnd{, secs[]{, &more%);
Func MMFrame(tStart, tEnd, flags%{, secs[]{, &more%); [10.06] onwards
```

tStart The start of the time range (in seconds) to search for video frames.

tEnd The end of the time range to search for frames.

secs[] If present, a real array to be filled with frame times.

more% If present, set to 1 if the time range contains more frames than can fit in the array, else 0.

flags% [10.06] onwards. 0 to get all frames, 1 to get only key frames, 2 to get only non-key frames. This is ignored for AVI files which always return all frames.

Returns The return value is either the number of frames returned in the secs array, or if secs is omitted, it is the number of frames in the time range.

All frames in an AVI are positioned at a time given by offset + n%/FPS where offset is the time of the first frame and FPS is the number of frames per second. Frames may not exist for all values of n%. Frames in an MP4 file will tend to be equally spaced, within the capabilities of the hardware, but can have gaps due to setting a low frame rate or due to lack of system bandwidth, slow codecs or other system activity during data capture.

See also:
MMAudio(), MMImage(), MMOffset(), MMOpen(), MMPosition(), MMRate(), MMVideo()

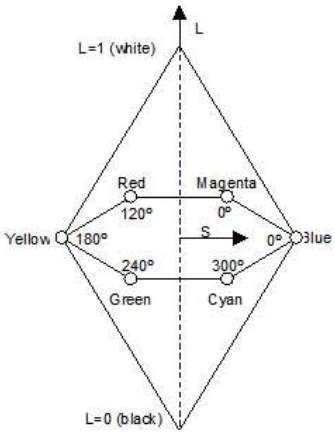
MMImage()

This copies the image at the current position in the current multimedia view to one or more data arrays. The two-dimensional arrays that collect the image are treated as x[h][v], where v is a vertical co-ordinate and h is a horizontal co-ordinate. x[0][0] is at the bottom left of the image. Use MMVideo() to get the image size. Sections of the array that do not map onto the image are not changed. BEWARE: Before Spike2 8.03, the image arrays were x[v][h].

```
Func MMImage(rgb%[][]);
Func MMImage(map[][]|mode%{, r[][]{, g[][]{, b[][]}}});
Func MMImage(-1, h[][]{, l[][]{, s[][]}}});
```

rgb% If the first argument is a two-dimensional integer array, it is returned with each array element holding one pixel in RGB32 format. Bits 0-7 hold the Blue intensity, bits 8-15 hold the Green and bits 16-23 hold the Red. Bits 24-31 are 0. Any additional arguments are ignored.

`mode%` If the first argument is 1, the `r`,`g` and `b` arrays are filled with Red, Green and Blue intensities. If `mode%` is set to 2, the arrays are filled with Cyan, Magenta and Yellow intensities. The colour components will be in the range 0 to 1.0, inclusive. If `mode%` is 0, the `r` array is filled with a monochrome representation of the data with 0 representing black and 1.0 representing white; the `g` and `b` arrays are set to 0, if they are present.



If the first argument is -1, the `h`, `l` and `s` arrays return HLS (Hue, Lightness and Saturation) data. Hue is in degrees around a colour hexagon with Blue at 0, Red at 120 and Green at 240 degrees. Lightness represents the amount of white in a colour, from 0 (black) to 1.0 (white). Saturation is a measure of the purity of a colour, from 0 (grey) to 1.0, which is a pure colour.

`map` If this is present, it should be a 3x3 array that maps the red, green and blue (RGB) in the image into the `r`, `g` and `b` arrays that you provide. The first array index sets the array: 0 for `r`, 1 for `g` and 2 for `b`. The second array index sets the colour: 0 for red, 1 for green and 2 for blue. The array values set how much of each colour should be added into the result.

If `R`, `G` and `B` (all in the range 0.0 to 1.0), represent the colour of a pixel, the `r` array pixel is set to `R*map[0][0]+G[]*map[0][1]+B*map[0][2]`. The `g` and `b` arrays (if present) are calculated with the first index of `map` set to 1 and 2.

`r,g,b` These arrays, when present are filled with RGB colour information for each pixel. If `map%` is used, the values in the arrays depend on the map values. If `mode%` is used, the values will be in the range 0 to 1.

`h,l,s` These arrays are used when the first argument is -1. They are returned holding the hue (0 to 360), lightness (0 to 1.0) and saturation (0 to 1.0).

Returns 1 if the copy succeeded and 0 if it failed.

You can use `EditCopy()` to put a bitmap copy of the image on the clipboard.

See also:
`EditCopy()`, `MMAudio()`, `MMFrame()`, `MMOffset()`, `MMOpen()`, `MMPosition()`, `MMRate()`, `MMVideo()`

MMOffset()

You can set the start time of the AVI file in the associated time view. We attempt to store this offset in the AVI file header in a supposedly unused region so that you only need do this once per file. Setting a 0 offset restores the AVI header to a standard state. You can also set an offset in the s2video program if your camera usually produces the same offset each time you use it.

```
Func MMOffset({offset});
```

`offset` If present, this sets the new offset in seconds. The offset is stored internally to millisecond resolution. The offset is saved to the AVI file when the multimedia window closes as long as the file is not read-only.

Returns The original multimedia offset at the time of the call, in seconds.

This functions is primarily intended to allow recordings made with s2video to be aligned. However, as we do not set a limit to the range of the offset, it can be used to align any AVI file to any offset in a file. If you do this, you will see the first or last video frame for time ranges beyond the time range spanned by the AVI file.

See also:
`MMAudio()`, `MMFrame()`, `MMImage()`, `MMOpen()`, `MMPosition()`, `MMRate()`, `MMVideo()`

MMOpen()

This counts, opens and returns the view handles of multimedia windows associated with the current time or multimedia view. Multimedia files associated with `fname.smr` are named `fname-1.avi`, `fName-2.avi` and so on. If a multimedia window is opened or located, it becomes the current view. If multiple windows open, the last-opened window becomes the current view. This does not access the `s2video` application window.

```
Func MMOpen({n%{, mode%}});
```

`n%` Set to 0 to open or count all associated multimedia files. Set to `n` (greater than 0) for the n^{th} multimedia file. It is not an error to open a file that is already open.

`mode%` This optional argument has the default value 0 and is the sum of the following:

- 1 Make new windows visible. Omitting this does not hide an existing window.
- 2 Reserved.
- 4 Do not open any new windows, do not change the current view. Use this to find a view handle or count the multimedia windows.

Returns If `n%` is 0 or omitted, the return value is the number of associated multimedia files that are open. If `n%` is greater than 0, and a file was opened, or was already open, the return value is the view handle.

See also:
`FileClose(), FileSaveAs() MMAudio(), MMFrame(), MMImage(), MMOffset(), MMPosition(), MMRate(), MMVideo()`

MMPosition()

This sets and gets the play position and state of the current multimedia window or sets the state of all multimedia windows associated with the current time view. Multimedia windows track any change in the associated time view position. Use this command to change the play position or state independently of the time view. Use `Rerun()` to replay a time view and all associated multimedia windows together to simulate data capture. Multimedia windows play in real time or at the rate set by the last `Rerun()` to the window. From version [10.01] you can rerun a multimedia window and use it to drive Cursor 0 in the associated time view.

```
Func MMPosition({pos{, sPlay%{, &gPlay%}}});
```

`pos` A position in seconds. If both `pos` and `sPlay%` are omitted, or both are negative, the command returns the current play position. Otherwise, if it is negative the value of `sPlay%` determines if it is taken as the current position or as the position of the end of the multimedia file.

`sPlay%` This argument controls the play state and the meaning of `pos`. If negative or omitted, it takes the sum of the values of the current play state (1 or 0) and cursor 0 link (2 or 0). Positive values are the sum of the following values in the range 0-7:

- 1 Run bit. Set 1 to run forwards, set 0 to Seek. A Seek operation causes the video system to discard all data and to prepare to play from time set by `pos`. A Seek can take a noticeable time, especially if a video is compressed. For example, a compressed file might have 1 key frame every 200 frames. A Seek operation will find the key frame at or before `pos`, then decode forward through the file until it reaches the desired time. You can use the `gPlay%` flags to detect when a Seek operation is complete. In an uncompressed file, all frames are key frames.
- 2 Cursor 0 link bit. Set this value to link the associated time view cursor 0 to the play position.
- 4 Run to bit. When combined with the run bit, run from the current position forward and stop at `pos`. If `pos` is negative, stop at the end of the multimedia file. If omitted, `pos` sets the initial seek position, if negative use the current position. There are two possible values for run to: 5 (run to with no cursor 0 link) and 7 (run to with cursor 0 link).

The value 4 is treated separately and is used to mean move by frame. If `pos` is greater than 0.0, step one frame forwards, if negative step one frame back. If zero seek to the current position. Beware that older versions of Spike2 (before 9.09 and 8.19) interpreted an `sPlay%` value of 3 as move by frame.

The value 6 is currently treated the same as 4, but is reserved and may be used for a different function in the future.

`gPlay%` If this argument is present it is returned as the sum of:

1 Set if the window is playing, unset if it is not.

2 Set if the play position is linked to cursor 0. This was added at [10.02, 9.10].

4 Set if the multimedia window has completed a Seek or run to operation since the last call to `MMPosition()`. To be updated, there must have been idle time or a `Yield()` call since the previous `MMPosition()` command. This flag was added at [10.02, 9.10].

Returns The return value after a step command is the position after the step. Otherwise it is the is the play position at the time of the call or -1 if an error occurred or no window was found. If the command is applied to a time view, the return value is for the lowest numbered multimedia window found.

Move by frame

Moving by frame can take a noticeable time if a video file is compressed. This is because of the need to find the previous key frame, then decode forwards to find the requested frame. Key frames can be rare (for example 1 every 200 frames) in a highly compressed file with a static image.

AVI files

To move by frame we have to decode information in the AVI file associated with the multimedia view. If this file contains missing frames, such as will be the case where `s2video` is set to sample at a slow frame rate, we skip over missing frames and move to the next real frame. This will work accurately if the frames (or missing frames) really did occur at the frame rate held in the AVI file.

MP4 files

Before version [10.07] this moved by the nominal time for a frame, which may be nonsense. From version [10.07] this attempts to move to the next or previous frame.

Waiting for move to position/frame to complete

The multimedia window hides a lot of complexity and much of the inner working is done on separate threads of execution that run in their own time. You must bear in mind that calls to `MMPosition()` are requests; just because you ask the video to move to a position does not mean that it has found the position (or that it will ever find it). If a file holds compressed video, it can take a considerable time to find a position. This is because the system searches for a key frame before or at the desired position, then processes frames forward from that point until it gets to the nearest frame that is not past the desired time. This process can take seconds...

For example, if you want to capture a screen image at a particular time, the following code may well not do what you want:

```
MMPosition(50);         ' Move to position
MMImage(0, imgArr[][]); ' Capture the image
```

This is because the position call requests the multimedia window to seek to position 50 seconds, but this could take a while to do, especially if the video is highly compressed and does not have many key frames. You may need code more like:

```
var timeOut:= Seconds() + 1.5;  ' allow 1.5 second seek time out
var flags% := 0;
MMPosition(50, 0, flags%);      ' Seek to 50 seconds
while (flags% < 4) and (Seconds() < timeOut) do
    Yield();                    ' allow idle time
    MMPosition(-1, -1, flags%); ' Wait for seek to complete
wend;
MMImage(0, imgArr[][]);         ' If it times out, this may not be useful
```

The first call to `MMPosition()` requests a Seek to position 50 seconds and collects the flags in case the operation completed during the call. The Seek completion can happen at any time after the request and is signalled by a Windows message. This message requires idle time for it to be processed, so the loop includes a call to `Yield()`. The second call to `MMPosition()` updates the flags so we can detect the end of the Seek.

See also:

`MMAudio()`, `MMFrame()`, `MMImage()`, `MMOffset()`, `MMOpen()`, `MMRate()`, `MMVideo()`, `Rerun()`

MMRate()

This requests any attached listener applications (usually `s2video` applications) to change the video frame rate. You cannot run faster than the camera frame rate. The rate is achieved by dropping frames from the video stream; the result may be faster or slower than you request. This command is usually used to move between a higher frame rate to record section of interest and a lower frame rate for uninteresting data and to save disk space.

```
Func MMRate(fps);
```

`fps` The desired frames per second. Set 0 for the minimum of the camera rate or the maximum frame rate set in the s2video application (that is, 0 sets the best rate that the s2video application can give you). If you set a negative frame rate, the command makes no change to the frame rate and returns the number of listener devices (normally s2video applications) that are registered with Spike2.

Returns The number of listener applications that Spike2 knows about (and has set the frame rate for if `fps` is greater than or equal to 0).

The normal use of this command is to modify the recorded video frame rate. However, it can also be used to detect that the s2video application is present and ready to record. You might want to do this if you use `ProgRun()` to start s2video as it can take several seconds for the application to start, locate the multimedia hardware to use and connect to Spike2. You can also use the `Listener()` command for this purpose.

Setting a rate of less than 10 and more than 0 is interpreted as setting a slow rate and will be saved in the `s2video` configuration as the new slow rate. It is recommended that you always use 0 to set the normal rate, even if you know the camera frame rate. This is because setting the camera rate may still cause frames to be dropped. If your camera runs at 30 FPS and you set 30.0, you could drop every other frame due to slight timing irregularities.

The rate can also be changed by the user selecting the rate control in the s2video application and by using the `SampleFPS()` when setting up sampling.

See also:
The Spike2 Video recorder, `Listener(), MMAudio(), MMFrame(), MMImage(), MMOffset(), MMOpen(), MMPosition(), MMVideo(), SampleFPS()`

MMVideo()

This returns information about the video content of the current multimedia window.

```
Func MMVideo({&hPix%{, &vPix%{, &fps}}});
```

`hPix%` If present, returned as the number of horizontal pixels in the image.

`VPix%` If present, returned as the number of vertical pixels in the image.

`fps` If present, returned as the frames per second value held in the image file. This is likely to be an approximate value. To find the times of frames use `MMFrame()`.

Returns 0 if no video is present or this is not a multimedia window or 1 if there is video in the multimedia window.

See also:
`MMAudio(), MMFrame(), MMImage(), MMOffset(), MMOpen(), MMPosition(), MMRate()`

Modified()

This command lets you get (and in some cases, set) the modified state of a view and if it is read only. Beware that clearing the modified flag for views that support this will allow you to close the view without being prompted to save changes.

```
Func Modified({what%{, new%}});
```

`what%` 0 (or omitted) to get or set the modified state, 1 to get or set the read only state.

`new%` The new state. -1 (or omitted) for no change, 0 to clear the state, 1 to set the state.

Returns The state at the time of the call, before any change.

The meaning and effect of this routine depends on the type of the current view.

Text view

A text view is considered modified if there are any changes since the last save. There is an exception to this; changes made by a script to the Log view do not count as modifications, only interactive changes (as the Log view does not have an associated disk file unlike all other text-based views). You can use `Modified(0,0)` to make the current state the last save point without saving the file; you cannot set the modified flag with this command.

Text views (but not the Log view) can be set read only with `Modified(1,1)` and the read only state can be cleared with `Modified(1,0)`. This prevents any text being added to them. Note that output sequence and script files open read only if they are marked read only on disk. This command does not change the read only status of the file on disk.

Time view

A Time window counts as modified if you make a change to it that would be written to the underlying `.smrx` or `.smr` file. `Modified(0)` reports if such a change has been made. Changes made to memory, duplicate or virtual channels do not count as modifications. Some changes are written immediately, others are buffered up and are only written when the file is closed or committed.

Modified(0,0)

You can force the data file to commit changes held in buffers and to write and changes in the file header with `Modified(0,0)`. Committing changes does not clear the modified flag. However, this does NOT guarantee that the changes will reach the physical disk surface, only that they are written through to the operating system. This means that should Spike2 stop working, as long as the system keeps running, your files will be OK. However, if system power fails, data may still be lost. If you use `Modified(0,0)`, all save/no save decisions made by `SampleWrite()` or triggered sampling become permanent up to the last data that was marked for saving. See `SampleAutoCommit()` for periodic commits.

Modified(0,1)

This does everything that `Modified(0,0)` does, and then asks the operating system to flush any data associated with the file through to the disk surface. BEWARE: this can be VERY SLOW (several seconds). This performs the same actions as the Sampling configuration Automation tab Flush option. Unless your data is very precious, and you can guarantee you are in a quiescent part of the experiment where having the entire system become unresponsive for several seconds is not a problem, do NOT use `Modified(0,1)`.

Modified(1)

`Modified(1)` reports the read only state. You cannot change the read only state of a time view; it depends on the read only state of the underlying `.smr` file.

Result and XY views

A result or XY view is modified if changed since the last save. You can set and clear the modified flag using `Modified(0, new%)`. There is currently no concept of a read only state for a result or XY views and `Modified(1)` always returns 0.

Other view types

The command is not implemented for other view types and will return 0.

MousePointer()

This command loads permanent mouse pointers from external files, creates new temporary mouse pointers and can delete temporary mouse pointers when they are no longer needed. Spike2 maintains a list of standard mouse pointers (see `ToolbarMouse()` for the list); any pointers added by this function are added to this list and are available for use by the mouse `Down%()`, `Move%()` and `Up%()` routines associated with `ToolbarMouse()`.

```
Func MousePointer(text$|nDel%);
```

`text$` This is a text string that defines a new mouse pointer. This is either the path to a data file that holds a cursor or an animated cursor, ending in `".cur"` or `".ani"` (not case sensitive) or it is a text string that defines a monochrome mouse pointer, as described below. The function returns the number of the new mouse pointer, or 0 if it was not created. Cursors loaded from a file are permanent and exist until Spike2 closes. There is a limit on the number of cursors that Spike2 will manage (currently set to 60); this should be more than enough for any reasonable purpose.

`nDel%` The number of a mouse pointer to delete, or -1 to delete all user-defined mouse pointers. You can only delete temporary mouse pointers, and you cannot delete a mouse pointer that is currently in use.

Returns The number of a newly created mouse pointer or 0 if it was not created or the number of mouse pointers that were deleted.

Text format to create a new pointer

Mouse pointers set by this function are 32 x 32 pixel images. Inside each image is the hot spot, being the position where the mouse is deemed to point. Each image pixel can be screen coloured, black, white or the inverse of the screen. In addition, you can designate a pixel to be the hot spot. This is coded in text as follows:

Character	Screen	Black	White	Inverse
Normal	space	b	w	i
Hot-spot	h	B	W	I

Cursors are defined by pixel by pixel, starting at the top left, moving horizontally and starting a new row every 32 characters. However, most mouse pointers are much smaller than 32 x 32 pixels, so you can stop a line early by adding a vertical bar character `"|"` or by a line feed character `"\n"`. Any character that is not a `space`, `b`, `B`, `w`, `W`, `h`, `i`, `I`, `|` or line feed is treated as a space. You do not need to provide 32 rows; omitted rows are treated as if they were filled with spaces. If there are multiple hot-spot characters the hot-spot position is set by the last hot-spot character. If there are none, the hot-spot is at the top left.

The following example creates a small square cursor:

```
var mp% := MousePointer(
"bbbbbbb|"
"bwwwwwb|"
"bw i wb|"
"bwihiwb|"
"bw i wb|"
"bwwwwwb|"
"bbbbbbb");
```

This could be written as:

```
var mp% := MousePointer("bbbbbbb|bwwwwwb|bw i wb|bwihiwb|bw i wb|bwwwwwb|bbbbbbb");
```

but the first arrangement is much easier to understand.

Loading mouse pointers from files

Cursors created with text strings are monochrome and not animated. You can also load coloured (sometimes called 3D) cursors and animated cursors from files. If you want to experiment with this, you can find suitable files in the `WINDOWS\Cursors` folder. On my machine, the following loads an animated stopwatch cursor:

```
var ani% := MousePointer("c:\\WINDOWS\\Cursors\\stopwtch.ani");
```

See also:

`ToolbarMouse()`

MoveBy()

This command can be used in both Text views and Grid views to move relative to the current position and to report the current position without moving. It can also be used with an external text file to set the file position.

Text view

This gets and sets the position of the text caret. You can move the text caret in a text window relative to the current position by lines and/or a character offset and you can extend or cancel the current selection. The caret position can be read as a position in the entire document or as a line and a column. If you use this to move past the end of a line, beware that in Windows, this is usually marked by two characters (`\r`, `\n`, `CR LF`).

```
Func MoveBy(sel%{, char%{, line%}});
```

`sel%` With `char%` present, if `sel%` is zero, all selections are cleared. If non-zero the selection is extended to the destination of the move. With `char%` omitted `sel%`=0 returns the character offset, 1 the line number and 2 the column.

`char%` If `line%` is absent, the new position is obtained by adding `char%` to the current character offset in the file. You cannot move the caret beyond the existing text.

`line%` If present it specifies a line offset. The new line is the current line number plus `line%` and the new character position is the current character position in the line plus `char%`. The new line number is limited to the existing text. If the new character position is beyond the start or end of the line it is limited to the line. You can use the `Draw()` command to position a text view to start at a given line.

Returns It returns the new position. `MoveBy(1,0)` returns the current position without changing the selection. See `sel%` (above) to get line and column numbers. You can use `XLow()` and `XHigh()` to get the first visible line and the line past the last fully visible line.

Unicode

When you use a Unicode build of Spike2, the text editor works in terms of byte positions in the text, not in terms of Unicode characters. If your text only contains ASCII characters with codes in the range 1 to 127 (0x01 to 0x7f), then positions in the text correspond with displayed characters. However, if your text holds characters with higher codes, each character may occupy from 1 to 4 bytes. We do not allow you to position the text caret in the middle of Unicode characters as then if you used `Selection$()`, the result could be rubbish. So the rules are that if you move the caret and the move ends in the middle of a multi-byte character, then the move continues in the same direction until the caret lies between Unicode characters. This means that character moves by 1 or -1 will always move the text caret by 1 Unicode character, as you would expect, though the position may change by more than 1.

External Text file

This moves the current position in the file for the next read or write operation.

```
Func MoveBy(sel%, char%);
```

`sel%` This is ignored and should be zero in case we decide to re-purpose it in the future.

`char%` The new position in the file as a byte offset relative to the current position (positive or negative).

Returns The position in the file (offset from the start) as a result of the move.

Grid view

In a grid view, this command is used to select cells in the grid relative to the last position. There is (currently) no script concept of selecting text ranges within a cell. The grid allows selection of multiple rectangular regions. See the `Selection()` command to get details of selected grid cells.

```
Func MoveBy(sel%{, col%{, row%}});
```

`sel%` If both `col%` and `row%` are omitted the action depends on the value of `sel%`:

`sel%`	Action
0	All selections are cleared, current position starts a selection and the cell number of the current position is returned.

1 None. Returns the column of the current position.

2 None. Returns the row of the current position.

If `col%` is present, `col%` (and `row%` if present) are added to the current column and row to generate a new position. If `sel%` is 0, all selections are cleared and the `row%` and `col%` start a new rectangular selection. If 1, the current selection is extended to the cell set by `row%` and `col%`. If 2, a new selection is started and added to any existing selection.

`col%` Optional. This is added to the last column position. The result is limited to the range 0 to the number of columns -1.

`row%` Optional, taken as 0 if omitted. This is added to the last column position. The result is limited to the range 0 to the number of rows-1.

Returns If `col%` is present, or if `sel%` is 0, the return value is the cell number that we moved to. This is calculated as the row number times the number of columns plus the column number. If `col%` is omitted, `sel%` = 0 returns the current cell number, `sel%` = 1 returns the current column number and `sel%` = 2 returns the current row number.

See also:
`Draw(),EditSelectAll(), MoveTo(),Selection(),Selection$(), XHigh(), XLow()`

MoveTo()

This command can be used in both Text views and Grid views. It can also be used with an external text file to set the next position to use for read or write.

Text view

This moves the text caret in a text window. You position the caret by lines and/or a character offset. You can extend or cancel the text selection. The first line is number 1. See `MoveBy()` to get the caret position as a line or column.

```
Func MoveTo(sel%, char%{, line%});
```

`sel%` If zero, all selections are cleared. If non-zero the selection is extended to the destination of the move and the new current position is the start of the selection.

`char%` If `line%` is absent, this sets the new position in the file. You cannot move the caret beyond the existing text. 0 places the caret at the start of the text.

`line%` If present it specifies the new line number. The new line number is limited to the existing text. `char%` sets the position in this line (and is limited to the line). You can use the `Draw()` command to position a text view to start at a given line.

Returns The function returns the new position in the file. You can use `XLow()` and `XHigh()` to get the first visible line and the line past the last fully visible line.

Unicode

In a Unicode build of Spike2, you can only position the caret between characters.

External Text view

This moves the current position in the file for the next read or write operation.

```
Func MoveTo(sel%, char%);
```

`sel%` This is ignored and should be zero in case we decide to use it in the future.

`char%` The new position in the file as a byte offset from the start.

Returns The position in the file as a result of the move.

Grid view

In a grid view, this command is used to select cells in the grid. There is (currently) no script concept of selecting text ranges within a cell. The grid allows selection of multiple rectangular regions.

```
Func MoveTo(sel%, col%, row%);
```

`sel%` If 0, all selections are cleared and the `row%` and `col%` start a new rectangular selection. If 1, the current selection is extended to the cell set by `row%` and `col%`. If 2, a new selection is started and added to any existing selection.

`col%` If this is positive, it sets the column unless it is too large, when it is limited to the available columns. If negative and `sel%` is 0 or 2, the entire row set by `row%` is selected unless `row%` is too large when the command is ignored. If negative and `sel%` is 1, the selection is extended from the current position to the first column.

`row%` If this is positive, it sets the row (unless it is greater than the number of rows when it is limited to the maximum value. If negative and `sel%` is 0 or 2, the entire column set by `col%` is selected (equivalent to selecting from (0, `col%`) to (*maxRow*-1, `col%`) unless `col%` is too large, when the command is ignored. If negative and `sel%` is 1, the selection is extended from the current position to the first row.

Returns Currently, this always returns 1.

To select the entire grid, use `EditSelectAll()` or set both `col%` and `row%` to -1.

See also:
`Draw()`, `EditFind()`, `EditSelectAll()`, `MoveBy()`, `Selection()`, `Selection$()`, `XLow()`, `XHigh()`

N

`NextTime()`

NextTime()

This finds the next item on a channel after a time. If a marker filter is in use, only data that is included in the filter is visible. It is an error to use this function in a result view.

```
Func NextTime(chan%, time{,&value|code%[]{,data[]|data%[]|&data$}});
```

`chan%` The channel number in the view to use.

`time` The time to start the search after. Items at the time are ignored. To ensure that items at time 0 are found, set the start time of the search to a negative time.

`value` Optional: for waveform channels it returns the waveform value. For event level channels, it is returned 0 if the transition is low to high, and 1 if the transition is high to low. If there is no event it returns the level at `time`; 0 for low, 1 for high.

`code%` This optional parameter is only used if the channel is a marker type (marker, RealMark, TextMark, WaveMark). From version [10.12] the first 4 elements (if they exist) are set to the marker codes associated with the item. The fifth element (`code%[4]`), if present, is set to the extra 32-bit value associated with each marker code in a 64-bit `.smrx` file.

Prior to version [10.12] this is an array with at least four elements that is filled in with the marker codes.

`data` Gets data from RealMark and WaveMark channels. If there is insufficient data, unused entries are unchanged. Integer arrays are for WaveMark channels and return the 16-bit data that holds the waveform. If WaveMark data has multiple traces, use a two dimensional array. The trace number is the second index. If you use a vector with multiple traces, the first trace is returned unless the `MarkTrace()` command has been used to set a different trace as the default.

`data$` A string returned holding the text from a TextMark channel.

Returns The time of the next item, -1 if there are no more items or a negative error code.

This function can be slow if run on a channel with a marker filter set and with the majority of events filtered out as Spike2 has to search for events that are in the filter. If you are using this to get Marker and extended Marker data, you could consider using `ChanData()`.

See also:
ChanData(), LastTime(), MarkTrace(), MaxTime()

O

Optimise()
OutputReset()

Optimise()

This optimises y axes over an x axis range in time, result or XY views in the same way as the Y Range dialog optimise button. You can optimise a hidden channel. In an XY view, all channels share the same y range, so optimising affects all channels. You can also use this on a spike shape window (with no arguments) to optimise the display.

```
Proc Optimise({cSpc{, start{, finish}}});
```

cSpc A channel specifier for the channels to optimise. If omitted, -1 is used, for all channels.

start The start of the region to optimise. This is in x axis units for a time or XY view and in bins for a result view. If omitted, this is the start of the window.

finish The end of the region to optimise. If omitted, this is the end of the window.

See also:
Channel specifiers, ChanOffset(), ChanScale(), YRange(), YLow(), YHigh()

OutputReset()

This command is equivalent to the Output Reset and Application Output Reset dialogs. It allows you to specify DAC and digital output levels to be set before and after sampling. There are three command versions: the first sets values and the second returns the set values, the third clears the settings.

```
Func OutputReset(flags%, const dacs%[], const dacv[], const dig%[]{,
rampt{, n1401%}});
Func OutputReset(dacs%[], dacv[], dig%[]{, &rampt{, n1401%}});
Func OutputReset({n1401%});
```

flags% When to apply. Sum of: 1 = at load/program start, 2 = before sampling, 4 = after sampling

dacs% An array of up to 8 elements. Element n corresponds to DAC n. When setting values, set each element to 1 to apply the associated DAC value, 0 to not apply the associated value. Values of 0 and 1 are returned when reading back values.

dacv An array of up to 8 DAC values in Volts to apply if the corresponding element of dacs%[] is not zero.

dig% An array of up to 16 elements, element n corresponding to digital output bit n. Set values as: 1=high, 0=low, -1 no change. Elements 0-7 correspond with the output sequencer DIGLOW command, elements 8-15 correspond with the DIGOUT command.

rampt Ramp time in seconds, default 0. This relates to a currently unimplemented feature. It will allow you to specify how long to take to ramp the DAC outputs to their final values for use in situations where a sudden DAC change could cause a problem.

n1401% Currently, set this to 0 to set/get values in the Output Reset dialog (sampling configuration) and 1024 for the Application Output Reset dialog (application preferences). We have plans to allow sampling with multiple 1401s. When this is enabled you will add the 1401 number to n1401%. If this is omitted, this value is taken as 0, meaning set the value in the current sampling configuration.

Return Both function versions return the flag% value at the time of the call.

The dacs%, dacv and dig% arrays can be any length without error. If longer arrays are supplied, the extra values are ignored. If shorter arrays are supplied, only the elements present are used. For the supplied values to

have any effect, at least one of the `flags%` values must be set and the desired output must be enabled with `dacs%` or `dig%`.

This command was added to Spike2 at version 7.11. The third variant to clear all settings was added at version 9.01.

P

PaletteGet()

Deprecated. This reads back the percentages of red, green and blue in a colour in the palette.

```
Proc PaletteGet(col%, &red, &green, &blue);
```

`col%` The colour index in the palette in the range 0 to 39.

`red` The percentage of red in the colour.

`green` The percentage of green in the colour.

`blue` The percentage of blue in the colour.

This function is now deprecated. You can replace `PaletteGet(col%, r,g,b)` with:

```
ColourGet(-1, col%, r, g, b); r*=100; g*=100;b*=100;
```

The `PaletteGet()` command should only be used when compatibility with older version of Spike2 is required.

See also:
Colour dialog, `ChanColour(), Colour(), PaletteSet(), ViewColour(), ViewUseColour(), XYColour()`

PaletteSet()

Deprecated. This sets the colour of one of the 40 palette colours. Colours 0 to 6 form a grey scale and cannot be changed. Colours are specified using the RGB (Red, Green, Blue) colour model. For example, bright blue is (0%, 0%, 100%). Bright yellow is (100%, 100%, 0%). Black is (0%, 0%, 0%) and white is (100%, 100%, 100%).

```
Proc PaletteSet(col%, red, green, blue{, solid%});
```

`col%` The colour index in the palette in the range 0 to 39. Attempting to change a fixed colour or a non-existent colour has no effect.

`red` The percentage of red in the colour.

`green` The percentage of green in the colour.

`blue` The percentage of blue in the colour.

`solid%` Ignored for many years. Was used to force the nearest solid (non-dithered) colour in ancient systems with limited colour capabilities.

This function is now deprecated. You can replace `PaletteSet(col%,r,g,b)` with `ColourSet(-1, col%, r/100, g/100, b/100)`. There is no equivalent of the `solid%` argument. `PaletteSet()` should only be used when compatibility with older versions of Spike2 is required.

See also:
Colour dialog, `ChanColour(), Colour(), PaletteGet(), ViewColour(), ViewUseColour(), XYColour()`

PCA()

This command performs Principal Component Analysis on a matrix of data. This can take a long time if the input matrix is large.

```
Func PCA(flags%, x[][]{, w[]{, v[][]}});
```

`flags%` Add the following values to control pre-processing of the input data `x[][]`:

1	Subtract the mean value of each row from each row
2	Normalise each row to have mean 0.0 and variance 1.0
4	Subtract the mean value of each column from each column
8	Normalise each column to have mean 0.0 and variance 1.0

You would normally pre-process the rows or the columns, not both. If you set flags for both, the rows are processed first. The most usual `flags%` value is 1 (or 0 if the rows already have a mean of 0). The Spike2 interactive PCA analysis sets `flags%` to 1 when working with waveforms.

`x[][]` A `m` rows by `n` columns matrix of input data that is replaced by the output data. The first array index is the rows; the second is the columns. There must be at least as many rows as columns (`m >= n`). If you have insufficient data you can use a square matrix and fill the missing rows with zeros. If you were computing the principal components of spike data, on input, each row would be a spike waveform. On output, each row holds the proportion of each of the `n` principal components scaled by the `w[]` array that, when added together, would best (in a least-squares error sense) represent the input data.

`w[]` This is an optional array of length at least `n` that is returned holding the variance of the input data that each component accounts for. The components are ordered such that `w[i] >= w[i+1]`.

`v[][]` This is an optional square matrix of size `n` by `n` that is returned holding the `n` principal components in the rows. Each row is a unit vector (the sum of the values squared is 1.0). This is equivalent to the `V`` matrix described in the overview of PCA.

Returns 0 if the function succeeded, -1 if `m < n`, -2 if `w` has less than `n` elements or `v` has less than `n` rows or columns.

You can find an overview of PCA in the Clustering spikes section of the manual. In the terms of that overview, `x[][]` corresponds with the X matrix on input and the U matrix on output, `w[]` is the diagonal of W and `v[][]` is the matrix V.

PlayOffline()

This command plays an area of the current time view through the 1401 DACs or through the sound card in the computer. If you close the view during sampling, output will cease. You cannot use the 1401 for sampling and `PlayOffline()` simultaneously. You cannot use the sound card if it is already in use. If you choose the sound card, the default wave out device is used.

Unless the waveform to replay is small (total points less than 32000), it is copied to the output device in chunks, as required. This operation happens in the background, but you must make time for it by allowing Spike2 to idle, or by checking the play position.

There are two versions of the command; the first starts output and the second reports the play position and allows you to stop the output early.

```
func PlayOffline(cSpc,dacs%,sTime,eTime{,rep%{,scale{,flag%}}});
func PlayOffline({what%{, &rep%}});
```

`cSpc` A channel specifier for up to 4 channels to be played. These can be any mix of waveform, WaveMark or RealWave channels. The channels can have different sample rates; the rate of the first channel is taken as the rate of all channels and the data for the other channels is matched to the first by linear interpolation. Gaps in the data are output as zeros.

`dacs%` This is either an integer vector of 1401 DAC channel numbers (the first DAC is numbered 0), or a single integer, being the number of the first DAC channel to use. If a single integer is used, and more than 1 channel is in use, the second, third and fourth channels use DACs `(dacs%+1) mod 4`, `(dacs%+2) mod 4`, `(dacs%+3) mod 4`. To use DACs 4 to 7 (with the Power1401), you must explicitly list the DAC numbers. If a vector is used, it can be `const`.

If the DAC number for the first channel is negative, output is to the system wave out device (usually a sound card). The maximum number of channels allowed depends on your system, but will usually be at least 2.

`sTime` The start time in the current time view. This, together with `eTime` defines the data region to play.

`eTime` The end time of the region in the current time view to output.

`rep%` In the set up call, this sets the number of times to play the output. If omitted, the value 1 is used. To play for as many repeats as possible use the value 0. The total number of points to play is limited to 2 to the power 32 (about 4 billion).

In the status call, this returns the number of repeats completed.

`scale` You can scale the replay rate in the range 0.25 to 4.0 (quarter speed to four times the channel rate). If you omit this argument, 1 is used, for output at the rate of the first channel.

`flag%` The sum of: 1 to position cursor 0 at the current replay position, 2 to rerun the file linked to the waveform output, 4 to broadcast cursor 0 changes and 8 (needs 4 set) to trigger active cursor searches on cursor 0 changes. The cursor is displayed if it is hidden. When the play finishes, the cursor is hidden if it was originally hidden.

`what%` Set this to –1 to stop output. Omit or set to 0 to report the position only. Both forms of the command return the position at the time of the call or –1 if there was no play in progress.

Returns The set up call returns 0 if all was OK, else –1 if there is a problem with the number of channels or DAC channels, -2 for an internal set up problem, -3 if the output rate is too fast, -4 if the rate is too fast to replay in chunks, -5 if memory was exhausted, -6 for a problem with the sound card, -7 for a stupid points count. The status call returns the replay position, in seconds, or –1 if there is no replay in progress, or it has finished.

See also:
Play offline dialog, `DlgShow()`, `Interact()`, `PlayWaveAdd()`, `ReRun()`, `Sound()`, `Speak()`, `Toolbar()`, `Yield()`

PlayWave...()

This family of commands gives you script control over the Play Waveform feature of Spike2. This has the following features:

- You can define up to 62 (20 with Micro2 and Micro3) areas in 1401 memory that are used to replay arbitrary waveforms through the 1401 DACs.
- Each area has an enable, one or two key codes, a label, a comment, a number of DAC channels, a size, a sampling rate and a speed override, a repeat count and a link to another area.
- You can define the waveforms to play as sections of an existing Spike2 data file, or set them as calculated waveforms using the script language, or they can be reserved area that you will fill dynamically with the script language.
- Areas can be linked if they have the same number of DAC channels and the same channel list. If you link areas, subsequent areas continue to play at the same rate as the original area.
- Areas can be set to start on an external trigger.
- Space is reserved for the areas when you open the Spike2 data file for sampling. The total size of the areas is limited by 1401 memory and by the need to leave sufficient 1401 memory to allow sampling to have a chance of working. Each area has a maximum size of 32 MB. See *Arbitrary Waveform Output* for the capabilities of each 1401 type.
- While sampling you can find out which area is playing and how far replay has progressed.
- You can update the waveforms in the areas while sampling and replay is in progress so it is possible to play back areas that are much larger than the available memory in the 1401.

Triggered sampling caveat

If you set an area up to play in triggered mode, the first data point of the area is transferred to the 1401 DACs so that it can be played exactly synchronously with the trigger. If you subsequently cancel that output, the act of cancelling may allow the DACs to update. If this is likely to be a problem for you it can be a good idea to make the first (and last) point of every waveform the same value (usually 0), so that you do not get unexpected glitches.

Script commands

The script commands are: `PlayWaveAdd()`, `PlayWaveChans()`, `PlayWaveComment$()`, `PlayWaveCopy()`, `PlayWaveCtrl()`, `PlayWaveCycles()`, `PlayWaveDelete()`, `PlayWaveEnable()`, `PlayWaveInfo$()`, `PlayWaveKey2$()`, `PlayWaveLabel$()`, `PlayWaveLink$()`, `PlayWavePoints()`, `PlayWaveRate()`, `PlayWaveSpeed()`, `PlayWaveStatus$()`, `PlayWaveStop()`, `PlayWaveTrigger()`

PlayWaveAdd()

This command adds a new area to the on-line arbitrary waveform output play wave list or replaces an existing area. When you create a data file, Spike2 reserves 1401 memory and transfers any stored waveform to it. The area is set to play once and is not linked to any other area. The area replay speed factor is set to 1.0 and the wave is set to non-triggered. You must use this command before you use `FileNew()` to create the sampling window. The order in which area are added determines the order of the buttons in the Play waveform toolbar. When an item is added, the area comment (new at version [10.20]) is set empty. You can set it with `PlayWaveComment$()`.

There are three command variants. The first adds a wave from the current time view or from a Spike2 data file (equivalent to the Add to online button in the Offline waveform output dialog), the second adds a wave from a data array, and the third reserves space without setting any data. There is a limit to the number of areas. In Spike2 version 9 onwards this is 62 ; before version 9 it was 10. It is an error to try to add a new key when the maximum number of keys is already defined. You can find how many keys exist by using `PlayWaveInfo$()` to get a string holding a list of the current keys. It is not an error to replace a key when the maximum number is defined.

Although you can define up to 62 areas, only the first 20 can be used with the Micro2 and Micro3.

```
func PlayWaveAdd(key$, lb$, dac%, sT, eT, wch%{, mem% {,path$}});
func PlayWaveAdd(key$, lb$, dac%, rate, const data{%}[]{[]});
func PlayWaveAdd(key$, lb$, dac%, rate, size%);
```

`key$` The first character of `key$` identifies this wave and triggers the wave playing in `SampleKey()`. It is an error to use `Chr$(0)` or an empty string as a key. Also, `Chr$(1)` and `Chr$(2)` have special uses and will not work as waveform triggers. It is also an error to attempt to define more keys that the maximum allowed number of areas. You can define a sub-key that can also be used to play an area with `PlayWaveKey2()`. Keys are case sensitive, so `"a"` and `"A"` are distinct.

From Spike2 version [10.17], if you set a key that matches an existing wave, the existing wave is replaced and maintains its position in the list of keys. Previously, the key was deleted first and the wave was added at the end of the list. This caused the order of buttons in the Play waveform toolbar to change.

`lb$` The label for the play wave control bar button that will play this wave, and record the character code as a keyboard marker. Labels can be up to 7 characters long. If you include `&` as a character, it will not appear on the button, but the next character will be underlined and can be used as a keyboard short-cut.

`dac%` Either a single DAC channel number or a `const` vector of DAC channel numbers. These are the outputs that will be used to play the data. The channel numbers must be in the range 0 to 3 (0 to 7 for the Power1401) and if more than one channel is specified, the channel numbers must be different.

`sT,eT` The start and end times of the data in either the current time view, or in the file identified by the `path$` variable to be used as a source of output data.

`wch%` Either a single channel, or a `const` vector of channels to use as a data source for playing. There must be one source channel for each output channel set by the `dac%` variable. The channels can be either waveform or WaveMark data. The sample rate is taken from the sampling rate of the first channel in the list. If subsequent channels in the list have different rates, data is interpolated.

`mem%` If present, and non-zero, the data is converted to a memory image and becomes independent of the data file. Otherwise, Spike2 stores the file name and extracts the data as required for sampling (so `path$` must be supplied or the current time view file must have been saved to disk).

`path$` This optional argument sets the name of the file to extract data from. If absent, the current time view data file is used as the data source. The named file must exist on disk and hold suitable data channels.

`rate` When the data does not come from a file, this value sets the sample rate for each channel in Hz. Spike2 will get as close to this rate as it can. The Power1401 mk II/-3 and Micro1401 mk II/-3 can set sampling intervals that are a multiple of 0.1 microseconds. The Power1401 allows intervals that are multiples of 0.2 microseconds. The Power3A and Micro4 also allow intervals that are multiples of 0.05 microseconds.

`data` This is either an integer or a real array. If there is more than one DAC channel to play, the array must have the same number of rows as there are DACs, that is if there are three DACs, the array must be equivalent to `var data[n][3];` where `n` is the number of data points. If this is an integer array, the bottom 16 bits of each element is played through the DACs. If the data is in a real array, we assume that the full range of the DACs is ±5 Volts and that the data is the required output value in Volts.

`size%` This is the number of data points per channel to reserve for this area. The data values are not specified and you must use `PlayWaveCopy()` to transfer data to the 1401 for playing after sampling has started.

Returns The memory bytes in the 1401 used to hold this data, or a negative error code.

This command does not transfer the data to the 1401, that happens when the start sampling command is given. Spike2 always keeps a minimum memory area for data sampling, so there is a limit to size of the waveforms that can be copied to the 1401. This size limit is not known until sampling starts. There is also a limit on the size of a waveform that can be stored in the list, which is 32,000,000 bytes (was 2,000,000 bytes before version [5.09]). Note that you can change the size of an area online with `PlayWavePoints()`; you cannot make the area larger than the size specified when sampling starts.

If you wish to link areas together or set the number of times the area is to be repeated or change the area speed factor, use one of the other `PlayWave...()` commands. The `SampleClear()` command removes all stored waveforms.

To start a waveform playing from the script you can use the SampleKey() function with the same key as set for the area or the output sequencer WAVEGO instruction.

This example generates two waveforms, one a rising ramp out of DAC 0 and the other a falling ramp from DAC 1. They are played at 1000 Hz.

```
var wave%[1024][2], dacs%[2];    'space for the waves and dac list
ArrConst(wave%[][0], 16); ArrIntgl(wave%[][0]);  'ramp up
ArrConst(wave%[][1], -16); ArrIntgl(wave%[][1]); 'ramp down
dacs%[0] := 0; dacs%[1] := 1;    'list of dacs
PlayWaveAdd("X", "Ramps2", dacs%[], 1000, wave%[][]);
```

The next bit of code will open a data file (assuming that there is a suitable sampling configuration set), start sampling and wait for sampling actually running, then set the area to play, wait for the area to have finished playing (in the simplest way) and then stop sampling.

```
var vh%;
vh% := FileNew(0, 1);     'Create a data file (should check vh%)
SampleStart(0);           'Tell it to start
while SampleStatus() <> 2 do Yield() wend; 'Wait for running
SampleKey("X");           'Tell area X to play
while PlayWaveStatus$() = "X" do Yield() wend; 'Wait for played
SampleStop();             'stop sampling
```

This is using the shortest and simplest code. "Real" code would check for error return values to avoid disappointment. For example, waiting while SampleStatus() is not 2 is not a good idea if SampleStart() failed.

If you want an area to play with a triggered start add the following line after the PlayWaveAdd() command:

```
PlayWaveTrigger("X", 1);  'Set this area to be triggered
```

In this case, the SampleKey("X") command will arm the output and the Trigger LED will turn on (unless you have routed the Trigger signal to the rear panel) and the output will wait for a trigger pulse on the Trigger input or rear panel.

See also:
FileNew(), PlayWaveChans(), PlayWaveComment$(), PlayWaveCopy(), PlayWaveCycles(), PlayWaveDelete(), PlayWaveEnable(), PlayWaveInfo$(), PlayWaveKey2(), PlayWaveLabel$(), PlayWaveLink$(), PlayWavePoints(), PlayWaveRate(), PlayWaveSpeed(), PlayWaveStatus$(), PlayWaveStop(), PlayWaveTrigger(), SampleClear(), SampleKey()

PlayWaveChans()

This function lets you read back or set the DAC channels assigned to a particular play wave area. You cannot change the DAC assignments after sampling has started. You cannot change the number of DACs set for an area without deleting and recreating the area.

```
func PlayWaveChans(key${, ch%[] {, set%}});
```

key$ The first character of the string identifies the play wave area.

ch% An optional integer vector used to collect or set the DAC channel numbers. The array size must match the number of channels when setting and be at least that size when reading.

set% If omitted or 0, the ch% array is filled in with the DAC channels used. If non-zero, the DAC channels are changed to the list defined by the ch% argument.

Returns The command returns the number of DACs in the area or a negative error code.

See also:
PlayWaveAdd(), PlayWaveInfo$(), PlayWaveLabel$(), PlayWaveLink$(), PlayWaveRate()

PlayWaveComment$()

The function gets and optionally sets the comment associated with each play area. This comment was added to Spike2 at version [10.20]. The comment is used to provide a tool tip for the Play waveform bar, and is available for any other purpose the user requires (for instance to hold information required by scripts, such as stimulation intensities).

```
func PlayWaveComment$(key${, new$});
```

`key$` The first character of the string identifies the play wave area.

`new$` If this is present, the comment for the area is changed to the string. If the string is more than 80 characters long, only the first 80 are saved. Leading and trailing white space is removed from the string before it is saved.

Returns The comment at the time of the function call.

Comments are a useful way to remind the user what an area contains. They can also be used by scripts to store information, such as stimulation intensities.

See also:
`PlayWaveAdd()`

PlayWaveCopy()

This command is used to update or read back a play wave data area in the 1401 memory. This can be done at any time that `SampleStatus()` returns 0, 1 or 2, even while a wave is playing.

```
func PlayWaveCopy(key$, data{%}[]{[]}{, offs%{, read%}});
```

`key$` The first character of the string identifies the area to be updated.

`data` This is either an integer or a real array. If there is more than one DAC channel to play, the array must have the same number of columns as there are DACs, that is if there are three DACs, the array must be equivalent to `var data[n][3];` where `n` is the number of data points. If this is an integer array, the bottom 16 bits of each element is played through the DACs. If the data is in a real array, we assume that the full range of the DACs is ±5 Volts and that the data is the required output value in Volts. The data in the array is copied to 1401 memory.

It is an error for the array size to be larger than the currently set play area in the 1401, even if it would fit in the area (due to the wrap around requirement). Due to an error, in the single channel case `data[]` was not accepted before [10.13].

From version [10.17], data can be `const` if `read%` is omitted or 0.

`offs%` The destination offset within the data area, in data points per channel. If the size of the data is such that the copy operation would extend beyond the end of the target area, the extra data is copied to the start of the area. If `PlayWavePoints()` has been used to reduce the area size, the copy operation wraps to the start at the size set by `PlayWavePoints()`. The first offset is 0. Use `PlayWaveStatus$()` to find the next offset to be written to the DACs.

`read%` Omit or set to 0 to set the memory in the 1401. Set to 1 to read back the 1401 data into the array. This argument was added at version [8.01]. The data array is not modified if `read%` is 0 or omitted.

Returns It returns a negative error code if no 1401 is open. The function returns 0 if all went well or if you call when sampling has stopped (when it has no effect).

Notes
Prior to version [10.02], you could not use this command when the 1401 was waiting for a triggered start (when `SampleStatus()` returned 1).

See also:
`PlayWaveAdd(), PlayWaveChans(), PlayWaveCycles(), PlayWaveLink$(), PlayWavePoints(), PlayWaveSpeed(), PlayWaveStatus$(), PlayWaveStop(), SampleClear(), SampleStatus()`

PlayWaveCtrl()

This gets and optionally sets options that control the use of arbitrary waveform output. You can read more about these options in the documentation for the Play waveform tab of the Sampling Configuration dialog.

```
Func PlayWaveCtrl(opt%{, new%});
```

`opt%` There is currently one option:

1 The play waveform output control. The values you can set with `new%` or read back are: 0 = keyboard, playwave tool bar, sequencer and script, 1 = toolbar, sequencer and script, 2 = sequencer and script only. It is an error to set values outside this range. The default value after `SampleClear()` is 0.

`new%` The new value for the control option.

Returns The value of the option selected by `opt%` at the time of the call (before it is changed by `new%`).

See also:
`SampleKey()`

PlayWaveCycles()

This function gets and sets the number of times to play a waveform area associated with a particular key. If this is used on-line, it will also change the number of repeats for the next play of the waveform.

```
func PlayWaveCycles(key$, {new%});
```

`key$` The first character of the string identifies the play wave area.

`new%` If present, this sets the number of cycles to play. The value 0 sets a very large number of cycles.

Returns The number of cycles set at the time of the call.

See also:
`PlayWaveAdd(), PlayWaveEnable(), PlayWaveInfo$(), PlayWaveLabel$(), PlayWaveLink$(), , PlayWaveRate(), PlayWaveSpeed(), PlayWaveStatus$(), PlayWaveStop()`

PlayWaveDelete()

This function deletes one or more play wave areas from the sampling configuration. If you do this while sampling is in progress it will not change the waves loaded to the 1401.

```
func PlayWaveDelete({keys$});
```

`keys$` If this is omitted, all play wave areas are deleted. If it is present, all areas with a key that is in this string are deleted. Case is significant for play areas; `"abc"` and `"ABC"` are not the same.

Returns The number of areas deleted or a negative error code.

See also:
`PlayWaveAdd(), SampleClear()`

PlayWaveEnable()

This function reports on the enabled state of a play wave area and optionally enables and disables it. Enabled areas are set up in the 1401 when sampling starts. Changes made with this function once sampling has started will not change the waves loaded to the 1401.

```
func PlayWaveEnable(keys$,{set%});
```

`key$` The first character of the string identifies the play wave area.

set% If present and zero, the area is disabled. If non-zero, the area is enabled.

Returns The enabled state (1=enabled, 0=disabled) of the area at the time of the call, or a negative error code.

See also:
PlayWaveAdd(), PlayWaveInfo$(), PlayWaveStatus$()

PlayWaveInfo$()

This function returns the list of keys associated with play wave areas, or gets the type of a particular area and the name of any associated data file.

```
func PlayWaveInfo$({key$, &size%, &type%});
```

key$ If omitted, the function returns the list of keys associated with the play wave areas. If present, information about a specific area is returned in the remaining arguments and the function returns any file name associated with the area.

size% If present, this is returned as the size of the data area in points per channel. This is not changed by PlayWavePoints() which sets the size actually used.

type% This returns the type of the area as: 0= unused, 1 = data is taken from a data file, 2 = area has a memory image of data (and may also have an associated data file), 3 = area is reserved by the script but there is no associated data.

Returns If there is no key$ value, then a string is returned holding one key character for each area. If there is a key, then the function returns the name of any data file associated with the area, in which case type% will be returned as 1 or 2.

See also:
PlayWaveAdd(), PlayWaveChans(), PlayWaveCycles(), PlayWaveEnable(), PlayWaveLabel$(), PlayWaveLink$(), PlayWavePoints(), PlayWaveRate(), PlayWaveSpeed(), PlayWaveStatus$()

PlayWaveKey2$()

You can define a second, sub-key, that can be used to identify an area for playing by SampleKey() or interactively. This can be useful when you reuse an area to have different contents or change an area size and you want to use a different key code to indicate this. It is an error to use this command if no 1401 is open for sampling. The original key code for the area still works if you set a second key.

```
func PlayWaveKey2$(key${, key2$});
```

key$ The first character of the string identifies the play wave area.

key2$ If present, the first character of this string sets the sub key. This can be an empty string to clear the sub key code. If key2$ matches an area code, or the sub key code for any other area, no change is made and the script stops with an error.

Returns The sub-key code that was active at the time of the call or an empty string if there was no sub-key.

See also:
PlayWaveAdd(), PlayWavePoints(), PlayWaveStatus$(), SampleKey(), SampleStatus()

PlayWaveLabel$()

This function returns and/or changes the label associated with each play area. The label can be up to 7 characters long and is used to label the buttons that appear on the Play waveform control bar. If you include & as a character, it does not appear in the label and the next character is underlined and can be used as a short cut to the button when the control bar is the current window.

```
func PlayWaveLabel$(key${, new$});
```

key$ The first character of the string identifies the play wave area.

new$ If this is present, the label for the area is changed to the string. If the string is more than 7 characters long, only the first 7 are used.

Returns The label at the time of the function call.

The label is a useful way to remind the user of the purpose of an area. If you need a longer description you can set the area comment.

See also:
PlayWaveAdd(), PlayWaveComment$()

PlayWaveLink$()

This links play wave areas together. Linked areas must have the same number of output channels and the same output channel list. The sample rate used is the sample rate for the area that is played first. You can change links during replay if SampleStatus() returns 0, 1 or 2. Both the area to link from and the area to link to must exist at the time of the call.

```
func PlayWaveLink$(key${, to$});
```

key$ The first character of the string identifies the play wave area.

to$ If present, the first character of this string sets the area to link to. Use Chr$(0) to cancel the link from the area set by key$.

Returns The key character of the area that was linked at the time of the call or an empty string if no area was linked or there was an error.

Notes
Prior to version [10.02], you could not use this command when the 1401 was waiting for a triggered start (when SampleStatus() returned 1).

See also:
PlayWaveAdd(), PlayWaveStatus$(), SampleStatus()

PlayWavePoints()

This function gets and sets the waveform area size associated with a particular key for use while sampling; it is an error to use this when not sampling data. You can only set sizes up to the original size of the area. This can be useful when you are working with a 1401 with restricted memory and need to reuse areas for different waves. If you change the size of an area while it is playing, the output immediately continues from the start of the area. This command does not affect the cycle count or any other feature of the area. You can use PlayWaveKey2$() to associate an additional key with this area to denote a different size or area content.

```
func PlayWavePoints(key${, new%});
```

key$ The first character of the string identifies the play wave area.

new% If present, this sets the new size in points per channel. From versions [9.11, 10.05], if new% is 0 or greater that the original area size, the original size is restored. Negative values make no change and report the original area size. Previously, out of range values generated an error.

Returns The size of the area in points per channel at the time of the call.

Implementation
Changes made by this command (and PlayWaveKey2$()) are not saved to the sampling configuration and are lost each time sampling stops. Each sample area has a fixed size inside the 1401 that is allocated when sampling starts. This command allows you to make the area appear smaller than the fixed size so as to match some particular waveform. If you try to use this command before sampling starts, or refer to an area that does not exist, the script will stop with a "Play wave area does not exist" error. The script also stops if you use this command with no 1401 open for sampling.

See also:
PlayWaveAdd(), PlayWaveCopy(), PlayWaveKey2$(), PlayWaveStatus$(), PlayWaveStop()

PlayWaveRate()

This function gets or sets the base play rate for a play wave area. This is the standard play rate that can be changed by PlayWaveSpeed(). Changes to the rate made after sampling starts have no effect on the output; use PlayWaveSpeed() for on-line changes and to read back the actual output rate during sampling.

```
Func PlayWaveRate(key${, new});
```

key$ The first character of the string identifies the play wave area.

new If present, this is the new play rate for the area, in samples per second. You can set a value in the range 0.01 to 500000 Hz and Spike2 will get as close as it can with the available hardware.

Returns The requested rate for the channel at the time of the function call.

See also:
PlayWaveAdd(), PlayWaveLink$(), PlayWaveSpeed(), SampleKey()

PlayWaveSpeed()

You can alter the sample rate for a play wave area by a factor of 0.25 to 4.0 with this command. Spike2 may not be able to play at the rate you request; it will set the closest rate it can. The Hz argument returns the achieved rate. On-line changes are allowed.

```
func PlayWaveSpeed(key${, new{, wait%{, Hz}}});
```

key$ The first character of the string identifies the play wave area. If this area is playing, or an area that this area links to, the rate will change during playing.

new If present, this is the new speed factor for the area, in the range 0.25 to 4.0. Spike2 gets as close to this speed factor as it can with the available hardware.

wait% If present and non-zero, any on-line speed change is postponed until the end of the current cycle and will happen within a few milliseconds of the cycle end.

Hz If present and a sampling document is open, it returns the real replay rate in Hz.

Returns The speed factor for the area at the time of the function call or 0 if there is no area defined by the key.

See also:
PlayWaveAdd(), PlayWaveLink$(), PlayWaveRate(), SampleKey()

PlayWaveStatus$()

This function returns information about waveform output during sampling.

```
func PlayWaveStatus${{&pos%{, &cyc%}}});
```

pos% If present, this returns the next position, in terms of the number of points per channel in the area, to write to the DAC hardware. The first position is 0. The DACs hold one data point (more if your 1401 supports and is using a DAC Silo) ready for output on the next clock tick, so pos% is ahead of where the DAC has got to. If you play an area with a triggered start but do not trigger it, pos% is not 0. This is because the DACs are already holding at least index 0 (more with a DAC Silo), ready to output when the trigger arrives. The pos% argument can be returned as -1 and cyc% returned as 1 if all data has been written to the DAC Silo, but playing has not yet finished.

cyc% If present, this integer variable is returned holding the number of cycles left to play (including the current cycle).

Returns The key code of the area that is playing or waiting for a trigger, or an empty string if no area is playing or sampling is not active. If an area is in Triggered mode, it counts as playing when it is waiting for a trigger.

To tell if an area in triggered mode has been triggered (started to output data), you need to see that `pos%` has changed or is set to -1.

DAC Silo

The DAC Silo is available with a Power3 and 2 and Micro4 and 3 (Power2 and Micro3 need up to date firmware). It allows data to be written through the DACs with a much lower processor overhead. Note that the DAC Silo is not used at lower sampling rates (less than 10 kHz) so that the reported position is reasonably close to the actual replay position.

See also:
`PlayWaveAdd(), PlayWaveCycles(), PlayWaveLink$(), PlayWaveRate(), PlayWaveSpeed(), PlayWaveStop(), SampleKey()`

PlayWaveStop()

This function requests that the currently playing wave is stopped, either immediately, or when the current cycle finishes.

```
func PlayWaveStop(cEnd%);
```

`cEnd%` If present and non-zero, the current cycle for the playing area will be the last cycle for that area, otherwise output will stop immediately.

Returns 1 for OK, 0 if not playing or a negative error code.

See also:
`PlayWaveAdd(), PlayWaveCycles(), PlayWaveStatus$(), SampleKey()`

PlayWaveTrigger()

This function reports and optionally changes the trigger state of a play wave area. If an area is triggered, a play request prepares the area but output does not start until a trigger signal is received by the 1401. This is the front panel Trigger input for the Micro1401 and Power1401 unless it is routed to the rear panel by the Edit Preferences menu.

```
func PlayWaveTrigger(keys${, set%});
```

`key$` The first character of the string identifies the play wave area.

`set%` If present this sets the triggered state. 0 = not triggered and non-zero = triggered.

Returns The trigger state (1=triggered, 0=not triggered) of the area matching the first character in `keys$` at the time of the call, or a negative error code.

If you want to know if an area in triggered mode has started to play, you can use the `PlayWaveStatus$()` command.

See also:
`PlayWaveAdd(), PlayWaveEnable(), PlayWaveStatus$()`

PolyEval()

This function evaluates a polynomial of the form:

$c_0 + c_1x + c_2x^2 + c_3x^3 \ldots c_nx^n$

at a given value of x given the coefficients c_i. You could write this code out in the script language, but this function is considerably faster, especially when the calculation involves complex numbers.

The command has the following variants that cover all combinations of real and complex coefficients and evaluation points:

Completely real evaluation

This function variation is used when the coefficients and the evaluation point are real numbers.

```
Func PolyEval(const c[], x);
```

`c` An array of real coefficients, length n (at least 1).

`x` The value at which to evaluate the polynomial.

Returns The value of the polynomial.

Complex evaluation

This is used whenever either the coefficients and/or the evaluation point are complex (have real and imaginary parts).

```
Func PolyEval(const c[]|c[][2], x|const z[2], &imag);
```

`c` An array of real coefficients, length n (at least 1) or a matrix with `c[][0]` holding the real parts of the coefficients and `c[][1]` holding the imaginary parts.

`x` The real value at which to evaluate the polynomial.

z The complex value at which to evaluate the polynomial, with z[0] holding the real part and z[1] holding the imaginary part.

`imag` A real variable that is returned holding the imaginary part of the result.

Returns The real part of the result.

We arbitrarily limit the order of polynomial to 100 (101 coefficients). It is your responsibility to ensure that the expression evaluation does not overflow, which means you must be aware of the behaviour of the polynomial when `|x|` or `|z|` > 1.

See also:

`PolyRoot()`

PolyRoot()

This function calculates the values (roots) of the polynomial equation:

$c_0 + c_1x^1 + c_2x^2 + c_3x^3 \ldots + c_nx^n = 0$

to generate the *n* roots r_i such that:

$(x - r_0)(x - r_1)(x - r_3) \ldots (x - r_{n-1}) = 0$

A polynomial of order *n* (the highest power of *x* in the polynomial) has *n* roots (some or all of which may be co-incident).

```
Func PolyRoot(const c[]|c[][2],r[][2]);
```

`c` An array of real coefficients or an array of complex coefficients with `c[][0]` holding the real parts and `c[][1]` holding the imaginary parts. The last coefficient and at least one other must be non-zero. We arbitrarily limit the number of coefficients to l000, but you should be aware that the accuracy will degrade as the order increases due to rounding errors and the root finding algorithm is likely to fail in the low hundreds of roots. The order of the polynomial is one less than the length of the first dimension of `c`.

`r` An array returned holding the roots of the polynomial as complex numbers. The first dimension of the array must be of length at least the number of coefficients -1. The roots are returned sorted in ascending order by their real values. If the coefficients are all real, the roots will be either real values with a zero (or close to it due to rounding errors) imaginary part, or occur in pairs of complex conjugates of the form *a* + *ib* and *a* - *ib*, where *i* represents the square root of -1 and *a* and *b* are real values.

Returns On success, the number of roots, which will be the number of coefficients-1, or 0 if there was a problem with the mathematics.

Example

The following example extracts the fifth roots of 1, that is we want to solve the equation $x^5 = 1$. In terms of a fifth order polynomial this is:

$-1 + 0x + 0x^2 + 0x^3 + 0x^4 + x^5 = 0$

The following code expresses this and prints the results:

```
var c[6] := {-1, 0, 0, 0, 0, 1};
var r[5][2];
var roots% := PolyRoot(c, r); ' solve x^5 = 1;
PrintLog("%.16g\n", r);
```

The results to 16 significant figures (around the accuracy of IEEE 64-bit doubles) are:

```
-0.8090169943749475, 0.5877852522924731
-0.8090169943749475,-0.5877852522924731
 0.3090169943749474, 0.9510565162951535
 0.3090169943749475,-0.9510565162951536
 1,0
```

where the two numbers represent the real and imaginary parts of the result. All the coefficients are real, and the results are two pairs of complex conjugates and one real value. Note that the third and fourth roots imaginary parts differ in the 16th decimal place.

Method

`PolyRoot()` uses Laguerre's method to locate each root. Our implementation finds the root nearest to 0 (for stability reasons), divides the root out of the polynomial and repeats with the reduced polynomial to find the next root. We then polish the roots by repeating the process for each root in turn, but this time starting the search at the roots found in the first pass and we do not reduce the polynomial (to avoid the accumulation of errors inherent in the first pass).

This is not guaranteed to find all roots; we have not seen it fail in real-world situations.

See also:

`PolyEval()`

Pow()

This function raises `x` to the power of `y`. If the calculation underflows, the result is 0.

```
Func Pow(x|x[]{[]...}, y);
```

`x` A real number or a real array to be raised to the power of `y`.

`y` The exponent. If `x` is negative, `y` must be integral.

Returns If x is an array, it returns 0 or a negative error code. If x is a number, it returns `x` to the power of `y` unless an error is detected, when the script halts.

See also:

`Abs()`, `ATan()`, `Cos()`, `Exp()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`

Print()

This command prints to the current view (log view, or created with `FileNew()` with a type of 1, 2 or 3, or with `FileOpen()` created with types 1, 2, 3 or 8) at the text caret. If the first argument is a string (not an array), it is used as format information for the remaining arguments. If the first argument is an array or not a string or if there are more arguments than format specifiers, Spike2 prints the arguments without a format specifier in a standard format and adds a new line character at the end. If you provide a format string and you require a new

line at the end of the output, include \n at the end of the format string (it generates the sequence CR LF in the output).

```
Func Print(form$|arg0 {,arg1 {,arg2...}});
```

form$ A string that specifies how to treat the following arguments. The string contains two types of characters: ordinary text that is copied to the output unchanged and format specifiers that convert the corresponding argument to text. If the argument is an array, all elements use the same format specifier. Format specifiers start with % and end with one of the letters d, x, c, s, f, e or g in upper or lower case. For a % in the output, use %% in the format string.

arg1,2 The arguments (of any type) used to replace %c, %d, %e, %f, %g, %s and %x type formats. Array arguments can be const.

Returns 0 or a negative error code. Fields that cannot be printed are filled with asterisks.

Format specifiers

The full format specifier is: %{flags}{width}{.precision}format

flags

The flags are optional and can be placed in any order. They are single characters that modify the format specification as follows:

- Specifies that the converted argument is left justified in the output field.
+ Valid for numbers, and specifies that positive numbers have a + sign.
space If the first character of a field is not a sign, a space is added.
0 For numbers, causes the output to be padded on the left to the field width with 0.
For x format, 0x is prefixed to non-zero arguments. For e, f and g formats, the output always has a decimal point. For g formats, trailing zeros are not removed.

width

If this is omitted, the output field will be as wide as is required to express the argument. If present, it is a number that sets the minimum output field width. If the output is narrower than this, the field is padded on the left (on the right if the - flag was used) to this width with spaces (zeros if the 0 flag was used). The maximum width for numbers is 100.

precision

This number sets the maximum number of characters to be printed for a string, the number of digits after the decimal point for e and f formats, the number of significant figures for g format and the minimum number of digits for d format (leading zeros are added if required). It is ignored for c format. There is no limit to the size of a string. Numeric fields have a maximum precision value of 100. The default precision value for real numbers is 6, for d and x formats is 1 and for string formats is 0.

format

The format character determines how the argument is converted into text. Both upper and lower cased version of the format character can be given. If the formatting contains alphabetic characters (for example the e in an exponent, or hexadecimal digits a-f), if the formatting character is given in upper case the output becomes upper case too (e+23 and 0x23ab become E+23 and 0X23AB). The formats are:

c The argument is printed as a single character. If the argument is a numeric type, it is converted to an integer, then the low byte of the integer (this is equivalent to integer mod 256) is converted to the equivalent ASCII character. You can use this to insert control codes into the output. If the argument is a string, the first character of the string is output. The following example prints two tab characters, the first using the standard tab escape, the second with the ASCII code for tab (8):

```
Print("\t%c", 8);
```

d The argument must be a numeric type and is printed as a decimal integer with no decimal point. If a string is passed as an argument the field is filled with asterisks. The following prints “ 23,0002”:

```
Print("%4d,%.4d", 23, 2.3);
```

e The argument must be a numeric type; otherwise the field is filled with asterisks. The argument is printed as {-}m.ddddde±xx{x} where the number of d’s is set by the precision (which defaults to 6). A precision of 0 suppresses the decimal point unless the # flag is used. The exponent has at least 2 digits (in some implementations of Spike2 there may always be 3 digits, others use 2 digits unless 3 are required). The following prints “2.300000e+01,2.3E+00”:

```
Print("%4e,%.1E", 23, 2.3);
```

f The argument must be a numeric type; otherwise the field is filled with asterisks. The argument is printed as {-}mmm.ddd with the number of d's set by the precision (which defaults to 6) and the number of m's set by the size of the number. A precision of 0 suppresses the decimal point unless the # flag is used. The following prints "+23.000000,0002.3":

```
Print("%+f,%06.1f", 23, 2.3);
```

g The argument must be a numeric type; otherwise the field is filled with asterisks. This uses e format if the exponent is less than -4 or greater than or equal to the precision, otherwise f format is used. Use a capital G to display the exponent as E rather than e. Trailing zeros and a trailing decimal point are not printed unless the # flag is used. The following prints "2.3e-06,2.300000":

```
Print("%g,%#g", 0.0000023, 2.3);
```

s The argument must be a string; otherwise the field is filled with asterisks.

x The argument must be a numeric type and is printed as a hexadecimal integer with no leading 0x unless the # flag is used. The following prints "1f,0X001F":

```
Print("%x,%#.4X", 31, 31);
```

Arrays in the argument list

The d, e, f, g, s and x formats support arrays. One dimensional arrays have elements separated by commas; two dimensional arrays use commas for columns and new lines for rows. Extra new lines separate higher dimensions. If there is a format string, the matching format specifier is applied to all elements. From Spike2 version [10.13], arrays can have zero size without error, in which case the format specification is replaced by nothing.

Infinity and Not a Number

The floating point number format normally stores a number within the floating point range. However, the format can also store positive infinity, negative infinity and Not a Number values. These are tricky to generate inside Spike2, but you can do it by using Virtual channel expressions like Ch(1)/Ch(2) where Ch(2) holds one or more zeros. You may also get these values if you read in data from a RealWave channel. If you try to print these values in f or g format you get #IND or QNAN for a NaN (for example the result of 0.0/0.0) or #INF or -#INF for an infinity (for example 1.0/0.0 or -1.0/0.0).

See also:

Message(), ToolbarText(), Print$(), PrintLog()

Print$()

This command prints formatted output into a string. The syntax is identical to the Print() command, but the function returns the generated output as a string.

```
Func Print$(form$|arg0 {,arg1 {,arg2...}});
```

form$ An optional string with formatting information. See Print() for a description.

arg1,2 The data to format into a string.

Returns It returns the string that is the result of the formatting operation. Fields that cannot be printed are filled with asterisks.

See also:

Asc(), Chr$(), DelStr$(), LCase$(), Left$(), Len(), Mid$(), Print(), PrintLog(), Replace$(), Reverse$(), Right$(), Str$(), UCase$(), Val()

PrintLog()

This command prints to the log window. The syntax is identical to Print(). The output always goes to the log views and is always placed at the end of the view contents.

```
Func PrintLog(form$|arg0 {,arg1 {,arg2...}});
```

`form$` An optional string with formatting information. See `Print()` for a description.

`arg1,2` The data to print.

Returns 0 or a negative error code. Fields that cannot be printed are filled with asterisks.

See also:
`Print(), Print$(), Message()`

Process()

Processes the current view using the process set by one of the `SetX...()` family of commands or interactively. For result and XY views, the time view to process data from must be open. The times for start and end of processing are times in the time view. Use `View(-1).Cursor(1)` to refer to time view times from result and XY views. There are two command variants:

Process a time range

Tells the current view to process a range of the time view.

```
Func Process(sTime, eTime{, clear%{, opt%{, dest%{, gate%{, len, pre{, mCd
%}}}}});
```

`sTime` The time to start processing from, in seconds. From [10.11] negative times are allowed. There is no data at negative times, but when using a `MeasureToChan()` or `MeasureToXY()` process a negative time means start from the beginning and ignore the minimum time since the last found item for the first search. Times greater than `Maxtime()` cause no processing. In triggered modes with no trigger channel, this sets the trigger time and `eTime` is ignored. In Phase histograms with no Cycle channel, this sets the start of a cycle.

`eTime` The end time for processing. In Phase histogram mode with no Cycle channel, this sets the end time of a single cycle.

`clear%` If present, and non-zero, the result view bins are cleared before the results of the analysis are added to the result view and `Sweeps()` result is reset.

`opt%` If present and non-zero, the result view is optimised after processing the data.

`dest%` Used when processing to a channel from `MeasureToChan()`. If not used it should be set to 0. This identifies the destination channel for processing. If omitted or 0, all suitable channels are processed. It is not an error if this doesn't match a destination channel, but no processing happens.

`gate%` If present and greater than 0, this is the channel number of an event or marker-based channel in the associated time view to use as a gate and both `len` and `pre` must be present. If 0, use the gating set by the most recent `ProcessGate()` command.

`len` The time to process for each gate event in the time `sTime` to `eTime`.

`pre` How far before each gate event to start processing. Negative values are allowed and start the processing after the trigger.

`mCd%` If present and `gate%` is a marker or derived type, it holds the marker code to use as the gate. Set this to -1 or omit it to use the current channel marker filter.

Returns This returns the number of items processed. This is the number of intervals considered for an INTH (even if they didn't fall in the histogram), the number of sweeps for sweep-based analysis, the number of data blocks for `SetPower()`. In the case of an error, a negative error code is returned.

Process one value (Measure Now)

This is equivalent to the Analysis menu Measurements->Measure Now command. For this to make any sense you must have set the positions of any cursors used in the measurement as no cursor search is done.

```
Func Process()
```

Returns 1 if a point was added to a result view, 0 if no point added and a negative code if an error was found.

See also:
MeasureToChan(), MeasureToXY(), ProcessAll(), SetAverage(), SetEvtCrl(), SetINTH(), SetPhase(), SetPSTH(), SetPower(), SetResult(), SetWaveCrl(), Sweeps()

ProcessAll()

This function runs all processes that use the current time view as a data source using the process set by one of the SetX...() family of commands or interactively. It is as if a Process() command were used for each target view using the current processing settings (the settings you would see if you opened the Process Settings dialog for the target view or channel). If you have already used a Process() command on a target, this sets the current processing settings.

```
Func ProcessAll(sTime, eTime);
```

sTime The time to start processing from, in seconds. From [10.11] negative times are allowed. There is no data at negative times, but when using a MeasureToChan() or MeasureToXY() process a negative time means start from the beginning and ignore the minimum time since the last found item for the first search. Times greater than Maxtime() cause no processing. In triggered modes with no trigger channel, this sets the trigger time and eTime is ignored. In Phase histograms with no Cycle channel, this sets the start of a cycle.

eTime The end time for processing, which must be positive. In Phase histogram mode with no Cycle channel, this sets the end time of a single cycle.

Returns Zero if no errors or a negative error code.

See also:
MeasureToChan(), Process(), MeasureToXY(), SetAverage(), SetEvtCrl(), SetINTH(), SetPhase(), SetPSTH(), SetPower(), SetResult(), SetWaveCrl(), Sweeps()

ProcessAuto()

This is equivalent to the Process Dialog for a New file in Automatic mode. The current view must be the view where the results appear. The processing parameters set by this command are used when this view is given a chance to update. Processes are created with the SetX...() family of commands or interactively.

```
Func ProcessAuto(delay, mode%{, opt%{, last{, leeway{, dest%}}}});
```

delay The minimum time between updates, in seconds.

mode% 0=accumulate all data. 1=clear the result, process the most recent last seconds.

opt% If present and non-zero, the result view is optimised after each process.

last The length of time to process in mode 1, in seconds; ignored in mode 0.

leeway How close cursor 0 can be to the file end for XY views and MeasureToChan().

dest% The channel for MeasureToChan(), ignored for other process types. If omitted or 0, all suitable channels are processed. It is not an error if this doesn't match a destination channel, but no processing happens.

Returns 0 or a negative error code.

See also:
MeasureToChan(), MeasureToXY(), ProcessTriggered(), Process()

ProcessGate()

This command is equivalent to the Gate Settings dialog. It allows you to define the start and end of the time region to be processed relative to events and markers for use by the next Process() or ProcessTriggered() command by setting the gate% argument to 0, which takes a copy of the gate state set by this command. This does not exist before version 10.

The command has three variants:

Set fixed duration mode

This command variant sets the current gate state with a fixed process period relative to a trigger. This is equivalent to the gating mode you can implement with the `Process()` command.

```
Func ProcessGate(chan%, pre, len{, mCd$});
```

`chan%` An event, level or marker channel to use as the trigger time.

`pre` Specifies the pre-trigger time in seconds, set negative to start after the trigger event.

`len` Specifies the sweep length in seconds.

`mCd$` If present and not an empty string, this defines the marker code filter to apply to `chan%` to generate the start events. If omitted, or empty, the current channel marker filter settings are used.

Return 0

Set gated duration mode

This command variant sets the current gate state with a variable duration. The gate start time is set in the same way as for fixed duration mode, but the end time now depends on detecting events.

```
Func ProcessGate(chan%, pre, mCd$, chan2%, post{, mCd2$});
```

`chan%` An event, level or marker channel to use as the time for the start event.

`pre` Specifies the pre-gate start event time in seconds, set negative to start after the start event.

`mCd$` If not an empty string, this defines the marker code filter to apply to `chan%` to generate the trigger events. If empty, the current channel marker filter settings are used.

`chan2%` An event, level or marker channel to use as the time for the stop event. This can be the same channel as `chan%`. The search for a stop event starts one clock tick after the start event.

`post` Specifies the post gate end event time in seconds, set negative to stop before the end event.

`mCd2$` If present and not an empty string, this defines the marker code filter to apply to `chan2%` to generate the stop events. If omitted, or empty, the current `chan2%` marker filter settings are used.

Return 0

Collect Gating information from current view

This command variant sets the current gating state of the `ProcessGate()` command to match the process attached to the current view (or makes no change if there is no suitable process).

```
Func ProcessGate();
```

Return 1 if there was a process attached to the current view, 0 if there was not.

Report the stored Gating state

This command variant reports the state of the stored gating information.

```
Func ProcessGate(item%{, &mCd$});
```

`item%` A negative code to select the information that is returned. See the table, below.

`Item%`	`mCd$` if present	Return value
-1	Trigger/gate start code	Trigger/gate start channel
-2	Trigger/gate start code	Pre-trigger/pre-gate start time
-3	Empty/gate end code	0/gate end channel
-4	Empty/gate end code	Sweep length/post-gate end time

The value before the / is for fixed duration mode, the value after is for gated duration mode.

`mCd$` If present, returned holding the marker code associated with `item%`.

Return The information selected by `item%`.

ProcessTriggered()

This is equivalent to the Process Dialog used with a New file in Gated mode. The name is unfortunate, and should really be `ProcessGated()`, but we kept the original name for backwards compatibility. The current view must be the view where the results appear. The processing parameters set by this command are used when this view is given a chance to update. Processes are created with the `SetX...()` family of commands or interactively.

```
Func ProcessTriggered(len, pre, gate%{, clr%{, opt%{, mCd%{, dest%}}}});
```

`len` The length of data to process around each gate event, in seconds.

`pre` The pre-event time, in seconds.

`gate%` If present and greater than 0, this is the channel number of an event or marker-based channel in the associated time view to use as a gate and both `len` and `pre` must be present. If 0, use the gating set by the most recent `ProcessGate()` command and `len` and `pre` and `mCd%` are ignored.

`clr%` If present, and non-zero, the result view bins are cleared before the results of the analysis are added to the result view and `Sweeps()` result is reset.

`opt%` If present and non-zero, the result view is optimised after processing the data.

`mCd%` If present and `gate%` is a marker or derived type, it holds the marker code to use as the gate. Set this to -1 or omit it to use the current channel marker filter.

`dest%` The channel for `MeasureToChan()`, ignored for other process types. If omitted or 0, all suitable channels are processed. It is not an error if this doesn't match a destination channel, but no processing happens.

Returns 0 or a negative error code.

See also:
`MeasureToChan(), MeasureToXY(), Process(), ProcessGate()`

Profile()

Spike2 saves information in the `HKEY_CURRENT_USER\Software\CED\Spike2` section of the system registry. The registry is a tree of keys with values attached to each key. If you think of the registry as a filing system, the keys are folders and the values are files. Keys and values are identified by case-insensitive text strings. The `Profile()` command manipulates the keys and values within the `Spike2` section of the registry.

You can also view and edit the registry with the `regedt32` (or `regedit`) program, which is part of your system. Select Run from the start menu and type regedt32 then click OK. Please read the regedt32 help information before using this program. It is a very powerful tool; careless use can severely damage your system.

Do not write vast quantities of data into the registry; it is a system resource and should be treated with respect. If you must save a lot of data, write it to a text or binary file and save the file name in the registry. If you think that you may have messed up the Spike2 section of the registry, use `regedt32` to locate the `Spike2` section and delete it. The next time you run Spike2 the section will be restored; you will lose any preferences you had set.

```
Proc Profile(key${, name${, value%{, &read%}}});
Proc Profile(key${, name${, value${, &read$}}});
```

`key$` This sets the key to use in the `Spike2` registry section. Set `key$` empty to use the `Spike2` key. You can use nested keys separated by a backslash, for example `"My bit\\stuff"` to use the key `stuff` inside the key `My bit`. The key name may not start with a backslash. Remember to use two

backslashes inside quote marks; a single backslash is an escape character. It is never an error to refer to a key that does not exist; the system creates missing keys for you.

`name$` This string identifies the data in the key to read or write. If you set an empty name, this refers to the `(default)` data item for the key set by `key$`.

`value` This is either a string or an integer. If `read` is omitted, `value` is written to the registry. If `read` is present, `value` is returned if the registry item does not exist, that is, it provides a "default" value.

`read` If present, it must have the same type as `value`. This is a variable that is set to the value held in the registry. If `name$` is not in the registry, `read` is set to `value`.

`Profile()` can be used with 1 to 4 arguments. It has a different function in each case:

1 The key identified by `key$` is deleted. All sub-keys and data values attached to the key and sub-keys are also deleted. Nothing is done if `key$` is empty.
2 The value identified by `name$` in the key `key$` is deleted.
3 The value identified by `name$` in the key `key$` is set to `value%` or `value$`.
4 The value identified by `name$` in the key `key$` is returned in `read%` or `read$`.

The following script example collects values at the start, then saves them at the end:

```
var path$, count%;
Profile("My data", "path", "c:\\work", path$); 'get initial path
Profile("My data", "count", 0, count%); 'and initial count
...                                     'your script...
Profile("My data","path", path$);       'save final value
Profile("My data","count", count%);     'save final count
```

A common use of the `Profile()` command is where a script starts by offering a dialog in which you prompt the user to select channels and settings. Our convention is to use the name of the script as the key and the variable name as the name, then start by collecting the values used the last time the script was run, let the user adjust the values in a dialog and then save the new values for use the next time.

Registry use by Spike2

`HKEY_CURRENT_USER\Software\CED\Spike2` holds the following keys:

BarList

This key holds the list of scripts (Script0, Script1...) to load into the script bar when Spike starts and the list of sampling configurations (Sample0, Sample1...) to load into the sample bar. The text associated with each keyword has the format:

```
Title|Path to the .s2s or .s2c file|Comment|code
```

See the `SampleBar()` and `ScriptBar()` commands for more. The `|code` is only used by the Sample bar.

Edit

This key holds the editor settings for scripts, output sequences and general text editing. If you change settings in this key, Spike2 will reread all the edit settings.

PageSetup

This key holds the margins in units of 0.01 mm for printing data views, and the margins in mm and header and footer text for text-based views. If you change settings in this key, Spike2 will reread the printer settings.

Preferences

The values in this key are mainly set by the Edit menu preferences. If you change any Edit menu Preferences value in this key, Spike2 will mark all preference values as unset and the next time it needs to use one, it will be read from the registry. The values are all integers except the file path, which is a string:

`E3 trigger at rear`	0=Sample and `PlayWave` trigger on front panel, 1=on rear
`Enhanced Metafile`	0=Windows metafile, 1=enhanced metafile for clipboard.
`Enter debug on error`	0=Do not enter debug, 1=enter debug

`Error log level`	Talker debug; low values give more information. 0=information, 1=warning (default), 2=error, 3=critical
`Event ports at rear`	0=Events 0&1 on front panel, 1=on rear panel.
`Fast online scroll`	0=Normal, 1=faster (inaccurate) scrolling algorithm
`Fill cursor labels`	0=XOR mode, 1=New filled background mode
`Force idle cycles`	See Edit Preferences Scheduler
`Force idle time`	See Edit Preferences Scheduler
`Global resource file name`	The name of the file, for example "Global"
`Global resource file search`	0=folder that Spike2 ran from, 1=data file folder, then the Spike2 folder, 2=data file folder only
`Global resource if in path`	0 or 1, see global resources
`Global resource if no file`	0 or 1, see global resources
`Global resource path`	The path, see global resources
`Global resource use`	0 or 1, see global resources
`Ignore DEFAULT configuration`	0=Normal (no ignore), 1=ignore `default.s2cx` if found so `last.s2cx` is used instead.
`Ignore resource X range`	0=don't ignore, 1=ignore
`Import configuration`	The Importer configuration string from the Import Options dialog.
`Last 1401 range`	The input range of the last 1401 seen. 5000 for +- 5V, 10000 for +- 10 V.
`Last import DLL`	Last used file import DLL (added at [10.14]).
`Line thickness codes`	Bits 0-3 = Axis code, bits 4-7 = Data code. The codes 0-15 map onto the 16 values in the drop down list. Bit 7=1 to use lines not rectangles to draw axes.
`Log axis default decades`	Number of decades to display in log mode in range 2-15
`Low channels at top`	0=Normal; Standard display shows low channel at bottom, 1=at top.
`Maximum accuracy for text export`	0=Standard accuracy, 1=exact (can be a lot of digits)
`Maximum Log lines`	Maximum lines for display in the Log view or 0 for no limit.
`Minimum channels in imported file`	Minimum number of channels when you import a data file. Setting a value less than 32 has no effect as this is the minimum.
`Minimum dot size for round dots`	Minimum dot size (0 upwards) to draw dots as a circle, not a square.
`Minimum Recycle time`	Minimum time, in seconds, for an unsaved time view file to be moved to the Recycle bin when closed.
`Metafile NoCompress`	0=Compress when multiple points per x pixel, 1=no compress.
`Metafile scale`	0-11 selects from the list of allowed scale factors.
`New file path`	New data file directory or blank for current folder.
`No background colour check`	0=colours should contrast with background, 1=no check
`No flicker-free drawing`	0=normal, 1=force old, see Edit Preferences, Compatibility
`No Power spectrum overlap`	0=normal, 1=force old, see Edit Preferences, Compatibility
`No save prompt`	0=Prompt to save derived views, 1=no prompt.
`No show toolbars on sample`	0=show, 1=if not running from a script, don't force sample toolbars on
`No tetrode data rescaling`	0=normal, 1=force old, see Edit Preferences, Compatibility
`No Y axis invert on drag`	0=allow invert, 1=no allow
`Old forward ref syntax`	0=strict syntax, 1=lax syntax to match [10.14]. Allow forward references to have no `';'` termination, e.g: `Text() halt; Proc Test() end;` See Edit Preferences, Compatibility
`Old SUMMARY format`	0=new format, 1=old, see Edit Preferences, Compatibility
`Output reset ...`	Part of the Output Reset system (many values)
`Save modified scripts`	0=Do not save, 1=save
`Script reference arguments need not match`	0=they must match, 1= they need not for backwards compatibility
`Use system locale for text`	0=normal, 1=use locale when formatting numbers (for example use comma as decimal point)
`Ten volt 1401`	0=5 Volt 1401, 1=10 Volt 1401, 2=same as last 1401.

Text print mode	0=Screen colours, 1=Invert light, 2=Black on White, 3=Colour on White
Update interval	See Edit Preferences Scheduler
Use ANSI var font	0=normal, 1=use old font in user-defined dialogs for backwards compatibility
Use colour	0=Use Black and White, 1=Use Colour (from the View menu).
Use Direct2D text edit	0=use old methods, 1=use potentially hardware assisted text drawing
Use old colours	0=normal, 1=force old method. Before version [5.04], the colour palette was saved in sampling configuration files; loading a sampling configuration set the colour preferences. We now save the palette in the registry. We will likely remove this mechanism at some point if no-one asks us to preserve it.
Use old event measures	0=normal, 1=force old method. Prior to [5.06], measurements from event based channels did not depend on the drawing mode. We will likely remove this mechanism at some point if no-one asks us to preserve it.
Use old waveform measures	0=normal, 1=force old, see Edit Preferences, Compatibility
Warn on close with memory chans	0=no warning, 1=not if from script, 2=always

We may well add more items that are not in the table. If you cannot find the option you want, set the option in Spike2 and close the program, then use the `regedt32` or `regedit` program to look at the path:

`HKEY_CURRENT_USER\Software\CED\Spike2\Preferences`

You should be able to locate the string used to save the option. You can then use this string to set or clear the option.

Dockable toolbars

The sub-keys with names starting `"Bars-"` are used by system code to restore dockable toolbars. Either delete them all or leave them all alone; any other change is likely to crash Spike2 on start up. If Spike2 does crash on start up, deleting all these sub-keys is a good idea; it does not do any harm (you'll get default control bar positions) and may allow the program to start.

Recent file list

This key holds the list of recently used files that appear at the bottom of the file menu.

Recover

This key holds the information to recover data from interrupted sampling sessions.

Settings

This is mainly used to remember the state of Spike2 between sessions:

Name	Type	Use
Cluster	SubKey	Clustering information.
ColourMap	SubKey	Holds the colour map information for user-defined maps and for the sonogram and clustering maps.
Find	SubKey	The find and replace dialog current strings and last 10 saved strings.
Output	SubKey	The recently used output sequencer files for the Sequencer tab of the Sampling Configuration dialog
VChan	SubKey	The recently used Virtual channel expressions.
Eval0-9	String	The recently used Evaluate bar expressions.
OLTN0-9	String	The On Line TextMark messages. If these end with a vertical bar, followed by 9 hexadecimal digits, the first hex digit has flags in bits 0-3 to display the associated code as an ASCII character and the following 8 hexadecimal digits are to be interpreted as a 32-bit integer holding the 4 associated marker codes. Bits

		0-7 for the first code, 8-15 with the second, 16-23 with the third and 24-31 the fourth.
`* chan filter`	String	The current Time, Result and XY filter keys separated by vertical bars for the Show/Hide Channel dialog.
`Watch1`	String	The last set of variables that were in the Watch debug window, separated by vertical bars.
	Binary	These hold information used internally by Spike2.

Tip
The *Tip of the Day* dialog uses this key to remember the last tip position.

Version
Spike2 uses this key to detect when a new version of the program is run for the first time.

Win32
In Windows NT derived systems, this key holds the desired working set sizes. The Help menu About Spike2 dialog displays the current working set sizes. See the Technical Support: Frequently asked questions section for more information about the Working set and error -544.

Minimum working set	Minimum size in KB (units of 1024 bytes), default is 1000
Maximum working set	Maximum size in KB, default is 8000 (8 MB). You can set much larger sizes, if you wish.

ProgKill()

This function terminates a program started using `ProgRun()`. This can be dangerous, as it will terminate a program without giving it the opportunity to save data.

```
Func ProgKill(pHdl%);
```

`pHdl%` A program handle returned by `ProgRun()`.

Returns Zero or a negative error code.

See also:
`ProgRun(), ProgStatus()`

ProgRun()

This function runs a program using command line arguments as if from a command prompt. Use `ProgStatus()` to test the program status, `ProgKill()` to terminate it. The program inherits its environment variables from Spike2, see `System$()` for details.

```
Func ProgRun(cmd$ {,code% {,xLow, yLow, xHigh, yHigh}});
```

`cmd$` The command string as typed at a command prompt. To run shell command `x` use `"cmd /c x"`. To run another copy of Spike2, use `"sonview.exe /M {filename}"`. We suspect that the longest command line that will work is 259 characters (despite the Microsoft documentation for the system call we use stating that it works for up to 32767 characters)

`code%` If present, this sets the initial application window state: 0=Hidden, 1=Normal, 2=Iconized, 3=maximised. Some programs set their own window state so this may not work. The next 4 arguments set the Normal window position:

`xLow` Position of the left window edge as a percentage of the screen width.

`yLow` Position of the top window edge as a percentage of the screen height.

`xHigh` The right hand edge as a percentage of the screen width.

yHigh The bottom edge position as a percentage of the screen height.

Returns A program handle or a negative error code. ProgStatus() releases resources associated with the handle when it detects that the program has terminated.

A successful return only means that the program has been launched without a problem. It does not mean that the program is ready for use. For example, if you use this command to launch the s2video application and then immediately start recording data, video is unlikely to be recorded as it takes a while for the program to start up and link itself to Spike2. In this particular case you can use the Listener() command to count the number of attached listener devices.

See also:
FileCopy(), FileDelete(), ProgKill(), ProgStatus(), System$()

ProgStatus()

This function tests if a program started with ProgRun() is still running. If it is not, resources associated with the program handle are released.

```
Func ProgStatus(pHdl%);
```

pHdl% The program handle returned by ProgRun().

Returns 1=program is running, 0=terminated, resources released, handle now invalid. A negative error code (-1525) means that the handle is invalid.

See also:
ProgKill(), ProgRun()

Q

Query()

Query()

This function is used to ask the user a Yes/No question. It opens a window with a message and two buttons. The window is removed when a button is pressed.

```
Func Query(text$, {,Yes$ {,No$}});
```

text$ This string forms the text in the window. If the string includes a vertical bar, the text before the vertical bar is used as the window title. There is no limit on the length of the text string you use here, but there is a limit on the space it can occupy in the Query dialog. You are allowed up to 80 dialog units wide (about 80 wide characters) and up to 40 dialog units high (at least 40 lines). To make use of this you must split the text into lines using \n to insert a new line. Alternatively, you can allow Spike2 to split up the lines, but this may not achieve the best possible results.

Yes$ This sets the text for the first button. If this argument is omitted, "Yes" is used.

No$ This sets the text for the second button. If this is omitted, "No" is used.

Returns 1 if the user selects Yes or presses Enter, 0 if the user selects the No button.

The following example generates a 4 line query with replacements for the Yes and No buttons.

```
Query("0123456789 123456789 123456789 123456789 123456789 123456789 123456789 \n"
      "0123456789 123456789 123456789 123456789 123456789 123456789 123456789 \n"
      "0123456789 123456789 123456789 123456789 123456789 123456789 123456789 \n"
      "0123456789 123456789 123456789 123456789 123456789 123456789 123456789 \n",
      "Oui", "Non")
```

See also:
Print(), Input(), Message(), DlgCreate(), DlgFont()

R

```
Rand()
RandExp()
RandNorm()
RasterAux()
RasterGet()
RasterSet()
RasterSort()
RasterSymbol()
Read()
ReadSetup()
ReadStr()
ReRun()
Replace$
Reverse$()
Right$()
Round()
```

Rand()

This returns pseudo-random numbers with a uniform density in a set range. The values returned are R * `scl` + `off` where R is in the range 0 up to, but not including, 1. Spike2 initialises the generator with a random seed based on the time. You must set the seed for a repeatable sequence. The sequence is independent of `RandExp()` and `RandNorm()`.

```
Func Rand(seed);
Func Rand({scl, off});
Func Rand(arr[]{[]...}{, scl{, off}});
```

`seed` If present, this is a seed for the generator in the range 0 to 1. If `seed` is outside this range, the fractional part of the number is used. If you use 0.0 as the seed, the generator is initialised with a seed based on the system time.

`arr` This real or integer array is filled with random numbers. If an integer array is used, the random number is truncated to an integer.

`scl` This scales the random number. If omitted, it has the value 1.

`off` This offsets the random number. If omitted it has the value 0.

Returns If the first argument is not an array, the return value is a random number in the range `off` up to `off+scl`. If a seed is given, a random number is still returned. If the first argument is an array, the return value is 0.0.

See also:
`RandExp(), RandNorm()`

RandExp()

This function returns pseudo-random numbers with an exponential density, suitable for generating Poisson statistics. The values returned are ***R*** * `mean` + `off` where ***R*** is a random number with the density function $p(x) = \exp(-x)$. When you start Spike2, the generator is initialised with a random seed based on the time. For repeatable sequences, you must set a seed. The sequence is independent of `Rand()` and `RandNorm()`.

```
Func RandExp(seed);
Func RandExp({mean, off});
Func RandExp(arr[]{[]...}{, mean{, off}});
```

`seed` If present, this is a seed for the generator in the range 0 to 1. If `seed` is outside this range, the fractional part of the number is used. If you use 0.0 as the seed, the generator is initialised with a seed based on the time.

`arr` This real or integer array is filled with random numbers. If an integer array is used, the random number is truncated to an integer.

`mean` This scales the random number. If omitted, it has the value 1.

`off` This offsets the random number. If omitted it has the value 0.

Returns If the first argument is not an array, the return value is a random number. If a seed is given, a random number is still returned. If the first argument is an array, the return value is 0.0.

The following example fills an array with event times with a mean interval `t`:

```
RandExp(arr[], t);          'Fill arr with event intervals
ArrIntgl(arr[]);            'convert intervals to times
```

See also:
`Rand(), RandNorm()`

RandNorm()

This function returns pseudo-random numbers with a normal density. The values returned are ***R*** * `scl` + `off` where ***R*** is a random number with a normal probability density function $p(x)$ = `exp(-x*x/2)/sqrt(2*pi);` this has a mean of 0 and a variance of 1. When you start Spike2, the generator is initialised with a random seed based on the time. For a repeatable sequence, you must set a seed. The sequence is independent of `Rand()` and `RandExp()`.

```
Func RandNorm(seed);
Func RandNorm({scl, off});
Func RandNorm(arr[]{[]...}{, scl{, off}});
```

`seed` If present, this is a seed for the generator in the range 0 to 1. If `seed` is outside this range, the fractional part of the number is used. If you use 0.0 as the seed, the generator is initialised with a seed based on the time.

`arr` This real or integer array is filled with random numbers. If an integer array is used, the random number is truncated to an integer.

`scl` This scales the random number. If omitted, it has the value 1.

`off` This offsets the random number. If omitted it has the value 0.

Returns If the first argument is not an array, the return value is a random number. If a seed is given, a random number is still returned. If the first argument is an array, the return value is 0.0.

See also:
`Rand(), RandExp()`

RasterAux()

This is now deprecated; use `RasterSort()` or `RasterSymbol()` instead. It is here for version 4 compatibility, but will be removed in a future release. This function returns and optionally sets some of the raster auxiliary values for a channel in the current result view.

Values 0 and 1 are used to sort the sweeps and can be selected in addition to time order by the `DrawMode()` command. Unset values read back as zero.

Values 2 to 5 are the times, in seconds relative to the start of the file, of four markers (circle, cross, square and triangle) to display in the sweep. The markers are drawn in the colours set for WaveMark codes 1 to 4. Markers appear if their position relative to the sweep trigger lies within the sweep. Unset values read back as a negative time.

```
Func RasterAux(chan%, sweep%, num% {,new});
```

`chan%` The channel in the result view. The channel must have raster data enabled.

`sweep%` The sweep number in the range 1 to the number of sweeps in the channel.

`num%` The auxiliary value to return and optionally change in the range 0 to 5.

`new` If present, this changes the auxiliary value for the sweep.

Returns The auxiliary value at the time of the call.

Replace `RasterAux(c%,s%,n%{,new})` with `RasterSort(c%,s%,n%{,new})` if `n%` is 0 or 1, and with `RasterSymbol(c%,s%,n%-2{,new})` if `n%` is 2 to 5.

See also:
`DrawMode()`, `RasterGet()`, `RasterSet()`, `RasterSort()`, `RasterSymbol()`, `SetEvtCrl()`, `SetPSTH()`, `SetResult()`

RasterGet()

This returns a sweep of raster data for a result view channel for which raster data has been enabled. Using this when the current view is not a result view causes a fatal error.

```
Func RasterGet(chan%{,sweep%{,&sT{,data{%}[]{,&tick{,&eT}}}}});
```

`chan%` The channel number in the result view. If this is the only argument, the return value is the number of sweeps in the channel.

`sweep%` The sweep number in the result view in the range 1 to the number of sweeps. If this argument is present, the return value is the number of times in the sweep.

`sT` This is returned holding the time in seconds of the start of the sweep in the original data. This is not the sweep trigger time (the time that corresponds to the x axis 0 in the result view). The trigger time is `sT-BinToX(0)`.

`data` This optional array is filled with sweep event times. A real array returns times in seconds; an integer array returns times in units of the microseconds per time used when the view was created. The number of times copied into the array is the lesser of the number of times in the sweep and the length of the array.

`tick` This optional real value is the number of seconds per time units used when times are returned as an integer array. That is, if you multiply each integer time returned in `data%[]`, you would get the time in seconds. If you created the result view using `SetResult()` it is the tick value you passed in, otherwise it is the `BinSize()` value from the associated time view.

`eT` This optional real value is returned holding the end time of the sweep, in seconds. This is usually only of interest for data created with `SetPhase()`.

Returns The count of sweeps in the channel, times in the sweep or a negative error code.

See also:
`RasterSet()`, `RasterSort()`, `RasterSymbol()`, `SetEvtCrl()`, `SetPhase()`, `SetPSTH()`, `SetResult()`

RasterSet()

This function sets the raster data for a sweep for a channel of a result view. You can replace the data for an existing sweep, or add a new sweep. It is a fatal script error to call this function when the current view is not a result view. From [10.11] this can set an empty sweep and invalidates any raster display.

```
Func RasterSet(chan%, sweep%, sT {,const data[]|const data%[] {,eT}});
```

`chan%` The channel number in the result view.

`sweep%` This is either in the range 1 to `Sweeps()` to replace the existing raster data for a sweep, or it can be 0 to add a new sweep. If you add a new sweep, the new sweep is added to all channels. The remaining channels have a sweep added that has the same start time, and no data. If your result view has multiple channels, only use a `sweep%` value of 0 for one of the channels.

`sT` This sets the time of the start of the sweep, in seconds. This is not the trigger time of the sweep (the time that corresponds to the x axis 0 in the result view). The trigger time is `sT-BinToX(0)`.

`data` This is a real or an integer array holding the events times for the sweep. The size of the array sets the number of items. If this is a real array, the times are in seconds. If this is an integer array, the times are

in the underlying tick units (see RasterGet() and SetResult() for details). Times outside the time range start to start+MaxTime()*BinSize() are ignored.

eT This optional value is the end time of the sweep. If you supply this, all times in the data array are mapped into the time period sT to eT, regardless of the x axis scaling set for the result view. This argument is usually only used when working with or emulating a Phase histogram.

Returns The sweep number that received the data or a negative error code.

See also:
RasterGet(), RasterSort(), RasterSymbol(), SetEvtCrl(), SetPhase(), SetPSTH()

RasterSort()

Each raster sweep has 4 values that can be used to sort displayed rasters; this function returns and optionally sets these values for the current result view. The DrawMode() command selects between time order and one of these values.

```
Func RasterSort(chan%, sweep%, num% {,new});
```

chan% The channel in the result view. The channel must have raster data enabled.

sweep% The sweep number in the range 1 to the number of sweeps in the channel.

num% The sort value to return and optionally change in the range 1 to 4.

new If present, this changes sort value num% for the sweep.

Returns The sort value at the time of the call. Unset values read back as zero.

See also:
DrawMode(), RasterGet(), RasterSet(), RasterSymbol(), SetEvtCrl(), SetPSTH(), SetResult()

RasterSymbol()

Each raster sweep has 8 values that can be displayed as symbols; this function returns and optionally sets these values for the current result view. The values are times, in seconds relative to the start of the file. The 8 markers are: circle, cross, square, up triangle, plus, diamond, down triangle, filled square. The markers are drawn in the colours set for WaveMark codes 1 to 8. Markers appear if their position relative to the sweep trigger lies within the sweep. Unset values read back as a negative time.

```
Func RasterSymbol(chan%, sweep%, num% {,new});
```

chan% The channel in the result view. The channel must have raster data enabled.

sweep% The sweep number in the range 1 to the number of sweeps in the channel.

num% The symbol number in the range 1 to 8 to get or set.

new If present, this sets the symbol time for the sweep. Set -1 to cancel the symbol.

Returns The symbol time at the time of the call. Unset symbols have a negative time.

See also:
DrawMode(), RasterGet(), RasterSet(), SetEvtCrl(), SetPSTH(), SetResult(), SetResult()

Read()

This function reads the next line from the current text view or external text file and converts the text into variables. The read starts at the beginning of the line containing the text cursor. The text cursor moves to the start of the next line after the read.

```
Func Read({&var1 {, &var2 {,&var3 ...}}});
```

`varn` Arguments must be variables. They can be any type. One dimensional arrays are allowed. The variable type determines how to convert the string data. In a successful call, each variable matches a field in the string, and the value of the variable changes to the value found in the field. A call to `Read()` with no arguments skips a line.

Returns The function returns the number of fields in the text line that were successfully extracted and returned in variables, or a negative error code. Attempts to read past the end of the file produce the end of file error code. If the first variable is a string, a return value of 0 means that a blank line was read.

It is not an error to run out of data before all the variables have been updated. If this is a possibility you must check that the number of items returned matches the number you expected. An array of length `n` is treated as `n` individual values.

Normally, the fields in the source string are separated by white space (tabs and spaces) and commas. Space characters are "soft" separators. You can have any number of spaces between fields. Tabs and commas are treated as "hard" separators. Two consecutive hard separators (with or without intervening soft separators), imply a blank field. You can use`ReadSetup()` to redefine the soft and hard separators. When reading a field, the following rules are followed:

1. Soft separator (space) characters are skipped over. If a hard separator is found or end of line, the field is empty (0 for a number, a blank string).
2. If the field is a string and the next character is a delimiter, it is skipped.
3. Characters that are legal for the destination variable are extracted until a non-legal character or a separator or a required string delimiter or end of data is found. The characters read are converted into the variable type. If an error occurs in the translation, the function returns the error. Blank fields assigned to numbers are treated as 0. Blank fields assigned to strings produce empty strings.
4. Characters are skipped until a separator character is found or end of data. If a soft separator is found, it and any further soft separators are skipped. If the next character is a hard separator it is also skipped.
5. If there are no more variables or no more data, the process stops, else back to step 1.

The source string is expected to hold data values as:

Type	Examples	Description

Real	`1.23e-23`
	`-23`
	`1:23:12:58.23`
	`12:34.56`
		A number with an optional decimal point and an optional exponent. The number can start with a minus sign.
		From version 8.06 you can read a time as `days:hours:minutes:seconds`. You may omit the days, or days and hours. The time is converted to seconds. Times of more than 100 years are not recognised.

Integer	`123, -1234567` `0x1234abcd`	A 64-bit integer value formatted as an optional minus sign followed by a list of decimal digits. It also accepts a hexadecimal value, being 0x followed by 1 to 16 hexadecimal digits (0-9, a-f, A-F).
String	`Some text` `"Quoted text"`	Strings can be delimited by double quote marks, which allows the strings to hold separators and to be followed by other fields. If a string is not delimited, it is deemed to run to the end of the source string, so no other items can follow it unless you set a hard string separator with `ReadSetup()`. You can use `ReadSetup()` to change the characters that delimit a string and also to define hard separator characters for non-delimited strings. String delimiters are not returned as part of the string.

Reading exact contents of text file

If you want to use `Read()` to fill a string with the line by line contents of a text file, including leading white space, you should call `ReadSetup("","")` before using `Read()`.This cancels all soft separators, which allows leading white space (spaces and Tab characters) to be read.

Example

The following example shows a source line, followed by a `Read()` function, then the assignment statements that would be equivalent to the `Read()`:

```
"This is text"    ,  2 3 4:23,, 4.56 Text too 3 4 5      The source line
n := Read(fred$, jim[1:2], sam, dick%, tom%, sally$, a, b, c);
```

is equivalent to:

```
n := 7;
fred$ := "This is text";
jim[1] := 2; jim[2] := 3; sam := 263; dick% := 0; tom% := 4;
sally$ := "Text too 3 4 5"
'a, b and c are not changed
```

The following script opens a text file and writes it line by line to the Log view (including leading white space on each line):

```
var fh%, line$;
fh% := FileOpen("*.txt", 8, 0,"Select a text file");   'open a file as text
if (fh% > 0) then                ' if we opened a file...
    ReadSetup("","");            ' cancel soft separators
    while Read(line$) >= 0 do    ' read while not EOF or error
        PrintLog("%s\n", line$); ' write whatever we got
    wend;
    ReadSetup();                 ' restore standard separators
    FileClose();                 ' we are done with the file
endif;
```

See also:
`EditCopy()`, `FileOpen()`, `ReadSetup()`, `ReadStr()`, `Selection$()`

ReadSetup()

This sets the separators and delimiters used by `Read()` and `ReadStr()` to convert text into numbers and strings. You can also set string delimiters and set a string separator. This applies globally, so if you have two input file streams open, it applies to both. It is possible that in the future we will devise a scheme to apply this to selected file streams.

```
Proc ReadSetup({hard${, soft${, sDel${, eDel${, sSep$}}}}});
```

`hard$` The characters to use as hard separators between all fields. If this is omitted or the string is empty, the standard hard separators of comma and tab are used.

`soft$` The characters to use as soft separators. If this is omitted, the space character is set as a soft separator. If `soft$` is empty, no soft separators are used.

`sDel$` The characters that delimit the start of a string. If omitted, a double quote is used. If empty, no delimiter is set. Delimiters are not returned in the string.

`eDel$` The characters that delimit the end of a string. If omitted, a double quote is used. If empty, no delimiter is set. If `sDel$` and `eDel$` are the same length, only the end delimiter character that matches the start delimiter position is used. For example, to delimit strings with `<text>` or `'text'` set `sDel$` to `"<'"` and `eDel$` to `">'"`. You can repeat a character to force different lengths.

`sSep$` The list of hard separator characters for strings that have no start delimiter. For example, setting `"|"` lets you read `one|two|three` into three separate strings. This can be the same as the characters used as hard separators set by `hard$`.

See also:
`Read()`, `ReadStr()`, `Val()`

ReadStr()

This function extracts data fields from a string and converts them into variables.

```
Func ReadStr(text$, &var1 {, &var2 {, &var3...}});
```

`text$` The string used as a source of data.

`var` The arguments must all be variables. The variables can be of any type, and can be one dimensional arrays. The type of each variable determines how the function tries to extract data from the string. See `Read()` for details.

Returns The function returns the number of fields in the text string that were successfully extracted and returned in variables, or a negative error code.

It is not an error to run out of data before all the variables have been updated. If this is a possibility you must check the returned value. If an array is passed in, it is treated as though it was the number of individual values held in the array.

See also:
`Read(), ReadSetup(), Val()`

ReRun()

This function controls the rerun of the current time view and is equivalent to the View menu ReRun command. You cannot use this on a file that is being sampled.

```
Func ReRun({run%{, sTime{, eTime{, scale}}}});
```

`run%` Set 1=start rerun, 0=stop rerun or omit for no change. Negative values return: -1=`sTime`, -2=`eTime`, -3=`scale`, -4=the rerun time or -1 if not rerunning.

`sTime` Sets the rerun start time. If omitted, 0 is used. This is ignored unless `run%` > 0.

`eTime` Sets the rerun end time. If omitted `MaxTime()` is used. Ignored unless `run%` > 0.

`scale` Sets the rerun time scale. If omitted, 1.0 is used. A value of 2 reruns twice as fast. Values from 0.01 to 100.0 are allowed. This is ignored unless `run%` > 0.

Returns If `run%` is positive, omitted or less than -4, the command returns the state at the time of the call: 0=not rerunning, 1=rerunning, 2=rerunning linked to a play offline waveform output, -1= rerunning is not allowed. Negative values of `run%` return the rerun settings as described for `run%`.

See also:
View menu Rerun, `PlayOffline()`

Replace$

Search a string for sub-strings and replace all occurrences with a replacement string.

```
Func Replace$(text$, find$, rep${, &n%});
```

`text$` The text to search.

`find$` The string to find in `text$`.

`rep$` The string to replace each occurrence of the `find$` text.

`n%` If present, returned set to the number of replacements made.

Returns The result of the replacements.

Beware that this can generate a huge string. We do not protect you against code such as the following that will likely run out of memory:

```
var n% := 100000;
var s$ := "x"*n%;   '100,000 'x' characters
var big$ := Replace$(s$, "x", "y"*n%); '10,000,000,000 'x' characters
```

If you want to replace text found with a regular expression, you could consider using the `InStrRE()` function to locate matches, then replace them in reverse order (so the replacements do not invalidate the character offsets and ranges found).

See also:
`Asc(), Chr$(), DelStr$(), InStr(), LCase$(), Left$(), Len(), Mid$(), Print$(), Print(), Reverse$(), Right$(), Str$(), UCase$(), Val()`

Reverse$()

Reverse a text string. This pays attention to any Unicode characters that use surrogate pairs of 16-bit characters. New at Spike2 version [10.04]. You can reverse the order of strings in an array with `ArrRev()`.

```
Func Reverse$(in$);
```

`in$` The input string to reverse.

Return A string of the same length as `in$` with the characters in the reverse order.

See also:
`Asc(), Chr$(), DelStr$(), InStr(), LCase$(), Left$(), Len(), Mid$(), Print$(), Print(), Replace$(), Right$(), Str$(), UCase$(), Val()`

Right$()

This function returns the rightmost `n` characters of a string.

```
Func Right$(text$, n%);
```

`text$` A string of text.

`n%` The number of characters to return.

Returns The last `n%` characters of the string, or all the string if it is less than `n%` characters long.

Unicode
In a Unicode build of Spike2 the underlying characters are stored in a vector of 16-bit values and `n%` is the number of the trailing 16-bit values to return. Most common Unicode characters fit in one 16-bit value, but some require two consecutive values, leading to the possibility that you might attempt to return half an extended character. The rule we apply is that if the start or end of any string section falls between the lead and trail codes of a Unicode surrogate pair, we move the start or end on by one position, to the start of the next character or the end of the string. In the case of `Right$()`, we reduce `n%` by 1 if the returned string would start half-way through a character (moving the start point on by one 16-bit value).

See also:
`Asc(), Chr$(), DelStr$(), InStr(), LCase$(), Left$(), Len(), Mid$(), Print$(), ReadStr(), Replace$(), Reverse$(), Str$(), UCase$(), Val()`

Round()

Rounds a real number or an array of reals to the nearest whole number. Values exactly half-way between two integral values round away from zero.

```
Func Round(x|x[]{[]...});
```

`x` A real number or an array of reals.

Returns If `x` is an array it returns 0. Otherwise it returns a real number with no fractional part that is the nearest to the original number.

`Round(x)` is equivalent to:

```
Func Round(x) return (x<0.0) ? Ceil(x-0.5) : Floor(x+0.5) end
```

See also:
`Abs(), ATan(), Cos(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Sin(), Sqrt(), Tan(), Trunc()`

S

SampleAbort()
SampleAutoComment()
SampleAutoCommit()
SampleAutoFile()
SampleAutoName$()
SampleBar()
SampleBigFile()
SampleCalibrate()
SampleChanInfo()
SampleChannels()
SampleClear()
SampleComment$()
SampleConfig$()
SampleDebounce()
SampleDigMark()
SampleEvent()
SampleHandle()
SampleIdle()
SampleInfo()
SampleKey()
SampleKeyMark()
SampleLimitSize()
SampleLimitTime()
SampleMode()
SampleOptimise()
SampleProcess()
SampleRepeats()
SampleReset()
SampleSeqClock()
SampleSeqCtrl()
SampleSeqStep()
SampleSeqTable()
SampleSeqTick0
SampleSequencer()
SampleSequencer$()
SampleSeqVar()
SampleStart()
SampleStartTrigger()
SampleStatus()
SampleStop()
SampleTalker()
SampleText()
SampleTextMark()
SampleTimePerAdc()
SampleTitle$()
SampleTrigger()
SampleUsPerTime()
SampleWaveform()
SampleWaveMark()
SampleWrite()
ScriptBar()
ScriptRun()
Seconds()
Selection()
Selection$()
SerialClose()
SerialCount()
SerialOpen()
SerialRead()
SerialWrite()
Set...()
SetAverage()
SetEvtCrl()
SetEvtCrlShift()
SetINTH()
SetPhase()
SetPower()
SetPSTH()
SetResult()
SetWaveCrl()

Sample...()

All the commands with names that start with Sample... are either relates to setting up a sampling configuration for sampling or are related to the currently sampling data file.

SampleProcess()
SampleRepeats()
SampleReset()
SampleScript()
SampleSeqClock()
SampleSeqCtrl()
SampleSeqStep()
SampleSeqTable()
SampleSeqTick0
SampleSequencer()
SampleSequencer$()
SampleSeqVar()
SampleStart()
SampleStartTrigger()
SampleStatus()
SampleStop()
SampleTalker()
SampleText()
SampleTextMark()
SampleTextMarkLink()
SampleTimePerAdc()
SampleTitle$()
SampleTrigger()
SampleUsPerTime()
SampleWaveform()
SampleWaveMark()
SampleWrite()

SampleAbort()

This cancels sampling a file or file sequence and closes any associated data, result and cursor views without saving them. If a result view derived from the data has been saved, the saved file remains. It is equivalent to the Abort button in the sample control toolbar.

```
Func SampleAbort();
```

Returns 0 if sampling was aborted, or a negative error code.

See also:
`SampleReset(), SampleStart(), SampleStop(), SampleStatus()`

SampleAutoComment()

This gets or sets file auto-commenting state as set in the sampling configuration dialog.

```
Func SampleAutoComment({yes%});
```

`yes%` If present a non-zero value turns on automatic prompting for file comments when sampling ends and zero turns it off. If absent, no change is made.

Returns the automatic commenting flag at the time of the function call, 0=off, 1= on.

See also:
`FileComment$(), SampleAutoFile(), SampleAutoName$()`

SampleAutoCommit()

This gets or sets the automatic file commit period as set in the sampling configuration dialog. There is an overhead associated with the commit operation; do not set needlessly short time periods. A period of 0 means no file commit. Note that you can request a commit at any time during sampling with the `Modified(0,0)` command.

```
Func SampleAutoCommit({every%});
```

every% If present this sets the automatic file commit interval in seconds. The value is limited to the range 0 to 1800 seconds (a maximum of 30 minutes). From [10.07] you can set -1 for Write through disk mode, which requests that the disk system writes all data to the drive immediately. Write through mode should be slower (on average), but can reduce the gap between files in a sequence when sampling at a high rate as flushing the file when it closes can take a long time (many seconds) to write all the buffered data from system memory to the disk.

Returns the automatic commit period in seconds at the time of the function call.

See also:
FileSave(), Modified(), SampleAutoFile(), SampleWrite()

SampleAutoFile()

This gets or sets the automatic filing state as set in the sampling configuration dialog, equivalent to the Save file at end of sampling box. Note that you must fulfil all the conditions (file name template, path, stopping condition, repeats and save at end of sampling) before automatic writing of a sequence of files will occur.

```
Func SampleAutoFile({yes%});
```

yes% If present, a non-zero value turns on automatic data filing when sampling ends and zero turns it off. If absent, no change is made.

Returns the automatic filing state at the time of the function call, 0=off, 1= on.

See also:
FileSave(), FilePathSet(), SampleAutoName$(), SampleAutoComment(), SampleRepeats()

SampleAutoName$()

This gets or sets the template for file auto-naming as in the sampling configuration dialog. The path is set by FilePathSet(). The name must be set before opening the file for sampling for the automatic name to be used.

```
Func SampleAutoName$({name$});
```

name$ If present, this is the template for file auto-naming. Leading and trailing white space is removed, after which an empty string turns off auto-naming. The name template is limited to 40 characters from Spike2 version 9.01, older versions limited it to 23 characters. There is no limit (apart from what the operating system can handle) on the length when the template is expanded. There is currently no check that the suggested name holds legal file name characters. If it does not, attempts to use the name will fail. See the sampling configuration documentation for details on the template string.

Returns The auto-naming template at the time of the function call.

See also:
FilePathSet(), SampleAutoFile(), SampleAutoComment()

SampleBar()

This gives you access to the Sample toolbar. It allows you to set a string to be associated with a button and to read back information about the existing buttons. If you call the command with no arguments it returns the number of buttons in the toolbar. Sample bar strings are saved in the system registry; see the Profile() command for more information.

```
Func SampleBar({n% {, &get$}});
Func SampleBar(set$);
Func ScriptBar(-1{, match$});
```

n% If set to -1, get$ must be omitted, all buttons are cleared and the function returns 0. When set to the number of a button (the first button is 0), get$ is as described below. In this case, the function returns

-1 if the button does not exist, 0 if it exists and is the last button, and 1 if higher-numbered buttons exist.

`set$` The string passed in should have the format described above. The function returns the new number of buttons or -1 if all buttons are already used.

`match$` If present, identifies the button(s) to remove. If the first argument is -1 and `match$` is omitted or empty, all buttons are cleared. Otherwise, `match$` is a case insensitive regular expression to match. You are attempting to match the same string as was used to set the button, so you can attempt to match the button text or the file name or the tool tip, or a combination. The function returns the number of deleted buttons. Clearing a matched button was added at [10.15].

Returns See the descriptions below. Negative return values indicate an error.

Sample bar string format

The string format is a set of up to 4 fields separated by vertical bars. The following formats are acceptable (and the label, comment or code fields can be blank).

```
path
label|path
label|path|comment
label|path|comment|code
```

If no vertical bars are found, the string is presumed to be the path to the configuration file (with extension `.s2c` or `.s2cx`). If there is one vertical bar, it is presumed to separate the label for the button from the path. If there are two vertical bars, a comment is presumed and if three, a code is also presumed to exist. So the following combinations are legal:

```
path
label|path
label|path|comment
label|path|comment|code
```

The code is a number in the range 0 to 3, being the sum of the flag values 1 and 2. If omitted, it is presumed to be 0:

1 Immediate start. If set (1), start sampling as soon as the Sample Bar button is clicked. If not set (0), the user must click the Start button in the Sample control toolbar.

2 Write to Disk. This only has an effect if the immediate start bit is set. If set, start sampling with write to disk enabled, else start with write to disk disabled.

When the string is read back by the `SampleBar()` command or written to the system registry, trailing blank fields and the associated vertical bar are removed.

For example, the following code clears the script bar and sets two buttons:

```
SampleBar(-1);    'clear all buttons
SampleBar("Fast|C:\\Spike\\Fast.s2c|Fast 4 channel sampling");  'No immediate start
SampleBar("Faster|C:\\Spike\\FastXX.s2c|Very fast sampling|1"); 'Immediate start, write disabled
```

The following deletes the `"Fast"` button, leaving the `"Faster"` one:

```
SampleBar(-1, "Fast.s2c");
```

See the `ScriptBar()` command for discussion of deleting a button and unexpected problems using regular expressions. For example, the following could also be used to delete the `"Fast"` button:

```
SampleBar(-1, "^Fast\\|");   'See ScriptBar() for an explanation
```

See also:

`App()`, Sample toolbar, `Profile()`

SampleBigFile()

This function returns and optionally sets the sample file type in the sampling configuration. This is equivalent to the File type field in the Sampling Configuration dialog. The function name comes about because we have taken over the function call used for setting the *Big File* flag in Spike2 version 7.

```
Func SampleBigFile({type%});
```

`type%` If present, this sets the type of file to create:

- 0 Create an old-style 32-bit data file (`*.smr`) with a maximum size of 2 GB.
- 1 Create a 32-bit data file (`*.smr`) with a maximum size of 1 TB.
- 2 Create a 64-bit data file (`*.smrx`) with a size limited only by the operating system.

Returns The file type set in the sampling configuration at the time of the call.

See also:
`FileNew()`, `FileSaveAs()`, `SampleClear()`

SampleCalibrate()

This sets or gets the waveform or WaveMark channel calibration. It changes the units, scale and offset fields only. See the Sampling Configuration dialog for a description of these fields. When used to read back information, the `unit$`, `scale` and `offset` arguments must be variables:

```
Func SampleCalibrate(chan%, units$, scale, offset{, get%});
Func SampleCalibrate(chan%, &units$, &scale, &offset, 1);
```

`chan%` The channel number of a waveform or WaveMark channel.

`units$` The new channel units (up to 5 characters) or a string variable to read the units.

`scale` The new scale factor for the channel or a real variable to read back the scale.

`offset` The new offset for the channel or a real variable to read back the offset.

`get%` Omit or set to 0 to set the calibration, set to 1 to read it back.

Returns 0 if all was well, or a negative error code if the channel is the wrong type.

See also:
`SampleTitle$()`, `SampleUsPerTime()`, `SampleWaveform()`, `SampleWaveMark()`

SampleChanInfo()

This function returns channel information that is common to most channels from the current sampling configuration. It also reports the fixed channel numbers that are associated with the special channels for keyboard, text and digital markers.

```
Func SampleChanInfo(chan%{, item%{, trace%}});
```

`chan%` A channel number in the sampling configuration in the range 1-100. The value 0 is reserved. Negative values return the channel numbers of special channels (see `SampleChannels()` to set the special channel numbers):

- -1 the Keyboard channel
- -2 the TextMark channel
- -3 the Digital marker channel.

`item%` If omitted, this takes the value 0, otherwise it determines the information to return. When used with `chan%` greater than 0 the information returned is:

0 Channel type. This is the same as returned by `ChanKind()` for all channels except Talker and Derived channels. The code 1007 is a RealWave channel with source data as double precision, 1009 is RealWave with the source data as single precision:

Type	Standard	Talker	Derived
Unused	0		
Adc	1	1001	
Event(-)	2	1002	
Event(+)	3		
Level	4	1008	
Marker	5	1003	

WaveMark	6	1004	
RealMark	7	1005	
TextMark	8	1006	
RealWave	9	1007	200
		1009	

1 Port number used by the channel (uses `trace%` for WaveMark channels, if supplied).

2 Expected event rate (same as waveform rate for waveform channel).

3 Ideal waveform rate (same as event rate for event channel). Items 2 and 3 differ for WaveMark channels.

4 Actual waveform rate for waveform and WaveMark channels, same as ideal for others. Prior to version [10.20] this returned 0 for Talker channels.

When used with negative value of `chan%`:

0 Return the special channel number even if it is not used.

1 Return the special channel number if it is used, or 0 if it is not used

`trace%` Optional, taken as 0 if omitted. The trace index for WaveMark channels in the range 0 to the number of traces -1.

Returns The information requested or a negative error code.

See also:
`ChanKind()`, `SampleCalibrate()`, `SampleChannels()`, `SampleTextMark()`, `SampleWaveMark()`

SampleChannels()

This function sets and gets the maximum number of channels in the data file and the position of the special channels (Keyboard marker, Text Marker and Digital Marker). The next time you sample, the data file will have space for this number of channels. This command is equivalent to the Set channels in new data files dialog.

You should use this command after `SampleClear()` and before you define your channels for sampling. If you use this after defining channels, we attempt to achieve what you ask with the least destruction of existing channels. `SampleClear()` sets the channels to a state equivalent to `SampleChannels(32, 0)`.

Spike2 version 9 onwards allows up to 2000 channels, versions 6-8 allow up to 400 channels. Version 5 allows up to 256 but reads up to 400. Version 4 reads files with up to 100 channels. Version 3 can only read files with 32 channels.

```
Func SampleChannels({nChan%{, spec%});
```

`nChan%` If greater than 0, this sets the number of channels in the next data file created for sampling in the range 32-2000. Numbers outside this range are set to the nearer limit. If zero, no change is made. Negative values return information. If you reduce the number of channels, any higher-numbered channels in the sampling configuration are deleted.

`spec%` If present and used with `nChan%` >= 0, this sets the position of the special sampling channels (Keyboard marker, Text Marker and Digital Marker). This value must be less than or equal to `nChan %`. The value 0 or a value greater than the number of channels sets the old special channel scheme (31, 30, 32). Setting values in the range 1-31 are currently treated the same as setting 0, but you should not rely on this always being the case. Values greater than 31 set the position of the Keyboard Marker channel. The Text Marker channel is one channel less than this and the Digital Marker channel is one less than that. If the new special channels positions clash with any existing non-special channels, the non-special channels are deleted.

For example, if `nChan%` were 200 and `spec%` were 100, the Keyboard marker channel number would be 100, the Text Marker channel would be 99 and the Digital Marker channel would be 98.

Returns If `nChans%` is omitted or any value that is not in the table below, the result is the same as for setting `nChans%` to -1. Otherwise the result is:

`nChans%`	Return value

-1 The maximum number of channels in the data file at the time of the call, before any change made by this function.

-2 The maximum channel number that can be sampled using a 1401 interface or a Talker. This is usually the same as the maximum channels in the data file. Before Spike2 9.01, the highest channel number you could sample to was the lower of 100 or the channels in the data file. The 1401 interfaces can sample up to 100 channels.

-3 The channel number of the top special channel, or 0 if the old scheme is in use.

Backwards compatibility

Before Spike2 version 9.01, this command had only 1 argument (`nChan%`) and did not support negative values of `nChan%`. It always returned the maximum number of channels in the file at the time of the call.

See also:

`FileNew(), FileSaveAs(), SampleClear()`

SampleClear()

This either removes a channel from the sampling configuration or resets the sampling configuration to a standard state. From version 8.12 it can also clean out the sampling resources (which hold information on window positions and extra processes to create result and XY views or extra channels).

Note that this leaves the currently set 1401 type alone. We would suggest that to get the most predictable settings, you should follow `SampleClear()` with `SampleOptimise(0, -1, type%)` where `type%` is the type of the 1401 to assume you will be using. Then set all the other sample configuration settings and finally use the `SampleOptimise()` command again with the desired optimisation level.

If you want to achieve the equivalent of the Reset button in the Channels tab of the Sampling configuration, you also need to call `OutputReset()` and also use the reset variant of the `CondSet()` command for all used conditioner channels.

```
Proc SampleClear({chan%});
```

`chan%` Optional argument (default value is 0). If present, this is either a sampling channel number or a code:

>0 The channel number in the sampling configuration to set to Off (remove from the configuration).

0 Equivalent to omitting `chan%`. This resets the sampling configuration to a standard state. It has no effect on the sampling resources.

-1 Resets the sampling configuration to a standard state and clears any sampling resources.

-2 Clears any sampling resources (equivalent to the Sample menu Clear Configuration command). This leaves the sampling configuration unchanged.

The `chan%` argument was added at version 7.00. Setting values of `chan%` < 1 is valid from version 8.12 onwards.

The standard sampling configuration state is:

- a 64-bit `smrx` data file of 32 channels (all Off except the keyboard).
- sampling mode is Continuous.
- microseconds per time is 10 and time per ADC is 10. This is a bit of a long interval for modern systems, but changing it would break backwards compatibility.
- Optimise range of time units is set to 1-50 us unless you have a 1401
- No output sequence files and all methods for triggering a sequence are enabled.
- Any output waveforms for playing during sampling are cleared and output control is set to allow all methods.
- Stop sampling when… fields are disabled and the automatic file name template is cleared.
- All sample rate optimising is set to the equivalent to `SampleOptimise(1,1,-1)` (the -1 means that the assumed type of the 1401 is not changed).
- Sampling automation is cleared, no directory path or Name temples, or repeats, No comment or save file at end of sampling, Triggering is set to use previous, the flush to disk time is set to 5 minutes and the

Sample menu 'Clear Configuration' equivalent
Before version 8.12 there was no exact script equivalent to the Sample menu Clear Configuration command, which forgets about duplicated views and view positioning for the next interactive sampling window. You can now use `SampleClear(-2)` to do this.

You can also achieve the effect of this without changing the current configuration by using the `FileNew(0, mode%)` script command without 2 added to the `mode%` argument. This will cause a single sampling window to be created at the default position.

See also:
`SampleMode(), SampleOptimise(),`

SampleComment$()

This function gets and sets the comment attached to a channel in the Sampling Configuration dialog and the label and comment used by the Sample Bar (also visible in Windows Explorer when inspecting Spike2 configuration files).

```
Func SampleComment$(chan%{, new$|exp%});
```

`chan%` The channel number in the window (1 to 100) or 0 for the Sample Bar comment or -1 for the Sample Bar label.

`new$` If present, the new comment or label. If the text is too long, it is truncated. From version [10.09] you can include place holder strings as described in the channel configuration dialog.

`exp%` 0 or omitted to return the comment without expanding any replacement fields, 1 to expand them.

Returns The original comment or label.

See also:
`SampleCalibrate(), SampleTitle$(), SampleWaveform(), SampleWaveMark()`

SampleConfig$()

This function gets information about the current sampling configuration as text.

```
Func SampleConfig$({get%});
```

`get%` What to get, taken as 0 if omitted. Values are:

- -1 New at [10.07]. The last file `.s2cx` configuration file name associated with the configuration that is not `LAST.s2cx` or `DEFAULT.s2cx`. This is an empty string if the file name is unknown or the configuration file was written by an older version of Spike2. This is the same file as the Copy Last command accessed by a right-click on the Sampling Configuration dialog title bar. The result may be the same as for `get%` set to 0.
- 0 This function gets the name of the file from which the current configuration was loaded or returns an empty string if the name is unknown. Sampling clears the file name as sampling can change the configuration. The file may have a `.cfgx` or `.smr` or `.smrx` file extension. Users can get this path copied to the clipboard by right-clicking the title bar of the Sampling Configuration dialog.
- 1 The configuration rendered as text using spaces to align data.
- 2 The configuration rendered as text using commas to separate fields.
- 3 The configuration rendered as text using Tab characters to separate fields.

Returns The information requested by the `get%` argument.

See also:
`FileSaveAs(), FileOpen(), SampleChanInfo(), SampleSequencer$()`

SampleDebounce()

This function gets and optionally sets the debounce period for Event-, Event+ and Digital Marker channels in the sampling configuration. It is not an error to set this value for other channel types, but has no effect. The debounce period sets the minimum separation of acceptable events; after an event, all following events that are closer than this period are ignored. There is a small time penalty for doing this, so this parameter should be left set to 0 unless it is required.

```
Func SampleDebounce(chan%{, ms});
```

`chan%` The channel number in the sampling configuration.

`ms` If present, the new value of the debounce period, in milliseconds in the range -1 to 1000. -1 means attempt to preserve simultaneous events by giving them consecutive clock ticks.

Returns The previous value of the debounce period, in milliseconds.

See also:
`SampleDigMark(), SampleEvents()`

SampleDerived()

This function adds a derived channel to the sampling configuration. A derived channel takes its input from a sampled data waveform or RealWave channel (currently not from Talker channels) and applies various processes to the channel (see `SampleProcess()`) before writing the result as a RealWave channel.

```
Func SampleDerived(chan%, srcCh%);
```

`chan%` The channel to create as a derived channel.

`srcCh%` The source channel. This must already exist and should be either a Waveform or a RealWave channel. If the source channel is a Waveform, the source data is converted to real values in user units as set for the channel before being processed as a derived channel and written to a RealWave channel in the output file.

Returns 0 if all is well or a negative error code.

See also:
`SampleProcess()`

SampleDigMark()

This adds the digital marker channel to the sampling configuration. The title and comment of the channel can be set with `SampleTitle$()` and `SampleComment$()`. You can read back channel information with `SampleChanInfo()`.

```
Func SampleDigMark(rate)
```

`rate` The expected sustained rate for digital markers on this channel in Hz.

Returns 0 if all went well, or a negative error code.

See also:
`SampleChanInfo(), SampleClear(), SampleComment(), SampleTitle$()`

SampleEvent()

This function sets a channel to sample event data. The title and comment of the channel can be set with `SampleTitle$()` and `SampleComment$()`. You can read back channel information with `SampleChanInfo()`.

```
Func SampleEvent(chan%, port%, type%, rate);
```

`chan%` The channel number in the file to use for this data (1 to the maximum allowed by your hardware).

`port%` The event port number.

`type%` The type of the event channel: 0=Events on a falling edge (Event-), 1=Events on a rising edge (Event +), 2=Events on both edges (Level)

`rate` The expected maximum sustained event rate on the channel in Hz.

Returns 0 if all went well, or a negative error code.

See also:
`SampleChanInfo(), SampleClear(), SampleComment(), SampleChanInfo(), SampleTitle$()`

SampleFPS()

This command is equivalent to the Video FPS when not saving field in the Sampling configuration Mode tab. It sets and reads the slow frame rate to use when sampling is disabled. This sets the slow rate *before* sampling starts, unlike `MMRate()` which changes it during sampling.

```
Func SampleFPS({fps});
```

`fps` If present, the new frames per second value to use in the range 0.0 to 100 (expecting values towards the 0.0 end of the range). Set the value 0 to disable this feature, otherwise the desired frames per seconds when sampling is paused due to sampling in Timed mode or due to the use Pausing writing to disk.

Returns The value of this parameter before any change was made by this command.

See also:
`MMRate()`

SampleHandle()

Returns view handles associated with the current sampling view. This can be used to position, show and hide the output sequencer and sampling control panels. `App()` also returns control panel handles and can be used at any time, even when there is no sampling view open.

```
Func SampleHandle(which%);
```

`which%` Selects which view handle to return:

- -1 The last view that has stopped sampling and not yet been closed. When dealing with automatic sampling of a sequence of files this applies only to the last file in the sequence. New at [10.09].
- 0 The currently sampling time view. If there are duplicates this is the currently active view, or failing that, the first view. Once sampling stops the handle is no longer returned and the result is 0.
- 1 The sampling control panel of the currently sampling time view.
- 2 The sequencer control panel of the currently sampling time view.
- 3 The sample status bar of the currently sampling time view.

Returns The view handle or 0 if the view does not exist or there is no sampling view open. If you ask for the Sampling time view or the last view to stop sampling and it is duplicated, you get the currently active view, and if none are active, the first view.

See also:
`App(), View(), ViewList(), Window(), WindowVisible()`

SampleIdle()

This function is usually not needed. In previous versions of Spike2, it gave time to the sampling process that managed the data transfer between the data acquisition device and the data file. From Spike2 version [8.00] the data capture always runs in a separate thread and we no longer need to give the sampling process any time (but see `SampleStart()`). If you call this, and a time view is sampling data or rerunning, this collects the latest time for which all data is up to date and passes this on to the windows. It also checks the state of sampling/rerun and when sampling to a sequence of data files, gives an opportunity to move on to a new file.

The background idle (which runs when the system has free time) also performs this function. If a script runs continuously without giving time to the background idle, the script will call this periodically. Many of the commands that handle data in a time view also perform this function to ensure that data is up to date.

```
Proc SampleIdle();
```

This runs one cycle of the sample idle. This does not allow windows to update, but before version [10.02] it would allow Spike2 to detect that a 1401 had started to sample, which would start the special sampling thread of execution that managed data transfer from the 1401 to the data file.

SampleInfo()

This command is here mainly for use by CED engineers to allow us to check how the timing services we provide to Talker and Listener devices are doing. The command is:

```
Func SampleInfo(what%)
```

`what%` An integer argument that specifies the information to return. Currently there are two values:

- 0 Return the time correction for the 1 millisecond system timer user by Listener commands (such as the s2video application). The correction is returned in seconds.
- 1 Return the time correction used by talker devices, which is for the Windows Performance Counter. The correction is returned in seconds.

Return The value selected by the `what%` argument or 0.0 if the argument value is unknown or the system is not sampling.

Listener timing correction

The 1401 sends a stream of data during sampling, and part of the stream is timing information. This information is sent periodically, with an adjustable period that is usually between 2 and 18 milliseconds. Each time a block of timing information is received, Spike2 notes the system time that corresponds to it. By tracking both the 1401 time and the system time, it is possible to calculate any drift between the time base used by the 1401 and the timebase used by the computer. Listener devices can also read the computer system clock, so if we know the drift between the 1401 clock and the computer clock, it is possible to correct times measured by Listener devices, such as the s2video application so that they match the times seen by the 1401.

The system clock ticks every millisecond, so each individual measurement is liable to a 1 count error, and as this is compared to a base time with a similar error, there could be up to 2 ticks uncertainty in any one time. However, by averaging over a time period, we can track the time base changes to better than a millisecond.

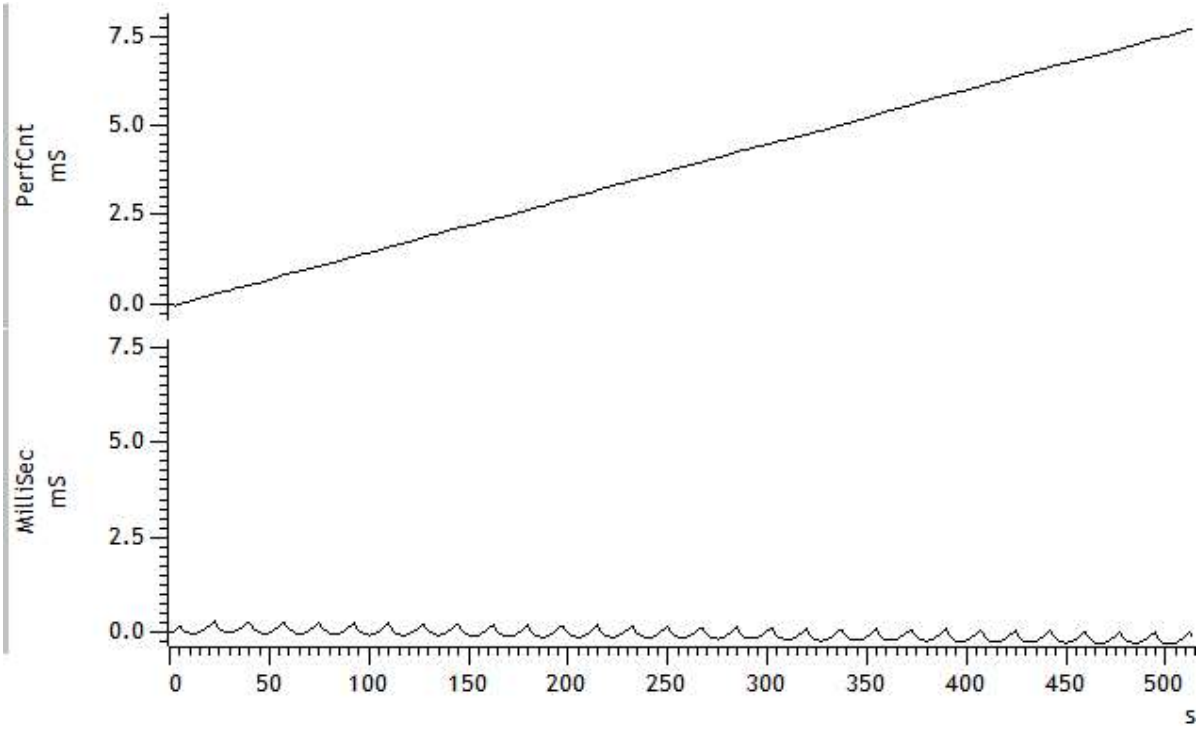
Talker timing correction

The Talker is a more recent addition than the listener and it uses the Windows Performance Counter (which did not exist when the Listener was designed). The performance counter has a much higher timing resolution than the millisecond timer, but we use it in much the same way.

Example time drift

The picture shows an example of the drift you might expect to see. In this particular case measured on my own computer while sampling 16 waveform channels at 65.5 kHz each, the Performance Counter drifted by around 7.5 milliseconds compared to the 1401 time and the system millisecond timer changed very slightly in the opposite direction. The periodic variation of the millisecond timer is probably due to a beating effect between 1401 updates and millisecond timer ticks; as the timer is only good to a millisecond at best, the variation is not significant in use. The drift of the Performance Counter (in this case, but it could just as easily be the

millisecond counter), is about 1.5 parts in 100,000. The crystals used for timing in electronic equipment are usually good to around 1 part in 100,000 (around a second per day), so the fact that the 1401 and the Performance Counter differ by around this figure is not unreasonable. The fact that the millisecond timer happens (in this case) to match the 1401 time to 1 part in 2,000,000 is just luck of the draw.



As it happens, we only use the millisecond counters and the Performance counters as reference points, so it does not really matter what rate they run at as long as it is reasonably consistent with time. Note that the counter speeds will also be affected by temperature. For interest, I repeated this measurement using the same PC and 1401 with a very low data rate (rather than 2 MB/second), and the results were identical.

SampleKey()

Adds an event to the keyboard marker channel of a sampling file as if you had typed it, including triggering the output sequencer and arbitrary waveform output. From version [10.03] you can choose to not trigger the sequencer and/or arbitrary waveform output. For this command to work, `SampleStatus()` must return 2 to indicate that sampling is in progress.

```
Func SampleKey(key$${, flags%});
```

`key$` The first character of the string is added to the keyboard marker channel.

`flags%` New at [10.03]. Optional value, 0 if omitted. This is the sum of two flags:

- 1 If set, do not send the key to the output sequencer where it might cause a jump to an instruction.
- 2 If set, do not send the key to the play wave system, where it might initiate waveform output.

Returns The time stamp of the added marker in seconds or -1 if no file is sampling. You can get -1 immediately after `SampleStart()` because it can take a few milliseconds for sampling to start.

Note that both the sequencer and the play wave system can be set to ignore triggers from the keyboard and the control panel, but script requests are always honoured.

If you have set both the output sequencer and the play wave system to respond to the same key, the play wave system is notified first followed by the output sequencer. The two actions will be close together, but not synchronised. If you require the output sequencer and play wave output to be synchronised, you can trigger the waveform start from the output sequencer, see `WAVEST`.

See also:
Sequencer control, Arbitrary waveform output, `PlayWaveAdd()`, `SampleAbort()`, `SampleReset()`, `SampleStart()`, `SampleStop()`, `SampleStatus()`, `SampleText()`, `SampleWrite()`

SampleKeyMark()

This restores the keyboard marker channel to the sampling configuration. The keyboard marker channel is automatically present in new sampling configurations and this command is only needed in rare cases when a user has imported a sampling configuration from a data file with no keyboard marker channel.

```
Func SampleKeyMark({rate})
```

`rate` The expected sustained rate for keyboard markers on this channel in Hz. If omitted, 1 Hz is set.

Returns 0 if all went well, or a negative error code.

See also:
`SampleComment$()`, `SampleDigMark()`, `SampleEvent()`, `SampleTextMark()`, `SampleTitle$()`

SampleLimitSize()

This corresponds to *Starting and stopping: Stop at file size* in the Automation dialog.

```
Func SampleLimitSize({size});
```

`size` The size limit for the output file, in kB. A positive value sets the size and enables the limit. A negative value sets the limit to the positive value of size, but disables the limit. A zero value, or omitting the argument, means no change.

Returns The limit before the call. If the limit is disabled, the size is returned negated.

See also:
`SampleLimitTime()`, `SampleMode()`, `SampleRepeats()`

SampleLimitTime()

This corresponds to *Starting and stopping: Stop at time* in the Automation dialog.

```
Func SampleLimitTime({time});
```

`time` The time in seconds to set as a limit. A positive time sets the limit and enables it. A negative time sets the limit to the positive time, but disables the limit. A value of zero, or omitting the argument, leaves the time limit unchanged.

Returns The limit before the call. If the limit is disabled, the time is returned negated.

See also:
`SampleLimitSize()`, `SampleMode()`, `SampleRepeats()`

SampleMode()

This function sets and gets the sampling mode in the Sampling configuration dialog. It is equivalent to the Sampling Mode tab of the Sampling Configuration when setting Continuous or Timed sampling modes. To set Triggered mode you should use `SampleTrigger()`; mode 3 is maintained here solely for backwards compatibility with ancient versions of Spike2.

```
Func SampleMode(mode%{, time, trig%|every{, scr${, atEnd%}}});
Func SampleMode(get%{, &scr$});
```

`mode%` This argument determines the action of the command:

0 Returns the current mode as 1, 2, or 3 for Continuous, Timed or Triggered. Any additional arguments are ignored.

1 Sets Continuous recording mode.

2 Sets Timed recording mode. At least three arguments are required.

3 Deprecated. Sets Triggered recording mode with all channels triggered. The first three arguments are required; the `scr$` and `atEnd%` arguments may NOT be used. Please use the `SampleTrigger()` command in place of this `mode%` value.

`time` In modes 2 and 3 it sets the `For` field of the Sampling mode group, in seconds.

`every` In mode 2 it sets the `Every` field for timed sampling, in seconds.

`trig%` The trigger channel number for mode 3. See the `SampleTrigger()` command for a description of the range of channels that can be set.

`scr$` From Spike2 version [10.09] you can supply the path to a script to associate with timed sampling. For a script to run, no other script must be running and the system must have idle time. Such sampling scripts typically have no user interface and run and exit very quickly. We do not limit what such scripts can do; once started these scripts behave as if you had run them interactively. In the `get%` variant, this optional argument returns the script name or an empty string if no script is set.

`atEnd%` 0 to run the script as soon as possible after each timed sample start. Set to 1 to run the script as soon as possible after each timed sample end.

`get%` Used to return information from the sampling configuration:

-1 Returns the `For` field value, valid in Timed mode.

-2 Returns the value of the `Every` field, valid for Timed sampling mode.

-3 Returns the `Trigger` field value, valid in Triggered sampling mode.

-4 Returns the `atEnd` field value and `scr$`, if present, returns any script set. New at version [10.09].

Returns For `mode%` values 0-3 it returns the current sampling mode as 1-3, or a negative error code. For values -1 to -3 it returns the information described for `get%`, above.

If you set triggered mode, triggers 2 to 4 are disabled and the command is equivalent to calling `SampleTrigger(1, trig%, -1, 0.0, time, -1)`.

See also:
`SampleLimitSize(), SampleLimitTime(), SampleScript(), SampleTrigger(), SampleUsPerTime()`

SampleOptimise()

This function sets and gets the sample rate optimising settings in the Resolution tab of the Sampling configuration dialog. Spike2 can match the requested sample rates for waveform and WaveMark channels by changing the microseconds per time unit, time units per ADC convert and by making waveform channels into Quick or Slow or Copy channels.

The available ADC sample rate is divided by giving one share to each WaveMark channel, one to each Quick channel and one to the entire Slow channel group. The actual rates for each waveform channel (Quick or Slow) can then be further down-sampled by taking one point in n. For example, with 4 Slow channels, the fastest rate for a Slow channel is one quarter the fastest rate for a Quick channel.

The rates are optimised each time you change any aspect of the sampling configuration that affects the sample rate. With full optimise set and slow sampling rates it can take an appreciable time for Spike2 to search all possible combinations for the best sample rates. As `SampleClear()` sets the equivalent of `opt%` to 1, we recommend that you follow `SampleClear()` with `SampleOptimise(0, -1, type%)` where `type%` is the code for the 1401 type you wish to optimise for, then create all the channels, and finally you use this command again with the desired optimisation mode set (usually 1 or 2) so that you pay the time penalty for optimising once.

Choosing the best rates is a complex process. When you do this interactively, you get to see the channel rates and it is easy to detect that the rates you want have not been achieved. If you use the script to do it, *we strongly recommend that you check the rates you have obtained*. Old code (before Spike2 8.08) could do this after

creating a new file and before starting to sample by checking the `Binsize()` of the waveform channels. From version 8.08 you can use `SampleOptimise(7/8, err)` to find the range of under-sampling or over-sampling. Under-sampling errors of 10% or more and over-sampling errors of 30% or more are logged to the Sampling Notes when you start to sample.

```
Func SampleOptimise(opt%,group%,type%{,usLo%{,usHi%,{dis%{, burst%}}}});
Func SampleOptimise(get%{, err});
```

`opt%` This sets the optimise method. Set -1 for no change or:

0 No optimise of microseconds per time or time per ADC.
1 Partial, optimise time per ADC, no optimise of microseconds per time.
2 Full, optimise both time per ADC and microseconds per time.

`group%` This controls the use of Quick channels. Set -1 for no change or:

0 Version 3 compatible; no Quick channels and channel divides up to 65535 only.
1 Group channels with the same ideal rate so they get the same actual rate. Use this value unless you really need one of the others.
2 Optimise for the least error in rate; channels with the same ideal rate may get different actual rates.

`type%` Set the type of 1401 to optimise for or -1 for no change. Settings marked * relate to 1401 types that are no longer supported for sampling. Remember that this need not be the 1401 that you will use (as long as the unit you will use is capable of at least your requested settings). The actual 1401 type is checked before you sample to make sure that only reasonable requests are made of it.

0 Works with all supported 1401s except a 1401*plus* with an old ADC. You usually do not want this.

1*	1401*plus* with an old ADC.	5	Power1401 625	9	Power1401-3
2	Power1401	6*	micro1401 mk I	10	Micro1401-4
3*	1401*plus* or a micro1401	7	Power1401 mk II		
4	Micro1401 mk II	8	Micro1401-3		

`usLo%` If present, it sets the low limit, in microseconds, for optimising microseconds per time unit (when `opt%` is 2). Values outside the range 1 to 1000 are ignored. If you are sampling to a 64-bit `smrx` file you should set this to 1.

`usHi%` If present, it sets the high limit, in microseconds, for optimising microseconds per time unit. Values outside the range 1 to 1000 are ignored.

`dis%` If present and not set to -1, this disables features for backward compatibility with versions of Spike2 before 6.05 and with older 1401s. `dis%` is the sum of:

1 No 10 MHz clock. Forces microseconds per time to be an integral number.
2 No WaveMark copy. If a waveform uses the same 1401 port as a WaveMark the Power1401 mk II would normally take only one data sample for WaveMark and then copy it for the waveform. Set this flag to prevent this.
4 No WaveMark divide. Set this bit to stop WaveMark channels using a divider. This stops a waveform channel sampling any faster than a WaveMark channel.
8 No dummy channels. When supported by the 1401, Spike2 normally adds in up to 4 dummy channels into the sampling if this improves the match between the requested sample rate and the actual rate.
16 No 64-bit optimisation of Microseconds per time unit in full optimise mode to give the same time units as Spike2 version 7.

`burst%` If present and not set to -1, this sets the state of the Burst mode check box. Set 1 for burst mode, 0 for non-burst mode.

`get%` Use the function with one argument to read back the current settings with codes 0-6. From Spike2 version 8.08 you can also use codes 7 and 8:

0 The current value of `opt%` in the range 0-2.
1 The current value of `group%` in the range 0-2.
2 The type of 1401 that is currently set for optimisation as described for `type%`.
3 The `usLo%` value that is used when `opt%` is 2.
4 The `usHi%` value that is used when `opt%` is 2.
5 The `dis%` flags for optimisation features that are disabled for backwards compatibility.

6 The current `burst%` setting, 0 for not burst, 1 for burst mode.
7 Return the channel number that is sampling the slowest compared to the ideal rate for the waveform or WaveMark channel or 0 if all channels sample rates are at least the ideal rate. The error is returned in `err` (if present) as a negative value in the range -1 up to 0.
8 Returns the channel number that is sampling the fastest compared to the ideal rate, or 0 if no channel is sampling faster than the ideal rate. The error is returned in `err` (if present) as a positive number.

`err` Use the function with two arguments and codes 7 and 8 to read back the proportional error of the channel sampling rate from the ideal rate (or 0.0 if the channel number is returned as 0). The returned `err` value is: `(rate - ideal)/ideal` which is negative if the rate is less than the ideal (`get%` = 7) and positive if it is greater (`get%` = 8).

Returns The setting version returns the current optimise method. When called with one argument, the return values are as documented for `get%`.

See also:
`SampleLimitSize(), SampleLimitTime(), SampleTimePerAdc(), SampleUsPerTime()`

SampleProcess()

This command adds a sample process to either a derived data channel or to a sampled Waveform or RealWave channel (not a Talker channel). You can add up to 5 (an arbitrary limit) processes to a channel. Each process increases the time taken to process a channel. This is equivalent to the Add button in the Derived channel dialog. You can also use command variants to report on the channel processes or to delete one or all processes from a channel.

There are various command variants, but the first two arguments are common to all:

```
Func SampleProcess(chan%, proc%{, ...});
```

`chan%` A channel in the sampling configuration to add the process to. This must either be a Derived channel created with `SampleDerived()` or a sampled Waveform or RealWave channel, in which case the output will be processed and written as a RealWave channel.

`proc%` An integer value that sets the type of process to add:

#	*Function*	*Additional arguments*
0	IIR digital filter	`type%, model%, order%, fr1{, fr2{, extra}`
1	Rectify the input signal	`mode%`
2	Difference the input signal	None
3	Down-sample the input signal	`ratio%`
4	Activity detect as Marker	`mode%, trig%, off%, tcDC, tcPeak, trgLev, offFr`
-1	Get process count	None
-2	Read back added process	`index%, args[]`
-3	Delete one or all processes	`index%`

Return Requested information or 0 for success or a negative error code, or stops the script with an error if arguments do not match any possible call or an impossible filter is requested.

Add IIR filter

You can add any of the types of IIR filter that are supported by the `IIRCreate()` script command, using the same arguments:

```
Func SampleProcess(chan%, 0, type%, model%, order%, fr1{, fr2{, extra}});
```

`type%` Sets the filter type as: 0=Low pass, 1=High pass, 2=Band pass, 3=Band stop.

`model%` Sets the filter model: 0=Butterworth, 1=Bessel, 2=Chebyshev type 1, 3=Chebyshev type 2, 4=Resonator.

`order%` Sets the filter order in the range 1-10. Resonators always set an order of 2.

`fr1` Sets the corner frequency in Hz for low pass, high pass filters, the centre frequency for resonators, and the low corner frequency for band pass and band stop filters.

`fr2` Sets the upper corner frequency in Hz for band pass/stop filters, otherwise ignored.

`extra` Sets the ripple for Chebyshev filters in dB in the range 0.01 to 1000 and the Q factor for resonators in the range 1 to 10000. If you are unsure of what is a suitable value, use the interactive IIR digital filter dialog to design a suitable filter and copy (or record) your settings.

We implement real-time filtering with IIR filters because they allow reasonably efficient incremental filtering in real time. You can generate filters with very sharp edges by increasing the filter order. However, you should be aware that the time taken to process the data increases with the filter order, and that band-pass and band-stop filters require twice the number of mathematical operations as low-pass and high-pass filters. Also, the higher the filter order, the more a position of a feature in a signal is delayed (this is true of analogue filters, too).

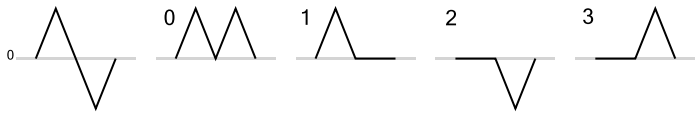
See the discussion of filters in the Digital Filtering section for more information.

Rectify the input

You can full-wave or half-wave rectify the signal. The command is:

```
Func SampleProcess(chan%, 1{, mode%});
```

`mode%` 0 (or omitted) for full wave rectification. 1 for half-wave rectification (preserves positive signals, negative signals set to 0), 2 for half-wave preserving negative signals and setting positive signals to 0, 3 inverts the signal then applied half-wave rectification. The image shows the effect of the modes on an input waveform.



You might use this to extract an EMG activity monitor channel by generating a derived channel from a waveform sampled at a few kHz then adding:

1. A high-pass IIR filter to remove any signal offset (if necessary)
2. Full-wave rectification
3. A low-pass IIR filter to smooth the signal
4. A Down-sample process to reduce the disk space used by the channel

Difference the input signal

Each input data point is replaced by the change between the input and the previous input. The command is:

```
Func SampleProcess(chan%, 2);
```

If you follow this with a low-pass filter, you get a measure of the rate of change of the signal. We adjust the data scaling to convert the result into the original units per second.

Down-sample the input signal

Take every n^{th} value from the input stream, starting with the first. The command is:

```
Func SampleProcess(chan%, 3, ratio%);
```

`ratio%` An integer down-sample ratio in the range 2 up to 10000. We do not let you divide by 1 as this is pointless. The limit of 10000 is arbitrary.

If you down-sample a signal, it is important the it does not contain any significant frequency components above half the output sample rate. If such components exists, they will be aliased back into the frequency range of 0 to half the sample rate. For this reason, you will usually have added a low-pass or band-pass filter to the channel before you add this process unless you know that the signal has no problematic frequency components.

Activity detect (convert wave to marker channel)

Track the input signal and detect activity or Peaks and/or Troughs. See the description of the Activity detect dialog for more information. This was added at [10.13].

```
Func SampleProcess(chan%, 4, mode%, trig%, off%, tcDC, tcPeak, trgLev,
offFr);
```

`mode%` 0=Activity, 1=Peaks, 2=Troughs, 3=Peaks and Troughs

`trig%` The marker code (0 to 255) to generate for activity start or Peak detect in Peak mode or Peak and Trough mode, or Trough detected in Trough mode.

`off%` The marker code (0 to 255) to generate for activity end or Trough detected in Peak and Trough mode, or -1 for no code. If in Peak and Trough mode and no code, the value `trig%` +1 is used.

`tcDC` The high-pass filter time constant to apply to the input waveform before the detection process in the range 0 to 1000 seconds. Setting the value to 0 is a special case that turns the filter off.

`tcPeak` The Peak time constant used in Activity mode, the maximum width of the Peak/Trough in other modes. This is in the range 0.001 to 1000 seconds.

`trgLev` The trigger level to detect activity or to cross for Peak and Trough modes in the range 0 to 1000000. In Trough mode, the value is negated before use. In Peak and Trough mode it sets the level for Peaks and -`TrgLev` sets the level for Troughs.

`offFr` In Activity mode, this is the fraction of the Trigger level the peak must decay to reach the Off condition. In Peaks and Troughs mode it is the fraction of the peak the signal must fall to within the `tcPeak` period to count as a detected peak. This must be in the range 0.0 to 0.99. In modes 0 and 3 a value of 0 is unlikely to be useful.

Get process count

Return the number of processes that were added to the channel.

```
Func SampleProcess(chan%, -1);
```

Return The number of attached processes or a negative error code.

Get added process arguments

Return the arguments used to create one of the added processes.

```
Func SampleProcess(chan%, -2, index%, args[]);
```

`index%` The index of the added process to enquire about. The first added process has index 0, the second has index 1, and so on.

`args[]` A vector of real values that is filled in with the arguments required to regenerate the process. If the vector is too small to hold all the values, the excess arguments are omitted from the right. The first argument at index 0 is returned holding the `proc%` argument of the original command. The following elements are filled in sequence from the remaining arguments.

Returns The number of elements that are available to be returned for the process (2 for down-sample, 5-7 for an IIR filter, 1 for the remainder). This value may exceed the size of `args[]`. This value is 0 if the channel does not exist or does not contain any processes or if `index%` does not match a process.

Delete one or all processes

Delete a nominated or all the processes attached to a channel.

```
Func SampleProcess(chan%, -3, index%);
```

`index%` The index of the added process to enquire about. The first added process has index 0, the second has index 1, and so on. Set this to -1 to delete all processes.

Returns The number of processes that were deleted.

See also:

`SampleDerived()`

SampleRepeats()

This gets and optionally sets the Sampling configuration Automation tab Repeats field, which is used to sample a sequence of files with automatically incrementing names. You can use it online to return the number of files left to sample. You can choose to trigger each file in the sequence, or just the first with `SampleStartTrigger()`.

```
Func SampleRepeats({files%});
```

`files%` If you omit this argument, the number of files is not changed. Positive numbers set the number of files to sample; 0 and 1 both sample a single file. Set negative values to report information during sampling:

- -1 Return the number of files that remain to be sampled (including the current file).
- -2 Return the number of repeats completed (not including the current file as this is not complete). This is 0 during the first file, 1 during the second, and so on. New at [10.09].
- -3 Return the current file number or -1 if none. File numbers may not be consecutive if some files that match the file name template already exist. New at [10.09].

Returns Positive or omitted `files%` returns the number of repeats set at the time of the call. Negative `files%` returns the requested information or -1 if there is no sampling document. The sampling document stops existing when the last file of a sequence of repeated files closes.

You must also set a file name with `SampleAutoName$()`, set automatic file saving with `SampleAutoFile(1)` and set a time or size limit with `SampleLimitTime()` or `SampleLimitSize()`, or load a suitable sampling configuration with these values set.

As each file reaches the time or size limit, it is saved, closed and replaced with the next file. Spike2 saves the file resources on close, including any duplicated files and processes, and will attempt to restore them with the next file. The view handle always references a file, but its file name and sampling status will vary. The last sampled file in the sequence is not closed. You can use `SampleStop()` or `SampleAbort()` to end a sequence early and `SampleReset()` to restart a file.

To make life easier for the script programmer, each time a file closes and sampling starts again, we save the view handle of the first data view of the file and reuse it for the first data view of the next file. As long as you do not duplicate this view and delete it, the first view of the entire file sequence has the same view handle. This allows you to save the handle from the `FileNew(0, ...)` command that creates the initial file and use it throughout the sampling session. Any other created views (duplicates, result and XY views) get the next available handle.

See also:
`SampleAutoFile(), SampleAutoName$(), SampleLimitSize(), SampleLimitTime(), SampleReset(), SampleStart(), SampleStartTrigger(), SampleStatus(), SampleStop()`

SampleReset()

This abandons sampling, deletes any data that has been written to disk, and returns to a state as if `FileNew()` had just been used to create a new data file. If it is used when sampling a sequence of files, the current file is restarted and the sequence continues.

```
Func SampleReset();
```

Returns 0 if sampling is in progress or is waiting for a trigger and the reset operation completed without a problem. Otherwise a negative error code.

See also:
`SampleAbort(), SampleStart(), SampleStop(), SampleStatus(), SampleWrite()`

SampleScript()

This script command is equivalent to the Script tab of the Sampling configuration dialog. The command sets and gets the values in the sampling configuration. It defines a script that can be run at defined times during sampling. There are no restrictions on what the script can do. It is exactly equivalent to running the same script

from the Script menu. However, most sampling scripts will take very little time to run and will have no user interface. This command was added at version [10.09]

```
Func SampleScript(scr$, flags%{, first{, every}});
Func SampleScript(get%{, &scr$});
```

`scr$` Path to the script to run. The script need not exist.

`flags%` Set of flags to define the times at which the script should run. The flags are the sum of:

- 1 Run when file created
- 2 Run when the `first` time is reached
- 4 Run at `first` + n * `every` reached
- 8 Run when sampling has stopped

`first` The time when the script should run if the run at if `flags%` contains 2. Setting values less than 0 is an error.

`every` The interval between timed runs if `flags%` contains 4. Setting values 0 or less is an error.

`get%` Determines the information to return. Values 0 or more are reserved. Other values are:

- -1 Returns the `flags%` value and `scr$`, if present, is set to the script name.
- -2 Returns the `first` value.
- -3 Returns the `every` value.

Returns 0 when setting values. Otherwise the value requested by `get%`.

See also:
`SampleMode(), SampleTrigger()`

SampleSeqClock()

This command changes the output sequence tick rate. You can change the default rate used when compiling a sequence that does not contain a `SET` or `SCLK` directive. You can also dynamically change the speed of a running sequence (the sequencer tick period is not guaranteed to be accurate during the change). When you change the default tick, time periods set in a sequence by `s()`, `ms()` and `us()` are set as accurately as the tick rate allows.

If you change the rate of a running sequence, all sequencer-tick related periods, including periods set by `s()`, `ms()` and `us()`, are scaled by the ratio of the new period to the period that applied when the sequence was compiled. For example, if a sequence had a `SCLK 1` directive at the start, and you changed the clock when running with `SampleSeqClock(1, 0.5)`, the sequence would run twice as fast, so periods generated with `s()`, `ms()` and `us()` would become half as long.

We anticipate that this instruction will only be used in very specialised circumstances.

```
Func SampleSeqClock({which%{, ms}});
```

`which%` Set to 0 (or omit) to get or set the default tick period used when compiling a sequence. Set to 1 to get or set the tick period in use by a running sequence.

`ms` If present and greater than 0, this sets the new tick period in milliseconds. If this period is less than the minimum period for the active 1401 interface, it is limited to the minimum period. The period will be set to microsecond accuracy; if you choose a period that is an integer multiple of microseconds, it is set exactly.

Returns The tick period in milliseconds at the time of the call. If you set `which%` to 1 when there is no running sequence, the returned value is 0.

See also:
`SampleKey(), SampleSeqStep(), SampleSequencer$(), SampleStart()`

SampleSeqCtrl()

This gets and optionally sets options that control the use of the output sequence. You can read more about these options in the documentation for the Sequencer tab of the Sampling Configuration dialog.

```
Func SampleSeqCtrl(opt%{, new%});
```

`opt%` There are three options:

1 The sequencer jump control. The values you can set with `new%` or read back are: 0 = keyboard, control panel and script, 1 = control panel and script, 2 = script only. It is an error to set values outside this range. The default value after `SampleClear()` is 0.

2 The minimum number of sequencer instructions to reserve space for. Values outside the range 0-8192 are set to the nearer limit. This is used when you load new sequences during sampling as space for the largest sequencer must be reserved before you start. Space is always reserved for the sequence that is loaded at the start of sampling, so setting a value of 0 is fine if you will not be loading any additional sequences during sampling, or if they are all the same size of smaller than the original. The default value after `SampleClear()` is 0.

3 The minimum number of table entries to reserve space for. Values outside the range 0-1000000 are set to the nearer limit. This is used when you are going to load new sequences during sampling. Space is always reserved for the table usage of the sequence that is loaded when sampling starts, so setting a value of 0 is fine if you will not load additional sequences, or if they all use the same or less space than the original sequence. The default value after `SampleClear()` is 0.

`new%` The new value for the control option.

Returns The value of the option selected by `opt%` at the time of the call (before it is changed by `new%`).

See also:
`SampleKey(), SampleSeqStep(), SampleSequencer$(), SampleStart()`

SampleSeqStep()

This command returns information about sequencer steps and the current sequencer step when running. For this command to be useful, there must be a sampling document open for sampling. There are three command variants:

Get current sequencer step when running

This returns the current sequencer step or -1 if not sampling. If running, but no sequencer has ever been set for this sampling session, the result is 0. Otherwise it returns the last known sequencer step. If you know that your sequence can stop at one of several locations in the sequence, you could use this command to detect the state of the sequence. Information about the current state of the 1401 is transferred to the host PC every few milliseconds.

```
Func SampleSeqStep();
```

Returns The current (last known) sequence step number, or -1 if not sampling if no sequence. The sequencer step is updated every few milliseconds.

Get list of active keys in the current sequence and their step numbers

This command variant was added at Spike2 version [10.08].

```
Func SampleSeqStep(&keys${, steps%[]});
```

`keys$` This string is returned holding the keys defined in the sequence in step number order.

`steps%` An array of integers. The first `Len(keys$)` elements are filled in with the 0-based step number that the key is associated with.

Returns -1 if no sequence, else the number of steps in the sequence. The number of keys is `Len(keys$)`.

Get information for a sequencer step

This command variant was added at Spike2 version [10.08].

```
Func SampleSeqStep(step%, &key$, &com$, &disp$);
```

`step%` The 0-based step number to get the sequence information for. It is an error for this to be out of the range 0 to 8192.

`key$` Returned holding key associated with the step or an empty string if no key.

`com$` Returned holding any comment associated with the step or an empty string.

`disp$` Returned holding any display string associated with the step or an empty string.

Returns -1 if no sequence, 0 if `step%` does not exist, else the number of steps in the sequence.

See also:
`SampleKey(), SampleSequencer$(), SampleSeqVar()`

SampleSeqTable()

If there is a sampling document with an output sequence, you can use this function to find the size of any table set in the sequence by the `TABSZ` directive. You can also use this to transfer data between an integer array and the table

```
Func SampleSeqTable({tab%[]{, offs%{, get%}}});
```

`tab%[]` An integer array holding items to transfer to the 1401 sequencer table or to hold items read back from the table. The array size sets the maximum item count

`offs%` This sets the index into the sequencer table to start the transfer. The first index in the table is 0. If this value is negative or greater than or equal to the sequencer table size, no data is transferred. If omitted, the value 0 is used.

`get%` Set 0 or omit this argument to transfer data to the sequencer table, set to 1 to transfer data from the sequencer table.

Returns If you call this with no arguments, the return value is the size of the sequencer table. Otherwise, the returned value is the number of items transferred between the sequencer table and the array. A negative error code is also possible, for example if there is no sampling document.

A sampling document must exist before you can use this function. See `FileNew(0,...)` to create a sampling document from the script language.

See also:
`SampleKey(), SampleSequencer$(), SampleSeqVar()`

SampleSeqTick0

If there is a sampling document open and the output sequencer is running, this command lets you read and set the zero value used by the `TICKS` sequencer instruction and set by the `TICK0` instruction. This command was added at Spike2 version 8.00.

```
Func SampleSeqTick0({tZero});
```

`tZero` If present, the new zero time to use for the `TICKS` instruction. It is not illegal to set a negative time, which may be useful if you want to take advantage of the overflow feature of the `TICKS` instruction.

Returns The zero tick time in seconds before any change made by this command was made or -1 if there is no sampling document, or no sequence or we are not sampling.

Although you could collect this values using `SampleSeqVar()` (the values are stored as Spike2 clock ticks in variables `V255` and `V256`), this command greatly simplifies the process and takes care of converting the result into a time in seconds.

See also:
TICKS, TICK0, SampleSeqVar(), Sequencer variables

SampleSequencer()

This function sets the sequencer file to attach to the Sampling configuration and sets and gets the sequencer mode (None=0, Text=1 or Graphical=2) and can save a graphical sequence as a text sequence. Use SampleSequencer$() to get the name of the current sequencer file in the sampling configuration. You can also use this command to change the current sequence while sampling and, since [10.15], to store a text sequence as part of the sampling configuration (avoiding the need to have a separate file). There are two command variants:

```
Func SampleSequencer(src$);
Func SampleSequencer({mode%{, src$}});
```

src$ A sequencer file name with extension .pls or the sequencer text in modes 5 and 6. In the first variant this sets the sequencer file name in the sampling configuration and sets Text mode; pass an empty string to set no sequencer file. In the second variant, src$ is ignored unless mode% is 1 or 3-7; omitting src$ is equivalent to passing an empty string.

mode% If omitted, the value -1 is used. This determines what action the command takes. Modes 5, 6 and 7 were added at version [10.15].

-1	No effect, return the current mode.
0	Set sequencer setting to None in the sampling configuration
1	Set a text sequence in the sampling configuration using the name set by name$. If name$ is empty, this is the same as mode 0. The sequence is not checked for validity.
2	Set graphical sequence mode in the sampling configuration (this is not really supported by the script as you cannot program the graphical sequencer from a script except by loading a sampling configuration with a graphical sequence set).
3	Write the current graphical sequence in the sampling configuration as a text sequence to the file name in src$. The sequence is constructed for the 1401 type set in the resolution tab or by the SampleOptimise() command
4	Used during sampling to replace the current output sequence with the sequence held in the file named by src$.
5	Set the text of the sequence saved in the sampling configuration to src$.
6	Used during sampling to replace the current output sequence with the text sequence held in src$. If src$ is empty, uses the text sequence held in the sampling configuration.
7	Set the text of the sequence saved in the sampling configuration to the contents of the file src$.

Returns The first variant returns the sequencer mode at the time of the call or a negative error code. The second variant returns values depending on mode%:

-1	The sequencer mode at the time of the call. 0=None, 1=Named file, 2=Graphical sequence, 3=text sequence saved in sampling configuration.
0	The sequencer mode at the time of the call.
1	The sequencer mode at the time of the call or a negative error code if the file does not exist.
2,5	The sequencer mode at the time of the call.
3	1=Not in graphical sequence mode, 0=success, -1=could not create file, -2=error writing the file, -3=too many varying pulses (the limit is 10).
4,6	0=success, 1=not sampling with a sequence, 2=not sampling, 3=internal error, 4=too many instructions, 5=table too big. A negative return value means that the file could not be found or did not compile to a legal sequence.
7	The sequencer mode at the time of the call or a negative error code if the file does not exist or could not be read.

Script generated sequence

If you want to generate a sequence within your script, this is possible by writing the script to a file, or from [10.15] by writing the sequence into the sampling configuration. You need to find a suitable location that you are allowed to write to. The following code demonstrates the idea:

```
const pulse$ :=
"            SET    0.01,1,0        ; Get rate & scaling OK\n"
"E0:         DAC    0,5             ; Start point for ramp\n"
"            RAMP   0,-5,0.001      ; Generate a ramp\n"
"            MARK   0               ; Generate digital marker\n"
"            DELAY  s(0.1)-3\n"
"            MARK   1,E0            ; Generate digital marker, back to start\n";

'Generate a file to use for pulse output
'Returns    The name of the file or an empty string
Func MakePulse$()
var path$ := FilePath$(-2)+"ramp.pls"; ' User's document folder
var tv% := FileOpen(path$, 8, 1);   'Create external text file for writing
if (tv% > 0) then                   'If opened OK...
    Print(pulse$);                  '...write the sequence and...
    FileClose();                    '...close the file
    return path$;
endif;
return "";                          'Failed, so no file name
end;
```

In this case the sequence code is fixed, but a script could also generate it. Then use it with code like:

```
var ok% := SampleSequencer(MakePulse$());
if ok% < 0 then ...handle error... endif;
```

From version [10.15]. on you could use:

```
SampleSequencer(5, pulse$);
```

avoiding the need to write the file.

See also:
`SampleKey(), SampleOptimise(), SampleSequencer$(), SampleSeqVar()`

SampleSequencer$()

This function returns the name of the sequencer file that is currently attached to the sampling configuration. Use `SampleSequencer()` to set the file.

```
Func SampleSequencer$({what%});
```

`what%` This argument was added at version [10.15] and determines what the function returns. If 0 or omitted, it returns the currently set file name. If `what%` is 1, it returns the sequencer text stored in the sampling configuration.

Returns It returns the current sequencer file name or text, or an empty string if there is no file or stored sequence. When `what%` is 0 or omitted the returned name includes the full path.

See also:
`SampleKey(), SampleSequencer(), SampleSeqVar()`

SampleSeqVar()

This is used during sampling with an output sequence, to get or set the value of an output sequencer variable. Values set before the sampling window exists are ignored. Values set before `SampleStart()` set the initial variable values and can be read back.

```
Func SampleSeqVar(sVar%{, new%});
```

`sVar%` The sequencer variable to set or read, in the range 1 to 256. Note that some of these variables have pre-defined uses.

`new%` The new value for the output sequencer variable. If present, the value of the variable is updated. Omit to return the variable value. A common error when setting variables for the `DAC` instruction is to set a value 65536 times too small. Beware that to be compatible with version 7, this value must be a 32-bit signed integer value, that is it must lie in the range -2147483648 to 2147483647. This can be an issue when calculating probability values for `BRAND`. From [10.13] we allow the range -2147483648 to 4294967295 to make it easier to use `BRAND`; previously it was an error to exceed 32-bit signed integer range.

Returns If you are setting a value, or this is used when there is no sampling document, the function returns 0. If you are reading a value, the function returns the value as a signed 32-bit number. If you want an unsigned value, use `result% & 0xffffffff`.

See also:
Output sequencer DAC instruction, `SampleSeqStep(), SampleSequencer$(), SampleStart()`

SampleStart()

Use this function after `FileNew()` has created a new time view based on the current Sampling configuration to request that Spike2 starts sampling, or arms the sample trigger to start sampling.

```
Func SampleStart({trig%});
```

`trig%` 0 = start sampling now (default), 1 = wait for a Trigger input signal, -1 = use the sample configuration trigger setting.

Returns 0 if all went well or a negative error code.

Separate sample thread of execution

To ensure that sampled data is not lost when Spike2 is busy updating the screen or running a script, separate threads of execution are created to handle the 1401 and Talkers; these threads move data from the external data device (1401 and/or Talkers) to the Spike2 data file. The main Spike2 user thread monitors the state of the data file and updates the screen as required in 'idle' time.

Idle time happens when Spike2 is waiting for you to tell it what to do. When a script is running, unless you deliberately grant idle time with script commands like `Interact()`, `Toolbar()`, `DlgShow()`, `Input()`, `Input$()`, `Query()` or `Yield()`, Spike2 is busy and sampled data files will not update the screen, and eventually Spike2 will be marked as unresponsive. Windows requires idle time to let it do housekeeping and to keep the system feeling responsive.

Sampling with a 1401 and idle time from version [10.02] onwards

The separate thread that handles 1401 input is created when the 1401 is told to start sampling (in triggered mode or immediately), and it detects when the 1401 has actually started to sample (which could be after a long delay in triggered start mode). If you do not provide idle time, Spike2 will not feel responsive and the display will not update, but the start of data transfer will be detected and data will be transferred.

Start up problems

The following code will probably not work as you intend:

```
SampleStart();           'Start sampling now
var t := SampleKey("A"); 'Trigger the output sequencer/Play wave system
```

If you run this, the variable **t** will most likely be set to -1 and the key will not be logged.

The problem is that the sampling system runs asynchronously to the script system. The call to `SampleStart()` sets everything up and starts the separate sample thread of execution. At this point, the sample state is "go requested" and `SampleStatus()` will return 0. At some point the sampling device (for example a 1401 interface) will start to sample and send a message to the sampling thread to indicate that sampling has begun. At this point `SampleStatus()` will return 2, to indicate that we are sampling. For example, on my work computer with a Micro4 interface, it takes around 5 ms after `SampleStart()` returns to `SampleStatus()` returning 2.

Here is a safer way to start immediately with a 1 second time out in case of a problem:

```
SampleStart();
Seconds(0, 1);   'Start a high resolution timer
while SampleStatus() <> 2 do
    if Seconds() > 1 then Message("Failed to start"); halt endif;
    Yield();
wend;
var t := SampleKey("A"); 'Trigger the output sequencer/Play wave system
```

The lazy programmer could simplify this to:

```
SampleStart(); ' Request sampling starts
Yield(0.1);    ' Allow time for startup
var t := SampleKey("A");
```

If you are running the output sequencer, which starts synchronously (to the microsecond) with 1401 sampling, you could use it to start whatever process is required, avoiding this problem.

Before version [10.02]

We provide this information in case you must write a script that also works with old versions of Spike2.

Before version 10.02, the separate thread of execution to handle data capture was not created until the 1401 had started to sample. Detection of the start of sampling and creation of the sample thread happened in idle time. If you did not provide idle time after calling `SampleStart()`, it was possible for sampling to fail. In most cases sampling did not fail because quite a few sampling-related commands (for example `SampleStatus()` and `MaxTime()`) allow one cycle of sample idle per call, which would often be enough to get the separate sample thread of execution started.

If you are writing a script that has to run on older versions of Spike2, and you really must do a lot of processing in a loop after calling `SampleStart()` you can use `Yield(0)` or `SampleIdle()` periodically (a few times a second) to give sampling a chance. You need to do this until sampling has started.

Detect that triggered sampling has started

A simple way to detect that triggered sampling has begun is to use `SampleStatus()`, which returns 1 when waiting for the trigger and 2 when triggered. You can also use `MaxTime()` on the sampling view, which will return 0 until sampling has started. Both of these commands allow the background sampling thread to run.

See also:

`SampleAbort(), SampleRepeats(), SampleReset(), SampleStop(), SampleStartTrigger()`

SampleStartTrigger()

This gets and optionally sets the trigger option used when sampling starts. It is equivalent to *Starting and stopping: Triggering* in the Sampling configuration Automation tab.

```
Func SampleStartTrigger({trig%});
```

`trig%` Omit this argument for no change. The following values can be used:

- -1 Use whatever state the user previously set in the sampling control panel (default state after `SampleClear()`)
- 0 Sampling is not triggered
- 1 Sampling starts on a trigger; repeated files are not triggered
- 2 Sampling starts on a trigger; repeated files are triggered

Returns The trigger state (as for the `trig%` argument) at the time of the call.

See also:

`SampleRepeats(), SampleReset(), SampleStop(), SampleStart(), SampleStatus()`

SampleStatus()

This function enquires about the state of any sampling.

```
Func SampleStatus();
```

Returns A code indicating the sampling state or -1 if there is no sampling:

-1	No sampling document exists.
0	A time view is ready to sample or sampling is in the process of starting but it has not started
1	Sampling is waiting for a trigger to start it (Trigger input or Event 3 for a 1401)
2	Sampling is now in progress
3	Sampling is stopping (the code changes to -1 when it has stopped)

The sampling process runs asynchronously with respect to the script language. For example, there will be a few milliseconds delay between `SampleStart()` and the `SampleStatus()` return value changing from 0 to 2. Commands that communicate with a sampling system, for example `SampleKey()`, `PlayWavePoints()` and the like will fail if the sampling status is not 2.

See also:
`SampleReset()`, `SampleStart()`, `SampleStartTrigger()`, `SampleStop()`, `SampleText()`, `SampleWrite()`

SampleStop()

This function stops sampling in progress, and is equivalent to the Stop button of the floating command window. The function does not return until sampling has stopped. If used in a file sequence, the sequence is cancelled and the current file is saved.

```
Func SampleStop();
```

Returns 0 if sampling stopped correctly or a negative error code.

See also:
`SampleAbort()`, `SampleReset()`, `SampleStart()`, `SampleStatus()`, `SampleWrite()`

SampleTalker()

This function adds a Talker channel to the Sampling Configuration. If the channel is already in use, it is replaced. See the Sampling Configuration dialog for Talker details. Note that this command adds a sampled channel that selects a talker and a talker port number. The added channel is set to match the current channel type and state of the Talker input port, whatever that may be. In many cases, this will be what you want, but if the talker is configurable, it may be difficult to predict what a particular talker port may generate.

```
Func SampleTalker(chan%, tkr${, port%});
```

`chan%` The channel number to use for the new channel. If you set this to 0, the talker is added to the 'Always Load' list, which allows you to add talkers that do not have associated sampling channels (for instance XTalk which allows you to simulate a keyboard on a remote machine).

`tkr$` The name of the Talker (this is not case sensitive).

`port%` The data channel (defined by the Talker) to add. The first channel is 0. Set 0 or omit if `chan%` is 0. If you omit this it defaults to -1, which is illegal unless `chan%` is 0.

Returns 0 if all went well, or a negative error code.

See also:
`TalkerInfo()`, `TalkerReadStr()`, `TalkerSendStr()`

SampleText()

This function adds text to a text marker channel during sampling. Text marker channels are created with the `SampleTextMark()` command.

```
Func SampleText(text$ {,time {,code%[]|code%{, flags%}}});
```

`text$` The text string to attach to the text marker. If the string is longer than the maximum length set for the channel, extra characters are ignored.

`Time` The time for the text marker. If this argument is omitted, negative, less than the time of the last text marker, greater than the current sampling time (but see `flags%`) or this is the triggered sampling mode trigger channel, then the current sampling time is used. If the time is the same as the last time written to the text marker channel, the time is incremented by one clock tick.

`code%` If an array, the first four elements of this array set the marker codes stored with the text string. If this argument is omitted the codes are set to 0. Codes are limited to the range 0-255. Only the lower 8 bits of codes outside this range are stored. From version [10.19], `code%` can also be an integer holding the 4 marker codes as:
`code0 + 256*(code1 + 256*(code2 + 256*code3)).`

`flags%` Added at version [10.19]. Sum of flag values and taken as 0 if omitted.

- 1 Do not limit the time to the current sample time (allows future times to be set).
- 2 Allow the first code to control sequencer jumps in the same way as keyboard markers. Added at [11.00].
- 4 Allow the first code to trigger a PlayWave area in the same way as keyboard markers. Added at [11.00].

Returns Before Spike2 9.01 and 8.13, this command always returned 0. From 9.01 and 8.13 onwards, it returns -1 if there is no sample document or no TextMark channel, otherwise it returns the time of the TextMark item added to the channel.

Example

The following example is a function you could call during sampling to add a text marker with a code:

```
Func AddTextMark(text$, code%)
var cd%[4];
cd%[0] := code%;  'Set the code
return SampleText(text$, -1, cd%[]); '-1 means the current time
end;
```

See also:

`SampleAbort(), SampleKey(), SampleReset(), SampleStatus(), SampleTextMark()`

SampleTextMark()

This function adds a text marker channel to the Sampling Configuration or reads back the TextMark channel settings. The channel number is forced to be either 30 or one less than the special channel position, if this is set. Each event on a text marker channel holds a time, marker codes and a text string. You can add text markers to this channel using the `SampleText()` command and from a serial line.

Channel create and read information

The first version of the command sets information, the second returns serial line information from the sampling configuration. The title and comment of the channel can be set with `SampleTitle$()` and `SampleComment$()`.

```
Func SampleTextMark(max%{, port%{, term$${, baud%{, bits%{, par%{, stop%{,
hsk%{, rate}}}}}}}});
Func SampleTextMark({&term$${, &baud%{ ,&bits%{, &par%{, &stop%{, &hsk
%}}}}});
```

max% The maximum number of characters that can be attached to each text marker in the range 1 to 200. If 0 is set, the channel is deleted from the list of channels in the sampling configuration. This value is returned by the function when reading back the channel settings. If max% is -1, the return value is the number of pre-set TextMark items are stored in the configuration (see below) and there must be no other arguments.

port% Serial port, in the range 1 to 19, to use to read on-line TextMark data. If omitted, no serial data is read. Use SampleChanInfo() to read back the port. A read back port value of 0 means no serial line input.

term$ Optional terminating character(s) for serial line read. If this is omitted when setting up the serial line, "\r" is used (carriage return, character code 13 or Str$(13)). From Spike2 [10.18] onwards you can use more that one character, for example "\r\n" or Str$(13)+Str$(10) for carriage return line feed. To allow you to embed arbitrary character codes (including 0, which would normally terminate a string), we interpret \xNN (where NN are two hexadecimal digits) as a character code. However, if you do this as a literal string you must double the backslash, otherwise the script compiler will attempt to interpret the backslash.

For example, if your line ended with the character codes 0 followed by carriage return (13), you could write "\\x00\\0x0d". This becomes "\x00\x0d" as the compiler replaces \\ with \ in string literals. Spike2 further interprets this before using it to detect line termination. If you wrote Str$(0)+Str$(13), the script system would see the 0 character and terminate the string before passing it as a terminator.

Before [10.18], input characters with codes less than 32 (control characters) were ignored unless they matched the terminator. Now, all characters are read; however non-terminating characters with code 0 are ignored as they cause strings to end. If you need to read arbitrary binary data from a serial port, you can use an on-line script to use the serial line support.

baud% The serial line Baud rate (number of bits per second). The maximum character transfer rate is about one-tenth this figure. All standard Baud rates from 50 to 115200 are supported. If you do not set a Baud rate, 9600 is used.

bits% The number of data bits to encode a character (7 or 8). If not set, 8 data bits are used.

par% Set this to 0 for no parity check, 1 for odd parity or 2 for even parity. If you do not set parity, 0 is used for no parity.

stop% The number of stop bits as 1 or 2. If not set, 1 stop bit is used.

hsk% The handshake mode, sometimes called flow control. 0= no handshake, 1=hardware handshake, 2=XON/XOFF protocol. If not set, 0 is used (no handshake).

rate Expected sustained average data rate in text markers per second. If omitted, the rate is set to 1.

Returns 0 or a negative error code when setting the channel. When reading the settings, the return value is the maximum number of characters to store in the text marker, or 0 if the channel is not enabled. From version [11.00] if max% is -1 the returned value is the number of pre-set items).

The rate is somewhat awkwardly placed. If you want to set the rate to something other than 1 and are not using the serial input you must still set reasonable serial parameters. For example, to set 100 characters at 200 Hz without using the serial input you could use:

```
SampleTextMark(100, 0, "\r", 9600, 8, 0, 1, 0, 200.0);
```

Manage TextMark pre-set items

These command variants were added at version [11.00]. They give access to the list of preset TextMark items that are stored in the sampling configuration. The first of these two variants sets a TextMark item in the list. If the text of the item matches an existing item (by a case-insensitive match), the existing item is deleted before the new item is added. There is an arbitrary limit of 200 items. The second variant reads back items that are already present in the list.

```
Func SampleTextMark(code%|code%[], text$);
Func SampleTextMark(&code%|code%[], &text$, n%);
```

code% If an array, the first four elements of this array set/get the marker codes stored with the text string. Codes are limited to the range 0-255. Only the lower 8 bits of codes outside this range are stored. If an integer, it holds the 4 marker codes as:
code0 + 256*(code1 + 256*(code2 + 256*code3)).

`text$` The text string attached to the text marker. When setting, the string is limited to the maximum length set for the channel, extra characters are ignored.

`n%` When reading back the pre-set items. this is the 0-based index of the item. If `n%` is outsize the range 0 to the number of items -1, the returned text is empty and the returned codes are 0. You can use `SampleTextMark(-1)` to get the number of pre-set items. It is an error to set `n%` to 200 or more.

Return When adding an item, 1=Item added, 0=item replaced, -1=list is full. When reading, 1=item read, 0=`n%` out of range.

Set link flags

This command variant was added at version [11.00]. It gives access to the link flags used for interactive TextMark setting during sampling. It is equivalent to the check boxes in the Pre-set TextMark list dialog.

```
Func SampleTextMark(-2{, flags%});
```

`flags%` If omitted or -1, no change is made. If present, the sum of the following flag values:

1 Lock the settings so users cannot change the list or the link flags.

2 Allow the first TextMark code to control output sequencer jumps in the same way as keyboard markers or the sequencer control panel.

4 Allow the first TextMark code to trigger a PlayWave area in the same way as keyboard markers.

Return The `flags%` value at the time of the call before any change.

See also:
`SampleChanInfo(), SampleComment$(), SampleClear(), SampleTitle$(), SampleText()`

SampleTextMarkLink()

This command is equivalent to the Match, Sequencer link and PlayWave fields of the Sampling configuration TextMark channel setup dialog. It sets the link between serial line TextMark input and the output sequencer and arbitrary waveform output and optionally allows you to override the method used to extract the code from the input. Setting a link allows the first code associated with the serial line TextMark input to cause the output sequencer to jump to a matching code and/or the arbitrary waveform output to play a waveform. This command was added at [11.00]. If you use the `SampleText()` command to generate markers, this has separate controls for the links and does not depend on this command.

There are two command variants, one to set the state in the sampling configuration and one to read it back:

Set flags and match

The first variant sets the flags and the text to used to locate and decode the marker code in the serial line input.

```
Func SampleTextMarkLink({flags%{, match$}};
```

`flags%` If omitted or -1, no change is made. If present, the sum of:

1 Reserved and should not be used. Present to make the remaining flags match the `SampleText` command.

2 Allow the first TextMark code to control output sequencer jumps in the same way as keyboard markers or the sequencer control panel.

4 Allow the first TextMark code to trigger a PlayWave area in the same way as keyboard markers.

`match$` This string sets the regular expression used to locate the code in the input. If omitted no change is made. The format of the string is the same as the format of the Match field in the Sampling configuration TextMark channel setup dialog.

Return The `flags%` value set at the time of the call.

Get flags and match

The second variant gets the text string used to find the code in the serial input.

```
Func SampleTextMarkLink(&match$);
```

match$ A string that is returned holding the regular expression used to locate the code with an optional code format (n^, h^ or c^) on the front.

Return The flags% value set at the time of the call.

See also:
SampleTextMark(), Sampling configuration

SampleTimePerAdc()

This sets and gets the number of time units set by SampleUsPerTime() that pass between each ADC conversion. This ADC conversion rate is then shared out amongst the waveform channels in the sampling configuration. This is the script equivalent of the Time units per ADC convert field in the Resolution tab of the Sampling Configuration dialog.

The product of the time per ADC and the microseconds per unit time must not exceed the capabilities of the 1401 you are going to use, see the Sampling Configuration dialog for details. Lower values will cause sampling to fail. If the optimise method is set by SampleOptimise() is not 0, this call will have no effect as time per ADC will be recalculated.

We *strongly* suggest that unless you really understand how calculating sample rates works or need to achieve a very particular set of rates, you *always* let SampleOptimise() work out the best values to use for a particular sampling configuration. It is possible that future hardware used for data capture will not make use of this value at all.

```
Func SampleTimePerAdc({new%});
```

new% The number of clock ticks per conversion in the range 1 to 32767. If this is omitted the value is not changed. Illegal values stop the script with a fatal error.

Returns The value of the time per ADC convert at the time of the call.

See also:
SampleOptimise(), SampleUsPerTime(), SampleWaveform(), SampleWaveMark()

SampleTitle$()

This gets and sets the title attached to a channel in the Sampling Configuration dialog.

```
Func SampleTitle$(chan% {,new$|exp%});
```

chan% The channel number.

new$ If present, the new title. If the title is too long, it is truncated. From version [10.09] you can include place holder strings as described in the channel configuration dialog.

exp% 0 or omitted to return the title without expanding any replacement fields, 1 to expand them.

Returns The title at the time of the call, or an empty string for illegal channel numbers.

See also:
SampleCalibrate(), SampleComment$(), SampleClear()

SampleTrigger()

This command has the same effect as using the Sampling Configuration Sampling Mode Tab and setting Triggered mode. We expect you to use this command after you have created all the channels in the sampling configuration.

This controls the four triggers that can be set to determine which data for a channel is written to disk. Each trigger is associated with a list of channels. The first trigger is also set by SampleMode() for backwards compatibility. Normally, the SampleMode() command is used to set Continuous mode (the default after SampleClear()), or to set Timed mode and is not used for setting Triggered mode.

You cannot change the triggering after sampling has started. Channels not associated with a trigger are written continuously. If you save data to a 64-bit `.smrx` data file, Spike2 will restrict the saved data to only the time range you request (except for Level Event data). If you sample to a 32-bit smr file, data is written in blocks and you will usually get at least the data you asked for, but you will usually get additional data saved.

```
Func SampleTrigger(trig%, src%, code$|code%, relS, relE{, cSpc{, scr${,
atEnd%{, off$|off%}}}});
Func SampleTrigger(trig%, get%{, &get$});
```

`trig%` A trigger number in the range 1 to 4. Setting a trigger sets triggered sampling mode. This is cancelled by `SampleClear()` or by calling `SampleMode()`.

`src%` The source channel for this trigger number. This can be any event-based channel number that is part of the data file; you cannot use a duplicate, memory or a virtual channel number. Setting a channel number greater than the return value of `SampleChannels()` or less than 1 causes nothing to change except for the special case of setting channel -3 which cancels this trigger (but still sets triggered sample mode). Setting a channel of the wrong type causes no change. When reading back the trigger state, any value less than 1 means 'no trigger set'.

`code` If the `src%` channel is Marker-based, you can choose to trigger only when the marker code matches one code or a range of codes. If the source is not a marker, `code` is ignored. When this is an integer you can choose to only trigger when the first marker code matches this value (in the range 0 to 255). Set -1 to accept all markers.

From [10.06] this argument can be a string holding a marker filter specification or an empty string to accept all markers.

`relS` The start of the triggered area relative to the trigger time, in seconds. This can be negative (to capture pre-trigger data) or positive, but must be less than `relE`. There are limits on how far into the past a trigger can go (based on data buffering); there is a warning message when sampling starts if the triggering requests are likely to be impossible.

`relE` The end of the triggered area relative to the trigger time (and can be negative). This must be greater than `relS`.

`cSpc` A channel specifier for the channels to trigger. If you omit this argument, or set -1, all channels are selected for triggering. You cannot use the standard channel specification values -2 or -3. You can use -6 for no channel, when we expect you to set a script. Unlike the trigger `src%` channel, we do not check that channels nominated for triggering exist.

`scr$` From Spike2 version [10.09] you can supply the path to a script to associate with this trigger. For a script to run, no other script must be running and the system must have idle time. Scripts run in response to triggers typically have no user interface and run and exit very quickly. We do not limit what such scripts can do; once started these scripts behave as if you had run them interactively. There is no check that `scr$` exists, however at run time if it does not exist it will not run!

`atEnd%` 0 to run the script as soon as possible after the trigger event is detected. Set to 1 to run the script `relE` seconds (must be greater than 0) after the trigger. This argument was added at version [10.09].

`off` If the `src%` channel is Marker-based, you can set a marker code or marker filter specification to match to terminate the trigger early. The argument type must match `code`; if `code` is an integer, `off` must be an integer. If `code` is a string, `off` must be a string. Added at [10.13].

`get%` You can use the second command variant to return trigger values. Get values are:

- -1 `src%`
- -2 `code%`, `get$` (if present) holds the code as a marker filter specification. The `code%` value is represented exactly if the marker filter specification is a single code in the first layer, or empty. Otherwise `code%` is the first code set in the first filter layer; `get$` has the full specification.
- -3 `relS`
- -4 `relE`
- -5 `cSpc` (-1=all, 0=multiple, 1-n for one, -6 for none. `get$` (if present) holds the channel specification, as text.
- -6 `AtEnd%`, `get$` (if present) holds the script name. Added at [10.09].
- -7 `off%`, `get$` (if present) holds the `off` code as a marker filter specification (the returned `off %` has the same limitation as the returned `code%`). Added at [10.13].

get$ Used when get% is -2 to get the code$ argument, -5 to get a text channel list, -6 to get the scr$ argument and -7 to get the off argument as text.

Returns The first variant returns 0 or an error code. The second variant returns the requested value.

See also:
Channel specifiers, SampleMode()

SampleUsPerTime()

This gets and optionally sets the basic time unit used for sampling. If the optimise method is set to full by SampleOptimise(2,...), changes to the basic time units have no effect as Spike2 chooses the best value to optimise the waveform sample rates. This is the script equivalent of the Microseconds per time unit field in the Resolution tab of the Sampling Configuration dialog.

```
Func SampleUsPerTime({new});
```

new If present, this sets the basic time unit in the range 1-1000 microseconds. Out of range values cause a fatal script error. If you sample with a Power1401 or Micro1401 mk II or -3, the value is rounded to the nearest 0.1 microseconds. If you are sampling to a 64-bit smrx file you want to set this to the smallest value (best time resolution) that is compatible with the desired sample rates.

Returns The current value of microseconds per time unit.

See also:
SampleOptimise(), SampleTimePerAdc()

SampleWaveform()

This function adds a waveform or RealWave channel to the list of channels required. If the channel is already in use, it is replaced. The units, scale and offset fields are set to the standard Spike2 defaults (input in Volts); use SampleCalibrate() to change them. The title and comment of the channel can be set with SampleTitle$() and SampleComment$().

```
Func SampleWaveform(chan%, port%, ideal{, real%});
```

chan% The channel number to use for the new channel in the range 1 to the maximum allowed by your hardware.

port% An unused (by a waveform channel) 1401 waveform port in the range 0 to the maximum allowed by your hardware.

ideal The ideal sampling rate that you would like for the port in Hz. Remember that you may not get this rate. Spike2 will set the nearest rate it can.

real% If this is omitted or 0, this adds a waveform channel to the configuration. If this is 1, it adds a RealWave channel. Support for RealWave channels was added at version 9.01.

Returns 0 if all went well, or a negative error code.

See also:
SampleCalibrate(), SampleChanInfo(), SampleComment$(), SampleClear(), SampleTimePerAdc(), SampleTitle$(), SampleUsPerTime(), SampleWaveMark()

SampleWaveMark()

This command adds a WaveMark channel to the sampling configuration and returns information about a WaveMark channel. See the sampling configuration dialog for details. The units, scale and offset fields are set to the Spike2 defaults (input in Volts); use SampleCalibrate() to set this. The title and comment of the channel can be set with SampleTitle$() and SampleComment$(). See the Sampling Configuration dialog for sample rate details. The first command version sets the channel, the second returns information:

Set the channel

```
Func SampleWaveMark(chan%, port%|const port%[], rate, size%, pre%{,
ideal{, nTr%}});
```

`chan%` The channel number of the WaveMark channel in the range 1 to the maximum number allowed by your hardware.

`port%` The 1401 waveform port in the range 0-127 or an array holding the ports to use. If an array is supplied, the ports are copied from the first `nTr%` elements. If the array is too short, ports for array index `i%` are set to `port%[0] + i%`.

`rate` The estimated maximum sustained spike rate, used to allocate buffer space.

`size%` The number of waveform samples to save for each WaveMark This must be in the range 6 to 126. Before [9.13, 10.09], `size%` had to be an even number regardless of the number of traces. From [9.13, 10.09] onwards, this is only the case for a single trace.

`pre%` The number of points before the peak/trough of each spike, range 0 to `size%`-1.

`ideal` If present, this sets the ideal sample rate, in samples per second, for all WaveMark channels. This must be in the range 0 to 500000.0 Hz. Rates of 1 Hz or less are ignored and do not change the WaveMark rate. We suggest that you use the value 0 to leave the rate unchanged. The last call to `SampleWaveMark()` with a valid rate sets the rate for all channels.

`nTr%` If present, this sets the number of traces to sample as 1, 2 or 4. The default is 1.

Returns 0 for success or a negative error code.

Read back information

```
Func SampleWaveMark(chan%{, &pre%{, &nTr%}});
```

`chan%` The channel number of the WaveMark channel in the range 1 to maximum number allowed by your hardware.

`pre%` If present, returned as the number of points before the peak/trough of each spike.

`nTr%` If present, returned as the number of traces to sample.

Returns `size%` (points per trace) or 0 if this is not a WaveMark channel.

You can use `SampleChanInfo()` to read back the `port%`, `rate` and `ideal` fields.

See also:

`SampleCalibrate()`, `SampleChanInfo()`, `SampleComment$()`, `SampleClear()`, `SampleTimePerAdc()`, `SampleTitle$()`, `SampleUsPerTime()`, `SampleWaveform()`

SampleWrite()

This command controls saving data to the file during sampling. It can be used when the output data file exists, even before sampling starts. There are two command variants:

```
Func SampleWrite(write% {,chan%|const chan%[]});
Func SampleWrite(write% ,chan%|const chan%[], from{, upto});
```

`write%` This determines the action taken by the command:

- -1 Report on the write state of the channel (or channels) given by the next argument.
- 0 Disable writing to disk (pause) the channels given by the next argument
- 1 Enable writing to disk (un-pause) the channels given by the next argument
- 2 If writing is enabled, mark a range defined by `from` and `upto` for saving to disk (for Triggered or Timed modes). This is pointless in Continuous mode as data is already being saved.
- 3 Mark the range defined by `from` and `upto` for saving to disk even if writing is disabled (for use in Continuous mode with save disabled)

`chan%` This sets the channels to operate on. If this is an integer, it is the channel number (0 or less for all channels). If it is an array, index 0 holds the number of channels following; the remaining elements hold a list of channel numbers. If omitted, all channels are used.

`from` Only used with `write%` as 2 and 3 to set the start time of a range to be saved to disk. Time ranges for saving can be set in the past (as far back as buffering allows) and in the future. Negative times are treated as zero.

`upto` Only used with `write%` as 2 and 3 to set the time up to which data is saved. If omitted or negative, saving continues to the end of the file.

Returns The overall state of writing for the channel or channels or a negative error code if no sampling document:

0 Disabled 1 Enabled 2 Some channels in list are enabled

The command can be used in the following ways:

Report the write enable state

```
Func SampleWrite(-1 {,chan%|chan%[]});
```

Calls with write set to -1 report on the state of writing for the channels set by the `chan%` argument or all channels if it is omitted. This is the state as set by **`SampleWrite(0|1,...)`**. You can use this at any time after the data file is opened.

Set overall channel save data policy

```
Func SampleWrite(0|1 {,chan%|chan%[]});
```

Calls to this command with `write%` set to 0 (disable) or 1 (enable) with 1 or two arguments set the overall save data policy for one or all channels. If you enable or disable all channels, this will set or clear the Write to Disk check box in the Sample control bar. If you disable a channel, no data will be saved until you enable it again or mark a range to be saved (see below).

In Timed and Triggered sampling modes, the timing and triggering continues, but if you set the overall channel policy to disabled, no data is written, regardless of the timing and triggering. Timed and triggered data will be written again if you enable writing on the channel.

You can set the channel save policy as soon as you create a file ready to sample and change it at any time until sampling stops. The initial state is applied when sampling starts. Changes are applied from the current sample time and overwrite any range-specific save commands from the current time onwards.

Mark a specific time range to be saved

```
Func SampleWrite(2|3 ,chan%|chan%[], from{, upto});
```

Calls to this command mark a range of data on one or more channels to be saved to disk. You can mark time ranges in the future or in the recent past. Spike2 buffers several MB of data, so you can make retrospective decisions about keeping data. If you omit the `upto` argument, the range extends to the end of time.

Time ranges can only be set after sampling starts as the initial overall channel save state is set then and wipes any range-based save commands. Ranges set by this command and by timed or triggered sampling are logically ORed together, that is, overlapped ranges are merged. Time ranges are used exactly with 64-bit files; with 32-bit files they mark 32kB sized data blocks for saving. Marking a range was added in Spike2 version 8.

Add to timed or triggered data (write% = 2)

With `write%` set to 2, this only has any effect when the overall write policy for a channel is set to enabled. This means it is only useful in Timed and Triggered sampling modes where it has the effect of adding extra data into the list of data to be saved.

Add save range when overall state is disabled (write% = 3)

With `write%` set to 3, this marks a time range for saving, even if the overall state is set to disabled. This is for use in continuous sample mode with the overall state set to disabled. It allows a script to make decisions about saving past or future data based on the experimental circumstances.

Interaction with data flushing

If you use the `Modified(0,0)` command or the Sampling configuration Automation tab Flush to disk option, you then cannot mark time ranges for saving that fell before the last saved data. Put another way, these flushing options will write all buffered data marked for writing. For example, if writing is off for a channel apart from a section from 100 to 110 seconds, and all this data from 0 up to the current time (say 200 seconds) is in the buffer and a flush option is used, the section from 100 to 110 seconds will be written to the disk. Although you

can still use the data from 0 to 100 seconds within Spike2 (it is still in the buffer), you cannot now mark it for saving. Data from 110 seconds onwards can still be marked for saving.

Level event channels

At the current time, level event data is always saved to ensure that we keep track of high and low levels correctly. This may be addressed in a future release where we make sure that we delete an even number of edges.

See also:

`SampleAbort()`, `SampleMode()`, `SampleReset()`, `SampleStart()`, `SampleStop()`, Sampling mode

ScriptBar()

This controls the Script toolbar and gets information, adds buttons and removes button. Call the command with no arguments to return the number of toolbar buttons. The first button is numbered 0. Script bar strings are saved in the system registry; see the `Profile()` command for more information.

```
Func ScriptBar({nBut%, &get$});
Func ScriptBar(set$);
Func ScriptBar(-1{, match$});
```

`nBut%` A 0-based button number and returns -1 if the button does not exist, 0 if it is the last button, and 1 if higher-numbered buttons exist. `get$` returns the information as for `set$`.

`set$` This holds up to 8 characters of button label, a vertical bar, the path to the script file including `.s2s`, a vertical bar and a pop-up comment. The function returns the new number of buttons or -1 if all buttons are already used.

`match$` Identifies the button(s) to remove. If the first argument is -1 and `match$` is omitted or empty, all buttons are cleared and the return value is 0. Otherwise, `match$` is a case insensitive regular expression to match. You are attempting to match the same string as was used to set the button, so you can attempt to match the button text or the file name or the tool tip, or a combination. The function returns the number of deleted buttons. Clearing a matched button was added at [10.15].

Returns See the descriptions above. Negative return values indicate an error.

For example, the following code clears the script bar and sets a button:

```
ScriptBar(-1);    'clear all buttons
ScriptBar("ToolMake|C:\\Scripts\\ToolMake.s2s|Build a toolbar");
```

The following, somewhat unexpected code, clears the button added above:

```
ScriptBar(-1, "^ToolMake\\|");
```

Explanation: In a regular expression, the `^` matches start of line (used so we do not match a line with `ToolMake` in the path) and `ToolMake` matches as you would expect. If we matched with `"^ToolMake"` this would work, but it would also match any longer label, such as `"ToolMaker"`. We need to match the terminating character | to be certain that we have all the label text.

In a regular expression, the vertical bar character is used for alternatives; `"cat|dog"` matches `"cat"` or `"dog"`. Even more confusingly, using `"^ToolMake|"` as a match will match anything! This is because the alternative, to the right of the vertical bar, is an empty string, which means *'I have nothing to match, so match anything*'.

The solution is to use the backslash as an escape character, *meaning remove any special meaning from the following character*. Sadly, if you use `"^ToolMake\|"`, this also matches everything because the script compiler also uses backslash as an escape character in literal strings, so `"\|"` becomes `"|"`. Using `"^ToolMake\\|"` works because the script compiler translates this to `"^ToolMake\|"`, which is what the regular expression requires.

See also:

`App()`, Script toolbar

ScriptRun()

This sets the name of a script to run when the current script terminates. You can think of this as *chaining* scripts. You can pass information to the new script using disk files or by using the `Profile()` command. You can call this function as often as you like; only the last use has any effect.

```
Proc ScriptRun(name$[, flags%]);
```

`name$` The script file to run. You can supply a path relative to the current folder or a full path to the script file. If you supply a relative path, it must still be valid at the end of the current script. Set `name$` to `""` to cancel running a script.

`flags%` Optional flags, taken as 0 if omitted. Sum of: 1 = run even if the current script ends in an error, 2 = keep loaded script in memory

If the file you name does not exist when Spike2 tries to run it, nothing happens. If the nominated script is not already loaded, Spike2 will load it, run it and unload it unless the keep loaded script in memory flag is set. If a loaded script calls `Yield()` or calls any function that allows the system to idle (`Toolbar()`, `DlgShow()`...), the script can be unloaded while it is still running. This is usually harmless unless the loaded script attempts to use `App(3)`, which will return 0 if the script is no longer in memory.

If you use debug and then stop the script executing, any script set to run is also cancelled.

See also:
`App()`, `Profile()`

Seconds()

This returns the time in seconds. This is used for relative time measurements. You can also set the timer. If you want the time into sampling, use the `Maxtime()` function. To find out how much time built-in script functions are using, see `DebugList()`.

```
Func Seconds([set[, hiRes]]);
```

`set` If present, this sets the time in seconds. The time is 0 when Spike2 starts.

`hiRes` If present, this selects between normal or high-resolution timing. A zero value sets the standard resolution of nominally 1 millisecond (but may be worse on some systems). Set 1 for the highest resolution available. Note that the default is 1 millisecond resolution, set when Spike2 starts. Changes to the resolution are persistent between scripts.

Returns If `hiRes` is not present, the function returns the time in seconds. This is the value before any new time is set. If `hiRes` is present, the return value is the time resolution, in seconds. This is 1 millisecond for the standard resolution and can be (much) less than 1 microsecond for the high resolution timer.

High resolution timer caveats

The function we use for the high-resolution timer is supposed to use a fixed frequency source. However, some machines (incorrectly) use a time based on CPU cycles; this is a problem for many laptops which change the CPU speed to save battery power. There are also reports that some multi-core machines report different times for each core. If you have a problem, the following web links may be useful: South bridge chip set problem, Athlon X2 problem. Note that computers from 2007 onwards are less likely to have problems.

See also:
`Date$()`, `DebugList()`, `MaxTime()`, `Time$()`, `TimeDate()`

Selection()

This is used with a grid view and the cursor regions and cursor values dialogs (and text-based views from [10.15]) to get the information on the selection. It is an error to use this with any other view type. A grid view can have multiple selections, which may overlap. Each selection is rectangular.

```
Func Selection([n%[, pos%[4]]]);
```

`n%` Optional argument that takes the value 0 if omitted. If zero, the command returns the number of selections. If greater than zero, this is the selection number to return information about.

`pos%` An optional array of at least 4 integer values. Indices 0 and 1 return the 0-based column and row (line) of the selection top left. Indices 2 and 3 return the column and row (line) of the selection lower right. If `n%` exceeds the number of selections, the returned values are for the last selection. When used in a text-based view, the row value is returned as a 1-based line number not a 0-based row.

Returns The number of selections if `n%` is 0 or omitted or -1 if `n%` exceeds the number of selections, otherwise 0.

Grid views
Grid views report selections in the data areas only, with column 0 at the left and row 0 at the top.

Cursor dialogs
Cursor dialogs can also report selections in the headers (which have index -1). A selection that starts at a row or column of -1 will extend to the end of that row or column.

Text Views (added at [10.15])
When used in a text view, there is another command variant:

```
Func Selection(n%, &sEnd%);
```

`n%` Currently the only value that is allowed is 1, which gets the position of main text selection. If you use any other value, the command returns -1 and `sEnd%` is unchanged.

`sEnd%` Returned holding the character position of the end of the selection. This can be less than the position of the start if the selection was backwards.

Returns The character position of the start of the selection or -1 if the selection number is out of range (currently only 1 is allowed).

Although text views do support multiple and rectangular selections, we do not support them from the script. `Selection(0)` always returns 1 (as there is always a current position) and we define the first selection as being the active one. Should we choose to support multiple selections from the script language in future, the additional selections will be with the first argument greater than 1.

See also:
`MoveBy()`, `MoveTo()`

Selection$()

This function returns the text in the current view that is currently selected. In a Grid view, selection is by cell, not within a cell and does not include row and column headings. You can also use this in the Cursor regions and values dialogs to get the text of the selected cells (including the row and column headings).

```
Func Selection$();
```

Returns The current text selection. If there is no text selected, or if the view is inappropriate for this action, an empty string is returned.

See also:
`EditCopy()`, `EditCut()`, `EditPaste()`, `MoveBy()`, `MoveTo()`

SerialClose()

This function closes a serial port opened by `SerialOpen()`. Closing a port releases memory and system resources; any characters held in the `SerialWrite()` output buffer waiting to be transmitted will be lost. Ports are automatically closed when a script ends, however it is good practice to close a port when your script has finished with it.

```
Func SerialClose(port%);
```

port% The serial port to close as defined for SerialOpen().

Returns 0 or a negative error code.

You can find the size of the SerialWrite() output buffer by using SerialWrite(port%) immediately after SerialOpen(), then monitor buffered characters before using SerialClose() if loss of buffered characters is an issue in your application.

TextMark channels

If you use this command to close a serial port that is in use during sampling for a TextMark channel, this will shut down the port and no more TextMark data will be captured.

See also:

SerialOpen(), SerialWrite(), SerialRead(), SerialCount()

SerialCount()

This counts the characters or items buffered in a serial port opened by SerialOpen(). Use this to detect input so your script can do other tasks while waiting for serial data. There is an internal buffer of 1024 characters per port that is filled when you use SerialCount(). The size of this buffer limits the number of characters that this function can tell you about. To avoid character loss when you are not using a serial line handshake, do not buffer up more than a few hundred characters with SerialCount().

```
Func SerialCount(port% {,term$|term%|term%[]});
```

port% The serial port to use as defined for SerialOpen().

term Optional argument that defines a 'line' terminator. If this is a string, it holds the characters that terminate the line and cannot include the character code 0. If an integer, it must be in the range 0 to 255 and defines a single character to terminate the line. If an integer array, the lowest byte of each array element forms the termination string (which can include zeros). See SerialRead() for more explanation.

Before version [10.18], this argument could only be a string.

Returns If term is absent or an empty string or array of size 0, this returns the number of characters that could be read. If term is set, this returns the number of complete items that end with term that could be read.

See also:

SerialOpen(), SerialWrite(), SerialRead(), SerialClose()

SerialOpen()

This command opens a port for use and from and can also report on the number of available COM ports. There are two variants

Open a port for use

This function opens a serial (RS-232) port and configures it for use by the other serial line functions. It is not an error to call SerialOpen more than once on the same port. The serial routines use the host operating system serial line support. Consult your system documentation for information on serial line connections and Baud rate limits.

```
Func SerialOpen(port%{, baud%{, bits%{, par%{, stop%{, hsk%}}}}});
```

port% The serial port to use, in the range 1 to 256. The number of ports depends on the computer. Modern machines usually only implement serial interfaces through USB plug-in devices and the port numbers are device dependent. The maximum port number was 9 before Spike2 version 7.09.

baud% This sets the serial line Baud rate (number of bits per second). The maximum character transfer rate is of order one-tenth this figure. All standard rates from 50 to 115200 Baud are supported. If you omit baud%, 9600 is used.

`bits%` The number of data bits to encode a character. Windows supports 4 to 8 bits, the Macintosh supports 7 or 8. If `bits%` is omitted, 8 is set. Standard values are 7 or 8 data bits. If you set 7 data bits, character codes from 0 to 127 can be read. If you set 8 data bits, codes from 0 to 255 are possible. If in doubt, use 8 data bits.

`par%` Set this to 0 for no parity check, 1 for odd parity or 2 for even parity. If you do not specify this argument, no parity is set. If in doubt, set no parity.

`stop%` This sets the number of stop bits as 1 or 2. If omitted, 1 stop bit is set. If you specify 5 data bits, a request for 2 stop bits results in 1.5 stop bits being used. If in doubt, set 1 stop bit.

`hsk%` This sets the handshake mode, sometimes called “flow control”. 0 sets no handshake, 1 sets a hardware handshake, 2 sets XON/XOFF protocol. No handshake is the simplest thing to do. If you set a hardware handshake, this relies on extra pins being correctly connected on the serial port. The XON/XOFF protocol achieves flow control by sending special characters, so does not require special connections.

Returns 0 or a negative error code. The error codes are negated Windows system errors. You can get a complete list by searching the web for "Windows System Error Codes". These errors are not understood by the `Error$()` command. Likely values are:

Code	Error	Likely explanation
-2,-3	File or Path not found	The port does not exist.
-5	Access Denied	The port is in use, but not by the script language.

Get a list of COM ports [10.11]

This command variant gets the number of ports, their port numbers and a text description.

```
Func SerialOpen({ports%[]{, names$[]}});
```

`ports%` If present, an array of integers that is set to the detected port numbers (in ascending order). If there are more ports that will fit in the array, the extra ones are discarded.

`names$` If present, an array of strings that is set a text description of the port number held in the corresponding element of `ports%[]`. If there are more ports that will fit in the array, the extra ones are discarded.

Returns The number of serial ports.

The following example displays the available serial ports:

```
var n% := SerialOpen();          'Get number of ports
var ports%[n%], names$[n%], i%;
n% := SerialOpen(ports%, names$);
for i% := 0 to n%-1 do
    PrintLog("%2d %s\n", ports%[i%], names$[i%]);
next
```

The result on my computer, which has a built-in COM port and one USB plug-in port, was:

```
 1 Communications Port (COM1)
 7 Prolific USB-to-Serial Comm Port (COM7)
```

Use with the TextMark sampling channel

From Spike2 version [8.04], the script language can access the same serial port that is being used during data sampling to input text into the TextMark channel. Ports used for sampling are opened when the sampling document is created and closed when sampling stops. You can use the port (usually with `SerialWrite()`) without calling `SerialOpen()` and rely on the sampling to open it for you. However, the port may close (due to sampling stopping) at an inconvenient moment. You can avoid this by calling `SerialOpen()` from the script. In this case, the port remains open until both sampling and the script have closed it. However, the act of opening a port also reconfigures it and flushes all data, which can also have undesirable effects. Having the port opened by the script will not prevent the sampling system from sharing the port (and setting the sampling Baud rate and other parameters). We may extend the serial system in the future to allow us to avoid reconfiguring the port in such a shared configurations.

We do not stop you using the `SerialRead()` command when the port is shared. However, this will cause competition between the TextMark input and the script language. You may be able to make this work in the

case where the attached device only generates input when you write to it. In this particular case, as long as there is no opportunity for the system to idle between your `WriteSerial()` command that asks for input and the `ReadSerial()` command that collects it, you may be able to get it to work. However, the `ReadSerial()` command reads as much input data as it can into a private buffer and extracts what it needs from this buffer. The TextMark input reads characters directly from the COM device, so will not see characters that have been grabbed by `ReadSerial()`.

See also:
`SerialWrite(), SerialRead(), SerialCount(), SerialClose()`

SerialRead()

This function reads characters, a string, an array of strings, or binary data from a nominated serial port that was previously opened with `SerialOpen()`. Binary data can include character code 0; string data never includes character 0. See the `SerialWrite()` command for notes about using this command during sampling with a serial port in use as a TextMark channel.

```
Func SerialRead(port%, &in$|in$[]|&in%|in%[]{,term$|term%|term%[]{, max
%}});
```

`port%` The serial port to read from as defined for `SerialOpen()`.

`in$` A single string or an array of strings to fill with characters. To use an array of strings you must set a terminator or all input goes to the first string in the array.

`in%` A single integer (`term$` and `max%` are ignored) or an array of integers (`term$` and `max%` can be used) to read binary data. Each integer can hold one character, coded as 0 up to 255. The function returns the number of characters returned.

`term` Optional argument that defines a terminator. If this is a string, it holds the characters that terminate the line and cannot include the character code 0. If an integer, it must be in the range 0 to 255 and defines a single character to terminate the line. If an integer array, the lowest byte of each array element forms the termination string (which can include zeros).

If this is an empty string or omitted or is an array of size zero, all characters read are input to the string, integer array or to the first string in the string array and the number of characters read can be limited by `max%`. The function returns the number of characters read.

If `term` is not empty, the contents are used to separate data items in the input stream. Only complete items are returned and the terminator is not included. For example, set the terminator to `"\n"` or `10` if lines end in line feed, or to `"\r\n"` if input lines end with carriage return then line feed. If `in$` is a string, one item at most is returned. If `in$[]` is an array, one item is returned per array element. The function returns the number of items read unless `in` is an integer (`in%` or `in%[]`) when it returns the number of characters returned.

Before version [10.18], this argument could only be a string.

`max%` Optional. If omitted or set to 0, the value 255 is used. This sets the maximum number of characters to read into each string or into the integer array. If a terminator is set, but not found after this many characters, the function breaks the input at this point as if a terminator had been found. There is a maximum limit set by the integer array size, the size of the buffers used by Spike2 to process data and by the size of the system buffers used outside Spike2. This is typically 1024 characters.

Returns The function returns the number of characters or items read or a negative error code. If there is nothing to read, it waits 1 second for characters to arrive before timing out and returning 0. Use `SerialCount()` to test for items to read to avoid a time out.

See also:
`SerialOpen(), SerialWrite(), SerialCount(), SerialClose()`

SerialWrite()

This writes one or more strings or binary data to a serial port opened by `SerialOpen()` or for sampling to a TextMark channel. Use the command with a single argument to find out how much space is available in the `SerialWrite()` output buffer (typically 1024 characters).

```
Func SerialWrite(port%{, out$|const out$[]|out%|const out%[]{, term$}});
```

`port%` The serial port to write to as defined for `SerialOpen()`. You can also write to a port opened for sampling as a TextMark channel; use `SampleChanInfo()` to read back the port.

`out$` A single string or an array of strings to write to the output. The return value is the number of strings written. If you want to embed a character code 0 in the stream you cannot send it using a string as a zero will act as a string terminator. Use the `out%` variant.

`out%` A single integer or an integer array to write as binary. One value is written per integer. The output written depends on the number of data bits set for the port; 7-bit data writes as `out% band 127`, 8-bit data writes as `out% band 255`. The return value is 1 if the transfer succeeded. You can output zeros with this.

`term$` If present, it is written to the output port after the contents of `out%`, `out%[]` or `out$` or after each string in `out$[]`.

Returns For success, the return value is as documented for the `out` argument. If there is no room in the output buffer for the data the return value is -1 except when `out$[]` is used when the return values is the count of strings actually sent. When called with a single argument, the command returns the number of free characters in the output buffer.

If you use the `SerialClose()` command before the system has had time to write buffered characters to the serial port, the buffered characters will be lost.

Use with TextMark serial channels

During data sampling, you can write to a serial port linked to a TextMark channel. The serial port used for TextMark channels opens when the data view is created, before sampling starts. The port is not tested for input until sampling is told to start, so there is a time period after the data file opens and before sampling starts in which you can both write to the serial port (to set up whatever device is in use) and read back responses without the data capture system grabbing the data.

Once sampling is told to start, even if sampling is waiting for a triggered start, the serial port is monitored for input. You can write to the port once sampling has been told to start, but reading data with `SerialRead()` will compete with the sampling. The TextMark channel grabs serial data in system idle time and is also given the opportunity to grab data periodically while a script runs, so will not be reliable. The only reliable method to read responses is to wait for the response to be sampled by the TextMark channel as data and read it from the channel.

Spike2 knows the difference between a port opened for a TextMark channel and one opened by `SerialOpen()`. If the port was opened for a TextMark channel, it will remain open, even if you use `SerialClose()` on it. Likewise, if you open a port with `SerialOpen()`, start and stop sampling using the same port for a TextMark channel, the port will remain open. However, the act of opening a port sets the port parameters and flushes any content, so you must take care.

There is no need to call `SerialOpen()` to allow access to the port linked to the TextMark channel. However, if you do not do this, the port will close when sampling stops.

See also:

`SerialOpen(), SerialRead(), SerialCount(), SerialClose()`

Set...()

The `Set...()` family of commands create result views attached to the current time view or to the time view associated with the current result view. Apart from `SetResult()`, which creates a result view that is not dependent on a time view, these routines match the Analysis menu New Result view commands. They do not

update the display (use Draw() or DrawAll()), nor do they perform any analysis (see the Process() command).

The commands return a view handle, or a negative error code. Errors include: Bad channel number, illegal number of bins, out of memory, illegal bin size. The new view is made the current view. However, it is created invisibly and must be made visible with WindowVisible(1) before it will appear when drawn.

Most commands accept a channel specifier, written as cSpc. It can be an integer channel number, or -1 for all, -2 for visible, -3 for selected or -6 for unselected channels. It can also be a channel specification string, such as "1..4,6,10" or an integer array where the first element is the channel count, the remaining elements are the channel numbers. Whatever channels are specified, only channels of the correct type for the command are used. If a channel is included twice, the first occurrence is used. It is an error if the resulting list is empty. Result view channels appear in the same order as in the specification. The first channel in cSpc generates result channel 1, the second generates channel 2, and so on.

There is also the Measure...() family of commands that generate XY views or a new channel of time view data holding the results of measurements. The commands are:

MeasureToXY	Create an XY view and associated measurement process
MeasureX	Set the x part of a measurement
MeasureY	Set the y part of a measurement
MeasureToChan()	Create a new Event/Marker/RealMark channel for analysis results
MeasureChan()	Add or modify a Measurement...() in XY or Time view

Choose a command for more information:
SetAverage(), SetEvtCrl(), SetEvtCrlShift(), SetINTH(), SetPhase(), SetPower(), SetPSTH(), SetResult(), SetWaveCrl(), SetWaveCrlDC()

SetAverage()

This command creates a result view to hold the sum or average of sweeps of waveform data and makes it the current view. See the Waveform average interactive commands for details of the analysis. The Process() command does the analysis. Sweeps() reports the number of sweeps accumulated. Omitted optional arguments have the value 0. The function is:

```
Func SetAverage(cSpc, bins% {,offset {,trig% {,flags%{,align%}}});
```

cSpc A channel specifier of waveform or WaveMark channels to average in the current time view, or in the time view associated with the current result view. Invalid channels are ignored. The channel list must hold at least one valid channel. All the channels must have the same sampling rate; any channels that do not match the sample rate of the first channel in the list are ignored. Each valid channel generates one result view output channel, numbered in order from 1.

bins% The number of bins required. There must be at least one bin. The maximum is limited by available memory. The bin width is the waveform sampling interval.

offset This sets the pre-trigger time, in seconds, to show in the result. If omitted, 0 is used.

trig% The channel number to use as a trigger for each sweep. If this is omitted, or set to 0, then each call to Process() takes the start time as the trigger time.

flags% This is the sum of flag values: 1=display the mean data and not the sum, 4=enable error bars, 32=count items per bin. If flags% is omitted, 0 is used.

align% How to align data to the trigger. 0=Next point, 1=nearest point, 2=linear interpolation, 3=cubic spline interpolation. If omitted, 0 is used.

Returns The function returns a view handle for the new view, or a negative error code.

See also:
Channel specifiers, BinError(), ChanData(), DrawMode(), Process(), Sweeps()

SetEvtCrl()

This creates a result view of event correlation histograms and optional raster displays and makes it the current view. The `Process()` command does the analysis. `Sweeps()` reports the number of triggers processed. You can take measurements from an auxiliary channel from each sweep and use this value to sort rasters, display a symbol and/or discard sweeps. See Event correlation in the Analysis menu for details. See `RasterSort()` and `RasterSymbol()` for a description of sort values and symbol times.

```
Func SetEvtCrl(cSpc, bins%, binsz{, offset{, trig%{, flags%{, aCh%{, Mn, Mx}}}}});
```

`cSpc` A channel specifier of event or marker channels in the current time view, or in the time view associated with the current result view. Invalid channels are ignored. The resulting channel list must hold at least one valid channel. Each valid channel generates one result view output channel, numbered in order from 1.

`bins%` The number of bins in the histogram. There must be at least 1 bin.

`binsz` The width of each bin in seconds. This is converted into underlying time units.

`offset` This sets the pre-trigger time, in seconds, to show in the histogram.

`trig%` The channel number to use as a trigger for each sweep. If this is omitted, or set to 0, then each call to `Process()` takes the start time as the trigger time.

`flags%` This is the sum of: 1 to scale the result as spikes per second, 2 to enable raster displays, 8 for backwards `aCh%` event search, 16 to exclude rather than include values that lie in the `Mn` to `Mx` range. If omitted, 0 is used.

`aCh%` Auxiliary measurement channel. The measured value sets raster sort value 1. For waveform or RealWave channels, this is the waveform value at the trigger time. For all other channel types, it is the latency of the first event on the channel before/after the trigger and it also sets the time of symbol 1. If you omit `aCh%`, there is no auxiliary channel.

`Mn,Mx` If present, these arguments set the range of measured values from `aCh%` that control if a sweep is included or excluded.

Returns The function returns a view handle for the new view, or a negative error code.

For an auto-correlation, set `trig%` the same as `chan%`. In this case, we do not count the result of correlating each event with itself. If you must count self-correlations, add `Sweeps()` to the bin holding the time shift of zero.

See also:
Event time cross correlogram, Channel specifiers, `SetEvtCrlShift()`, `DrawMode()`, `Process()`, `RasterSet()`, `RasterSort()`, `RasterSymbol()`, `Sweeps()`

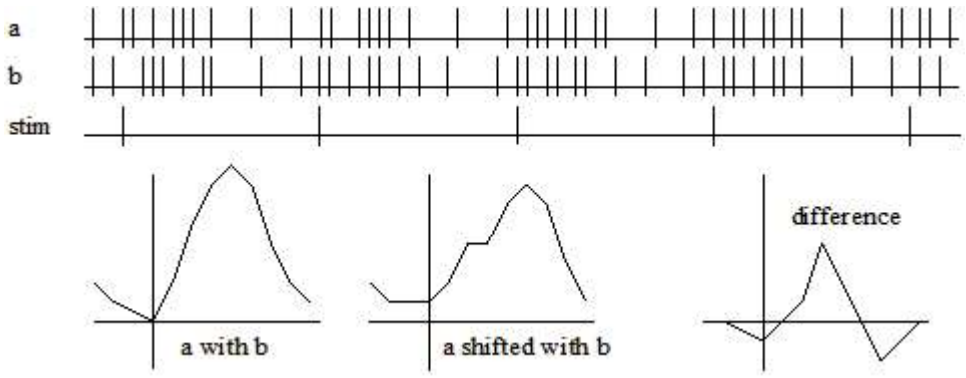
SetEvtCrlShift()

This can be used when the current result view was created by `SetEvtCrl()`. It sets a time shift for the next `Process()` command to produce shuffled correlations by shifting one channel with respect to the other without changing the result view time axis.

```
Proc SetEvtCrlShift(shift);
```

`shift` The time shift, in seconds, to used. This value can be positive or negative. It causes the data on the `trig%` channel to be correlated with data from channel `chan%` at a time `shift` later. The shift is set back to zero after the `Process()` command on this result view.

Consider two channels of events (to be correlated) and a channel marking stimuli:



If the stimulus channel holds events at some constant interval `int`, by shifting all events on one of the two channels by `int`, the only correlation between the two channels left is the correlation due to the stimulus. Thus by forming the difference between the unshifted correlation and the shifted correlation, you can obtain the correlation between a and b with the effect of the stimulus removed.

See also:
`SetEvtCrl(), Process()`

SetFHst()

This function creates a result window to hold a frequency histogram and makes it the current view. The `Process()` command does the analysis. Each interval processed increments the value returned by `Sweeps()`, even intervals that are too short or too long to contribute to the histogram. The frequency is calculated as the inverse of the interval between consecutive events.

```
Func SetFHst(cSpc, bins%, binsz{, minFr});
```

`cSpc` A channel specifier of event or marker channels in the current time view, or in the time view associated with the current result view. Invalid channels are ignored. The resulting channel list must hold at least one valid channel. Each valid channel generates one result view output channel, numbered in order from 1.

`bins%` The number of bins required. There must be at least one bin. The maximum is limited by available memory.

`binsz` The width of each bin in Hz. This must be greater than 0.

`minFr` This sets the start of the first analysis bin, in Hz. Lower frequencies are counted for `Sweeps()`, but not included in the histogram. If omitted, 0 is used.

Returns The function returns a view handle for the new view, or a negative error code.

The histogram runs from `minFr` on the left to `minFr + bins% * binsz` on the right.

See also:
Channel specifiers, `Process(), SetINTH(), Sweeps()`

SetINTH()

This function creates a result window to hold an interval histogram and makes it the current view. The `Process()` command does the analysis. Each interval processed increments the value returned by `Sweeps()`, even intervals that are too short or too long to contribute to the histogram.

```
Func SetINTH(cSpc, bins%, binsz{, minInt});
```

`cSpc` A channel specifier of source event or marker channels in the current time view, or in the time view associated with the current result view. Invalid channels are ignored. The resulting channel list must hold at least one valid channel. Each valid channel generates one result view output channel, numbered in order from 1.

`bins%` The number of bins required. There must be at least one bin. The maximum is limited by available memory.

`binsz` The width of each bin in seconds. This is converted into underlying time units.

`minInt` This sets the start of the first analysis bin, in seconds. Shorter intervals are counted for `Sweeps()`, but not included in the histogram. If omitted, 0 is used.

Returns The function returns a view handle for the new view, or a negative error code.

See also:
Channel specifiers, `Process(), SetFHst(), Sweeps()`

SetPhase()

This function creates a result view to hold phase histograms with optional raster displays and makes it the current view. The `Process()` command does the analysis. `Sweeps()` reports the number of cycles analysed. You can measure a value for each sweep from an auxiliary channel and use this value to sort rasters, display a symbol and discard sweeps. See the description of the Analysis menu Phase histogram for details. See `RasterSort()` and `RasterSymbol()` for a description of sort values and symbol times.

```
Func SetPhase(cSpc, bins%{, minCyc{, maxCyc{, cycle%{, flags%{, aCh%{, Mn, Mx}}}}});
```

`cSpc` A channel specifier of source event or marker channels in the current time view, or in the time view associated with the current result view. Invalid channels are ignored. The resulting channel list must hold at least one valid channel. Each valid channel generates one result view output channel, numbered in order from 1.

`bins%` The number of bins required. There must be at least one bin. The maximum is limited by available memory. The bins are each 360/`bins%` wide.

`minCyc` The minimum cycle time to use in seconds. If two consecutive cycle markers are closer than this time, the cycle is ignored, and is not counted by `Sweeps()`. If this is omitted, a value of 0.0 is used.

`maxCyc` The maximum cycle time to use in seconds. If two consecutive cycle markers are further apart than this, the cycle is ignored, and is not counted by `Sweeps()`. If this is omitted, there is no limit on cycle size.

`cycle%` The channel number to use as the cycle start and end marker. If this is omitted, or set to 0, then each call to `Process()` takes the start time and end time as marking a single cycle.

`flags%` This is the sum of: 2 to enable raster displays, 8 for backwards `aCh%` event search, 16 to exclude rather than include sweeps with measured values that lie in the `Mn` to `Mx` range. If omitted, 0 is used.

`aCh%` Auxiliary measurement channel. The measured value sets raster sort value 1. For waveform or RealWave channels, this is the waveform value at the cycle start. For all other channel types, it is the latency of the first event on the channel before/after the cycle start and it also sets the symbol 1 time. If you omit `aCh%`, there is no auxiliary channel.

`Mn,Mx` If present, these arguments set the range of measured values from `aCh%` that control if a sweep is included or excluded.

Returns The function returns a view handle for the new view, or a negative error code.

See also:
Event phase histogram, Channel specifiers, `DrawMode(), SetEvtCrl(), Process(), RasterSet(), RasterSort(), RasterSymbol(), Sweeps()`

SetPower()

This function creates a result view to hold a power spectrum and makes it the current view. The `Process()` command does the analysis. The function is as follows:

```
Func SetPower(cSpc, fftsz% {,wnd%});
```

cSpc A channel specifier of source waveform channels in the current time view, or in the time view associated with the current result view. Invalid channels are removed from the list. The resulting channel list must hold at least one valid channel. All the channels must have the same sampling rate; any channels that do not match the sample rate of the first channel in the list are ignored. Each valid channel generates one result view output channel, numbered in order from 1.

fftsz% The size of the transform used in the FFT. This must be a power of 2 in the range 16 to 262144. The result view has half this number of bins. The width of each bin is the sampling rate of the channel divided by fftsz%. Each block of fftsz% data points processed increments the value for Sweeps().

wnd% The window to use. 0 = none, 1 = Hanning, 2 = Hamming. Values 3 to 9 set Kaiser windows with -30 dB to -90 dB side band ripple in steps of 10 dB. If this is omitted a Hanning window is applied.

Returns The function returns a view handle for the new view, or a negative error code.

See also:
Channel specifiers, ArrFFT(), Process(), Sweeps()

SetPSTH()

This creates a result view to hold a peri-stimulus time histogram with an optional raster display and makes it the current view. TheProcess() command does the analysis. Sweeps() returns the number of triggers processed. You can measure a value for each sweep from an auxiliary channel and use this to sort rasters, display a symbol and discard sweeps. See the description of the PSTH in the Analysis menu for details. See RasterSort() and RasterSymbol() for a description of sort values and symbol times.

```
Func SetPSTH(cSpc, bins%, binsz {,offset {,trig% {,flags%{,aCh%
{,Mn,Mx}}}}});
```

cSpc A channel specifier of source event or marker channels in the current time view, or in the time view associated with the current result view. Invalid channels are ignored. The resulting channel list must hold at least one valid channel. Each valid channel generates one result view output channel, numbered in order from 1.

bins% The number of bins required. There must be at least one bin.

binsz The width of each bin in seconds. This is converted into underlying time units.

offset This sets the pre-trigger time, in seconds. If omitted, 0.0 is used.

trig% The channel number to use as a trigger for each sweep. If this is omitted, or set to 0, then each call to Process() takes the start time as the trigger time.

flags% This is the sum of: 1 to scale the result as spikes per second, 2 to enable raster displays, 8 for backwards aCh% event search, 16 to exclude rather than include values that lie in the Mn to Mx range. If omitted, 0 is used.

aCh% Auxiliary measurement channel. The measured value sets raster sort value 1. For waveform or RealWave channels, this is the waveform value at the trigger time. For all other channel types, the value is the latency of the first event on the channel before/after the trigger and it also sets the symbol 1 time. If you omit aCh%, there is no auxiliary channel.

Mn,Mx If present, these arguments set the range of measured values from aCh% that control if a sweep is included or excluded.

Returns The function returns a view handle for the new view, or a negative error code.

See also:
Peri-Stimulus time histogram, Channel specifiers, DrawMode(), SetEvtCrl(), Process(), RasterAux(), Sweeps()

SetResult()

This function creates a result view of user-defined type, attached to no time view and with no implied `Process()`. It becomes the current view. `Sweeps()` will return 0 unless you set the sweep count.

```
Func SetResult({chans%,} bins%, binsz, offset, title$, xU${, yU$ {,xT$
{,yT$ {,flags% {,tick}}}}});
```

`chans%` This sets number of channels in the new result view. Omit for 1 channel. See the `View()` documentation for access to multiple channel result view data. The maximum channels in a result view is currently limited to 2000. If you use a large number of channels, each channel generates multiple underlying windows, and there is a limitation on the number of such objects that can be used at one time in an application.

`bins%` The number of bins in the view.

`binsz` The width of each bin. Bins should have a positive non-zero width.

`offset` The x axis value at the start of the first bin.

`title$` The window title.

`xU$` The x axis units.

`yU$` Optional, y axis units, blank if omitted.

`xT$` Optional, x axis title (otherwise blank).

`yT$` Optional, y axis title (otherwise blank).

`flags%` Add 1 for mean (not sum) mode for errors, 2 to enable raster data, 4 to enable error bars, 32 for individual counts for each bin. The default value is 0.

`tick` This is required if you enable raster data. Result views store raster data as 32-bit integers. `tick` is the time resolution for this data. When working with a time view, set `tick` to `BinSize()`, the time resolution of the time view. The maximum time of a raster event is 2147483647`*tick` seconds.

Returns The function returns a view handle for the new view, or a negative error code.

See also:
`BinError(), DrawMode(), RasterGet(), RasterSet(), Sweeps(), View()`

SetWaveCrl()

This creates a result view to hold the waveform correlation between waveform channels sampled at the same rate and makes it the current view. The `Process()` command does the analysis. `Sweeps()` returns the number of data points used on the reference channel.

```
Func SetWaveCrl(cSpc, ref%, bins%{, offset});
```

`cSpc` A channel specifier of source waveform channels in the current time view, or in the time view associated with the current result view. Invalid channels are removed from the list. There must be at least one valid channel. Channels that do not match the sample rate of the first channel in the list are ignored. Each valid channel generates one result view output channel, numbered in order from 1.

`ref%` The reference waveform channel that is correlated with all the source waveform channels.

`bins%` The number of bins in the correlation. The bin width is the channel sampling interval. This calculation is slow compared to the other `SetXXXX` commands. The time taken is proportional to the length of the area processed times `bins%`.

`offset` The time to show before the zero time shift, in seconds. If omitted, 0 is used.

Returns The function returns a view handle for the new view, or a negative error code.

See also:
Channel specifiers, `SetWaveCrlDC(), Process(), Sweeps()`

SetWaveCrlDC()

To use this function the current view must be a waveform correlation result view. It sets whether the DC levels of the two signals is removed before the correlation or not.

```
Func SetWaveCrlDC({useDC%})
```

`useDC%` If present, a zero value removes the DC, a non-zero value (default) includes it.

Returns The `useDC%` value before the call. It returns a negative error if this view is a result view but not a waveform correlation.

If you change the setting the view is drawn at the next opportunity.

See also:
`SetWaveCrl(), Process()`

Sin()

This calculates the sine of an angle in radians, or converts an array of angles in radians into sines.

```
Func Sin(x|x[]{[]...});
```

`x` The angle, expressed in radians, or a real array of angles. The best accuracy of the result is obtained when the angle is in the range -2π to 2π.

Returns When the argument is an array, the function replaces the array with the sines and returns 0 or a negative error code. When the argument is not an array the function returns the sine of the angle.

See also:
`Abs(), ATan(), Cos(), Cosh(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sqrt(), Sinh(), Tan(), Tanh()`

Sinh()

This calculates the hyperbolic sine of a value or of an array of values.

```
Func Sinh(x|x[]{[]...});
```

`x` The value, or an array of real values.

Returns When the argument is an array, the function replaces the array elements with their hyperbolic sines and returns 0. When the argument is not an array the function returns the hyperbolic sine of the argument.

See also:
`Abs(), ATan(), Cos(), Cosh(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sqrt(), Sinh(), Tan(), Tanh()`

SMControl()

This function sets and reads the toolbar and edit field states of the current spike monitor window. You can open this window and get the view handle with the **`SMOpen()`** command. See the documentation of the spike monitor window for a full description.

```
Func SMControl(item%{, new});
```

`item%` This identifies the item:

0	Draw mode	0=3D, 1=2D, 2=2D separated
1	Fade to background	0=no fade, 1=fade
2	Colour last spike only	0=colour all, 1=colour last, the rest are grey
3	Use thick lines	0=use thin lines, 1=use thick lines (slow)

4	Timed (smooth) update	0=update on new data, 1=update on timer
5	At end mode	0=use cursor 0, 1=at end of file
6	Lock y axes	0=track data size, 1=fix at current size
7	Show duplicate channels	0=Show originals only, 1=show duplicates
8	Maximum spikes	In the range 1 to 40
9	Time range	In the range 0.003 to `MaxTime()`
10	Display rectangles	0=no, 1=show front and back rectangles
11	Front rectangle x offset	In the range 0 to 0.9
12	Front rectangle scale	In the range 0.1 to 1.0
13	Back rectangle x offset	In the range 0 to 0.9
14	Back rectangle scale	In the range 0.1 to 1.0

`new` If present, this sets the state of the item. See the table for acceptable values.

Returns The item state at the time of the call or -1 if the item does not exist.

See also:
Spike Monitor window, `SSChan(), SSClassify(), SSOpen(), SSParam(), SSRun()`

SMOpen()

This function gets the spike monitor window handle or opens it. The current view must be a time view or a spike monitor window. Close the spike monitor window with **`FileClose()`**. Use the **`SMControl()`** command to control the window. You can use the **`Window()`** and **`WindowVisible()`** commands to size, show and hide the window.

```
Func SMOpen({type%{, mode%});
```

`type%` This argument is assumed to be zero if omitted. Allowed values are:

- 0 Returns the view handle of the spike monitor window associated with the current view, or 0 if there is no open window.
- 1 Open the spike monitor window. Returns the handle or 0 if we failed.

`mode%` If this is zero or omitted, the dialog is created invisibly. Set 1 to make it visible. This is ignored unless `type%` is 1.

Returns The return value is the view handle of the spike monitor window or 0.

See also:
Spike Monitor window, `FileClose(), SMControl(), ViewLink(), Window(), WindowVisible()`

Sound()

This command has two variants. The first plays a `.WAV` file or system sound through your sound card if your system has multimedia support. The second plays a tone of set pitch and duration. If you want to play data held in a channel of a data file, see the `PlayOffline()` script command. If you want to convert text to speech, see the `Speak()` script command.

Multimedia (sound card) output

This command variant uses a simple system feature that will play a nominated file of less than some maximum size (256 kB the last time I looked into this, but it could be less on some systems) through the multimedia sound system. As we have made no provision in Spike2 for setting this up, you must have arranged your system so that the default sound output is routed to wherever you require it.

```
Func Sound(name${, flags%);
```

`name$` The name of a `.wav` file or of a system sound. You can supply the full path to the file or just a file name and the system will search for the file in the current directory, the Windows directory, the Windows system directory, directories listed in the `PATH` environmental variable and the list of directories mapped in a network. If no file extension is given, `.wav` is assumed. The file must be short enough to fit in available physical memory.

A blank name ("") halts sound output. If `name$` is any of the following (case is important), a standard system sound plays:

`"S*"`	Asterisk	`"SS"`	Start	`"SE"`	Exit	`"SH"`	Hand
`"SW"`	Welcome	`"S?"`	Query	`"SD"`	Default	`"S!"`	Exclamation

What you get for these sounds depends on the operating system and sound scheme. For example, in Windows 10, the default scheme renders all of these as the same sound and only Asterisk (the default) and Exclamation are settable.

`flags%` This optional argument controls how the data is played. It is the sum of:

0x0001	1	Play asynchronously (start output and return). Without this flag, the `Sound()` command waits until replay ends.
0x0002	2	Silence when sound not found. Normally `Sound()` plays the system default sound if the nominated sound cannot be found.
0x0008	8	Loop sound until stopped by another `Sound()` command; use `""` for `name$` to stop output.. You must also supply the asynchronous flag if you use loop mode.
0x0010	16	Don't stop a playing sound. Normally, unless the "No wait" flag is set, each command cancels any playing sound.
0x2000	8192	No wait if sound is already playing. `Sound("",0x2010)` returns 1 if a previous asynchronous sound is finished and 0 if not.

If you don't supply this argument, the flag value is set to 0x2000.

Returns The multimedia output returns non-zero if the function succeeded and zero if it failed.

Motherboard speaker output (simulated by sound cards on modern systems)

Many years ago, PC motherboards had a header that could be connected to a small speaker. This was originally used to signal system start up failures and was driven by a timer chip that produced a square wave output at a settable frequency. Once sound cards became standard, systems stopped supporting this feature and the underlying software support for it (Beep) was dropped from 64-bit XP and Vista. However, from Windows 7 onwards, Beep was rewritten to use the default sound device, and this function works on systems with a sound card.

```
Func Sound(freq%, dur{, midi%});
```

`freq%` If `midi%` is 0 or omitted this holds the sound frequency in Hz. If `midi%` is non-zero this is a MIDI value in the range 1-127. A MIDI value of 60 is middle C, 61 is C# and so on. Add or subtract 12 to change the note by one octave.

`dur` The sound duration, in seconds. The script stops during output.

`midi%` If present and non-zero, `freq%` is interpreted as a MIDI value.

returns 0 or a negative error code.

See also:

`PlayOffline()`, `Speak()`

Speak()

If your system supports text to speech, this command allows you to convert a text string into speech. We do not provide facilities to set up voices or to route the sound output; you must do this from the Speech applet in the control panel. There are two command variants. The first outputs text as speech; the second is for control of the output and provides status information.

```
Func Speak(text${, opt%});
Func Speak({what%{, value}});
```

`text$` A string holding the text to output, for example `"Sampling has started"`.

`opt%` This optional argument (default value 1) controls the text conversion and output method. It is the sum of the following flags:

1 Speak asynchronously. Without this flag the command waits until speech output is over before returning.

2 Cancel any pending speech output.

4 Speak punctuation marks in the text.

8 Process embedded SAPI XML. For example:
`Speak("Emphasis on <EMPH>this</EMPH> word",8);`

16 Reset to the standard voice settings before speaking.

`what%` An optional variable, taken as 0 if it is omitted:

0 Returns 1 if speech is playing and 0 if it is not.

1 Wait for up to `value` seconds (default value 3.0) for output to end. The return value is 0 if playing is finished, 1 if it continues after `value` seconds.

2 Returns the current speech speed in the range -10 to 10; 0 is the standard speed. If `value` is present, it sets the new speed.

3 Returns the current speech volume in the range 0 to 100, 100 is the standard volume. If `value` is present, it sets the new volume.

`value` An optional argument used when `what%` is greater than 0.

Returns If there is no speech support available, or a system error occurs, the command returns -1. Otherwise the first command variant returns 0 if all is well and the second variant returns the values listed for `what%`.

To use TTS (text to speech), you need a suitable sound card and the Microsoft SAPI software support (included in all currently supported versions of Windows). Note that this command uses the same mechanisms as the Info window speech output (`InfoSpeak()` command).

See also:
`Sound()`, Embedded SAPI XML, `InfoSpeak()`

Embedded SAPI XML

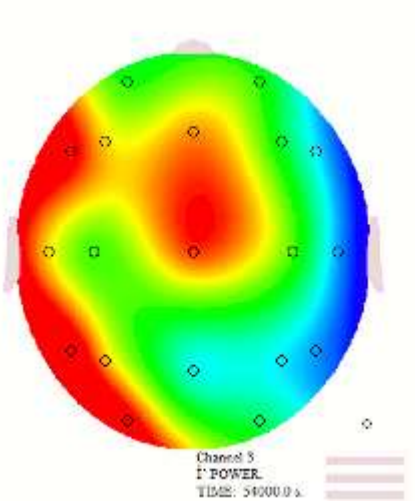
It is possible to embed instructions to the "voice" within the text to change how the text is rendered. Here are some simple examples of embedded XML:

```
Speak("<EMPH>This</EMPH> is an emphasis", 8);
Speak("<RATE SPEED='-5'>This speech is at a relative speed of -5.</RATE>", 8);
Speak("<PITCH MIDDLE='5'>This is high pitched speech.</PITCH>", 8);
Speak("<VOLUME LEVEL='50'>This is quiet speech.</VOLUME>", 8);
Speak("<SPELL>Spell this out.</SPELL>", 8);
Speak("Five hundred milliseconds of silence <silence msec='500'/> just occurred.",
```

There are other more complicated commands you can give. See this web site for more information on this advanced topic.

Spline2D

Given a set of (x_i, y_i) or (x_i, y_i, z_i) positions and the values v_i at each position, this command allows you to generate interpolated values at arbitrary (but local to the existing points) positions by interpolation. The interpolation is based on the distance of each interpolated point from the known points, so it can work in any number of dimensions, but we have implemented it in 2D (x,y) and 3D (x,y,z). We also provide functions to help in generating bitmaps of the result.



For example, in EEG work, you might have a set of measurements from electrodes attached to a head and you want to map the power in a particular frequency band (which is known at each electrode) across the entire head. In the example image the small circles are electrode positions at which the power in a particular frequency band is known and the colours represent power density with red as high and blue as low.

The `Spline2D()` function is useful for a reasonable number of (x, y) positions (say a maximum of a hundred or so). The example picture uses 19.

The function uses two basic methods to map the data: Inverse distance weighting and Radial basis functions. Both methods use the distances r_i of a general position (x, y) or (x, y, z) from each of the i positions, which means that x and y (and z) must be measured in the same units. Put another way, this is not a suitable method to use if your y position is distance and your x position is time.

If you have data points on a rectangular grid (such as might occur when looking at the time course of the signals from a linear array), cubic splining treating the x and y axes as independent may be more appropriate.

The command has several variants, some of which are used to set up the problem, some to provide the data and some to obtain the interpolations.

`Func Spline2D(const p[][]);`	Set data point positions
`Func Spline2D(const v[]{, mode%{, flags%{, v1}}});`	Set values and interpolation mode
`Func Spline2D(x, y{, z});`	Find interpolation at a position
`Func Spline2D(out[], const pos[][]);`	Interpolate a list of positions
`Func Spline2D(xy[][], xLo, yLo, xHi, yHi);`	Interpolate into rectangular area

As this command will often be used to create bitmaps, we provide support for efficient mapping of levels to colours. You must use one of the calls to set a colour scale before the calls that generate an image.

`Func Spline2D(const map%[], mLo, mHi);` `Func Spline2D(map$, mLo, mHi, index%);`	Set colour scale to use
`Func Spline2D(xy%[][], xLo, yLo, xHi, yHi);`	Generate image
`Func Spline2D(xy%[][], const pos[][][]);`	Generate image from positions

Set (x, y) or (x,y,z) data point positions

This command variant must be called first and resets any remembered information including resetting the interpolation both `mode%` and `flags%` to 0. It sets the (x, y) positions for use by the other command variants.

```
Func Spline2D(const p[][]);
```

`p[][]` A matrix of x,y or x,y,z positions. That is the matrix is either `p[2][n]` for 2D interpolation or `p[3][n]` for 3D interpolation. `p[0][]` is a vector of x values, `p[1][]` is a vector of y values, and if it exists, `p[3][]` is a vector of z values.

Returns It returns the number of positions that will be used or -1 if points are not distinct (different).

Set the values and the interpolation method

This command variant is called after setting the positions. It sets the values at each of the positions and optionally sets the interpolation method and any required settings. It sets up the command so that it can compute the interpolations efficiently. This is separate from setting the positions as it is very common for the positions to be fixed, but the values to vary with time. If the positions do not change we can reduce the calculations required for each new set of value, which saves time.

```
Func Spline2D(const v[]{, mode%{, flags%{, v1}}});
```

`v[]` A vector of values. This vector must be at least as long as the number of positions. If there are n positions, the first n elements of `v[]` are used.

`mode%` This determines the method used for interpolation. If omitted, the last set `mode%` is used. The reset value is 0. There is more information about the methods, below. The values currently implemented are:

- 0 Inverse distance weighting. `v1` sets the inverse power to use. Positive values greater than 0 and less than 20 or so can be used. Large values may cause arithmetic underflow or overflow.
- 1 Polyharmonic spline using r.
- 2 Polyharmonic spline using $r^2 ln(r)$. This is the 'thin plate spline'.
- 3 Polyharmonic spline using r^3.
- 4 Polyharmonic spline using $r^4 ln(r)$. The distance r is scaled by dividing by `v1`, which is taken as 1 if omitted.
- 5 Radial basis function using $exp(-r^2/v1^2)$. The distance r is scaled by dividing by `v1`, which is taken as 1 if omitted.

`flags%` Flag settings control aspects of the interpolation (there is only one flag at present). If omitted, the flags take the last set value. The reset value is 0. The value is the sum of:

- 1 Subtract the mean of the `z[]` values from the data before use, then add the mean back to the interpolated values. This affects how the results behave outside the locality of the supplied data. If the spline method tends to zero at large distances, this causes it to tend to the mean value.

`v1` Optional parameter. If omitted it takes the last set value. The reset value is 1. Its use depends on `mode%`. Unless explicitly mentioned in the `mode%` description, it should be left at the value 1.

Returns -1 if an error, otherwise the `mode%` value that will be used.

Interpolate single value

This command variant returns the interpolated value at a designated point. It is an error to call this method without having previously set the positions and set the values and interpolation method.

```
Func Spline2D(x, y{, z});
```

`x,y,z` The position to interpolate the result. The z value should not be present for a 2D interpolation and is required for a 3D interpolation. We do not restrict the range of positions, but you should be aware that using a position well away from the locality of the positions that define the spline may not generate useful results.

Returns The interpolated value.

Interpolate a list of values

This command variant fills a vector of values with interpolated values based on a list of positions passed in. This works with 2D or 3D positions. You could use this to interpolate values over a 3D surface mapped onto a plane. It is an error to call this without having set the positions and the interpolation values and method.

```
Func Spline2D(out[], const pos[][]);
```

`out[]` A vector that is filled with interpolated values for each position in `pos`. The number of values returned is the shorter of this vector and the second dimension of `pos`. That is, **`min(len(out, len(pos[0][]))`**.

pos A matrix that is either 2 x n or 3 x n, for 2D or 3D interpolation. pos[0][] is a vector of x values, pos[1][] is a vector of y values and for 3D interpolation, pos[2][] is a vector of z values.

Returns The number of values written to out.

Interpolate to rectangular grid

The next variant is used in the 2D case to map a rectangular region of the (*x*, *y*) plane into a matrix such that each element of the output represents the interpolated value at that grid position. You must have called the routines to set the positions, values and interpolation method. This is equivalent to using the interpolate single value code over a rectangular grid. Added at Spike2 version 9.06.

```
Func Spline2D(xy[][], xLo, yLo, xHi, yHi);
```

xy% This is a matrix xy[nx%][ny%] that is filled in by interpolating values, where nx% is the number of x pixels from left to right and ny% is the number of y pixels.

xLo The x position used to calculate the xy%[0][] elements of the matrix.

yLo The y position used to calculate the xy%[][0] elements of the matrix.

xHi The x position of the end of the range spanned by the matrix. The *x* increment per bin in the *x* direction is (xHi - xLo)/nx%.

yHi The y position of the end of the range spanned by the matrix. The y increment per bin in the *y* direction is (yHi - yLo)/ny%.

Returns 0.

Set colour mapping to create rectangular bitmap

These command variants fill a rectangular integer matrix with values that map from the interpolation. The expectation is that the mapping array will be filled with colour values and that the xy%[][] array will be written as a bitmap.

The first two variants determine how interpolated values map to colours (or colour indices in a palette). You must call one of them before the calls that generate a rectangular bitmap. An alternative that avoids knowledge of bitmap formats is to generate an interpolated read array of RBG colours, then use ArrMapImage() to create the result.

```
Func Spline2D(const map%[], mLo, mHi);
Func Spline2D(map$, mLo, nHi{, nMap%));
```

map%[] An integer vector of values that are to be used in the bitmap to represent interpolated values in a range determined by mLo and mHi. The formula used to determine the element of map% to use for an interpolated value *v* is:

```
mIndex% := (v - mLo) * (Len(map%)-1)/(mHi - mLo);
```

Values outside the range mLo to mHi are mapped to the nearer end of the map. You can use this mechanism to implement a variety of bitmap formats. For example, a simple approach is to use the 32-bit per pixel format with bits 0-7 for blue, 8-15 for green and bits 16-23 for red (with the option of using bits 24-31 for an alpha channel). Alternatively, if you were using a palette format, the map% array might only hold the integer values 0-255 for an 8-bit palette.

map$ Either empty, or the name of a colour scale. If empty, or if the name is not found, the colour scale identified by mapInd% (the index of the scale) is used. This creates an internal equivalent of the map%[256] array; each array element uses bits 0-7 for blue, 8-15 for green and bits 16-23 for red. This matches the expected internal Windows bitmap format.

nMap% This is used as the item number of the colour scale if map$ does not identify a colour map. Valid item numbers are currently in the range -2 to 20 (but not all numbers are valid). If this does not identify a map, a default monochrome map is used.

mLo The interpolated value to map to the first element of the colour map. It is a fatal error for mLo to be the same as mHi.

mHi The interpolated value to map to the last element of the colour map.

Returns The variant that looks up a colour map returns 1 if a nominated map was found. Otherwise 0 is returned.

Map into rectangular bitmap

The next variant is used in the 2D case to map a rectangular region of the (*x*, *y*) plane into a matrix such that each element of the output represents the colour of a pixel in a bitmap. You must have already set up the mapping of interpolated values into integers (set a map and `mLo` and `mHi`) and have called the routines to set up the interpolation.

```
Func Spline2D(xy%[][], xLo, yLo, xHi, yHi);
```

`xy%` This is a matrix `xy%[nx%][ny%]` that is filled in by interpolating values, then mapping these values to an integer, where `nx%` is the number of x pixels from left to right and `ny%` is the number of y pixels. We only change bits 31-0 of the 64 bit values held in `xy%`. Further, if any of the bits 63-32 are set, we DO NOT CHANGE the value (or calculate it). This allows you to mask out areas of the output and to fill these areas (if you wish) with graphical elements that never change. If you do not want to be concerned with this, just make sure that the upper 32-bits of each element of `xy%` are zero.

`xLo` The x position used to calculate the `xy%[0][]` elements of the matrix.

`yLo` The y position used to calculate the `xy%[][0]` elements of the matrix.

`xHi` The x position of the end of the range spanned by the matrix. The *x* increment per bin in the *x* direction is `(xHi - xLo)/nx%`.

`yHi` The y position of the end of the range spanned by the matrix. The y increment per bin in the *y* direction is `(yHi - yLo)/ny%`.

Returns The number of elements that were outside the range `mLo` to `mHi`.

Map 3D or 2D to a rectangular bitmap

The next variant maps a 3D (or 2D) surface onto a 2D bitmap (for example to map interpolations of values on the surface of a human head onto a flat plane). You must have already set up the mapping of interpolated values into integers (`map%`, `mLo` and `mHi`) and have called the routines to set up the interpolation.

```
Func Spline2D(xy%[][], const pos[][][]);
```

`xy%` This is a matrix `xy%[nx%][ny%]` that is filled in by interpolating values, then mapping these values to an integer, where `nx%` is the number of x pixels from left to right and `ny%` is the number of y pixels. The same comments about masking out values applies as for the rectangular map, above.

`pos` This is either `pos[2][nx%][ny%]` for a 2D mapping or `pos[3][nx%][ny%]` for a 3D mapping. The first dimension holds the x, y (and z) co-ordinates in the `pos[0][][]`, `pos[1][][]` (and `pos[2][][]`) elements. The numbers of elements in each dimension must be large enough to match the `xy%` matrix and to match the 2D or 3D mapping.

Returns The number of elements that were outside the range `mLo` to `mHi`.

Inverse distance weighting

This is a relatively simple-minded form of interpolation, which weights the contribution of each known point to the result at any arbitrary point by some negative power of the distance to the known point. The basic idea is that if d_i is the distance to the i^{th} known value v_i, the interpolated value is:

$$\sum v_i/d_i^p \;/\; \sum 1/d_i^p$$

The larger the value of *p*, the more the effect of local points dominates the result (*p* need not be integral). High values (16, for example), tend to generate hard edges in the interpolation along the lines joining the known points. You can see examples of this here.

This formula appears to break down as you approach any of the known points due to division by zero, but you can see that as you get very close to any point *n* you can ignore the contribution of all other points (as $d_n << d_i$ for all other points), so the formula becomes $v_n/d_n^p \;/\; 1/d_n^p$ which simplifies to just v_n.

Radial basis functions

The idea here is that given our fixed points, to place values w_i (to be calculated) at each of the fixed points such that the interpolated value at a point that is a distance r_i from each of the fixed points is given by:

$\sum w_i f(r_i)$

The w_i are calculated such that the interpolation passes through each of the fixed points.

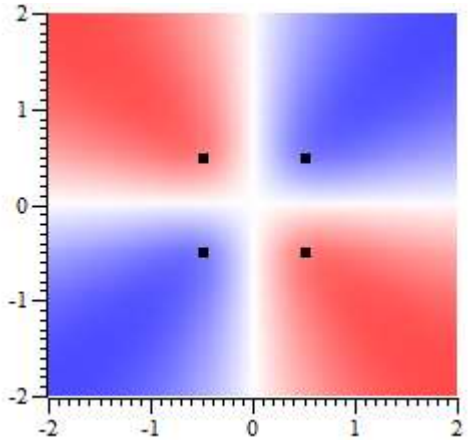
The $f(r)$ are known as radial basis functions, and many are possible. We implement a few that are likely to be useful. The first 4 in the list are also known as Polyharmonic Splines (r^n for n odd, $r^n ln(r)$ for n even).

r	This is a linear interpolation between the calculated points.
$r^2 ln(r)$	This is a Polyharmonic spline that is also known as the Thin plate spline. This has several nice properties that make it a good choice for smooth splining.
r^3	Polyharmonic spline.
$r^4 ln(r)$	Polyharmonic spline.
$exp(-r^2/\sigma^2)$	Gaussian. Setting a sensible σ value is important. Too small a value results in spikes at each fixed data point.

The generation of the w_i values involves inverting a matrix (which is an $O(n^3)$ process, with n the number of fixed points, another reason to keep the number of fixed points within reason). If your input data points all have the same sign and are all a long way from zero, you may be able to improve the results by setting the `flags%` value to remove the mean value from the data before interpolation, then add it back after.

Spline2D example

The following example code shows how you can generate a bitmap from scattered data points. In this case, we have 4 data points (the black squares) with known values of 1, -1, 1 and -1 (clockwise from the top left). The data is interpolated using the "thin plate spline" mechanism using red to represent positive values and blue to represent negative values.



The image to the right shows the result. To interpret this, the darker the red, the more the interpolating surface is raised. The darker the blue, the more the interpolating surface is lowered. The white cross represents the places where the interpolating surface passes through 0.

The code required to set up the problem is:

```
const p[][] := {{-0.5, 0.5}, {0.5, 0.5}, {0.5, -0.5}, {-0.5, -0.5}};
Spline2D(p);                'set the initial point positions (square)
const v[] := {1, -1, 1, -1};'the data values for each of the positions
Spline2D(v, 2);             'set point values in "thin plate spline mode"
```

Setting a colour map

We could now use the `Spline2D(x, y)` calls to obtain the splined values as any desired (x, y) position. From [9.06] we can use a command variant to fill a rectangular grid with the interpolated values. However, we want to generate a bitmap, so we need a map from data values to colours. To do this, we generate a list of integer values, each value representing a colour in RGB format and we set the colours to span a range of values. Colours are coded from 0 (off) to 255 (`0xff`, full on) as:

```
blue% + green%*0x100 + red%*0x10000
```

Hexadecimal notation is convenient as it uses 2 digits for each colour. Black is `0x000000`, bright blue is `0x0000ff`. Bright green is `0x00ff00` and bright red is `0xff0000`. White is `0xffffff`. Setting the same

proportion of red, green and blue generates a grey scale. The following code sets up a map from bright blue through white to bright red. We set this to map the range -2 to +2.

```
var col%[511], i%;
for i% := 0 to 255 do col%[i%] := 0xff + i%*0x10100 next;
for i% := 256 to 510 do col%[i%] := 0xff0000 + (510-i%)*0x101 next;
Spline2D(col%, -2, 2);      'colour map blue - white - red for -2 to +2
```

From [9.06], there is a much simpler way to set a colour map, which is to choose one of the system colour scales that are used for Sonogram displays and clustering. For example to set the Rainbow preset colour map:

```
Spline2D("Rainbow", -2, 2); 'Set the Rainbow pre-defined colour map
```

You can generate additional maps with the `ColourSet()` script command.

Generate an image file

The next task is to map this into a bitmap and then save it in a file. The data points form a square with side 1 centred at 0,0 and we choose to display the mapping from (-2,2) to (2,2) using a 200 x 200 pixel bitmap. We save the bitmap to `test.bmp` in the folder `My Documents\Spike11`. : We start by generating the bitmap pixel data in an integer array. We know that the `bmp%` matrix will be initialised to 0, so we do not need to worry about having some of the data masked out due to the upper bits of the `bmp%[][]` image being non-zero (see the *Using a mask* section, below).

```
const xLo := -2, xHi := 2, yLo := -2, yHi := 2;
var bmp%[200][200];          'bitmap space 200 x 200 pixels
Spline2D(bmp%, xLo, yLo, xHi, yHi); 'make a bitmap
var path$ := FilePath$(-4) + "test.bmp"; 'Where to save a bitmap
```

We now need to convert this into an image file. From [9.06] onwards there are two ways to do this: backwards compatible, and via the clipboard.

Backwards compatible method

This method generates a Windows `.bmp` file the hard way, using the binary file writing features built into Spike2. You can find the code for `WriteBmp()` in the `BWriteSize()` script command example (so we don't repeat it here).

```
WriteBmp(path$, bmp%, 0);    'See BWriteSize example for code
```

Using the clipboard

This takes advantage of two new features in [9.06]:

```
EditCopy(bmp%);               ' Put the image on the clipboard as a bitmap
EditImageSave(path$);         ' Write the image to a file as a bitmap
```

Display in an XY view

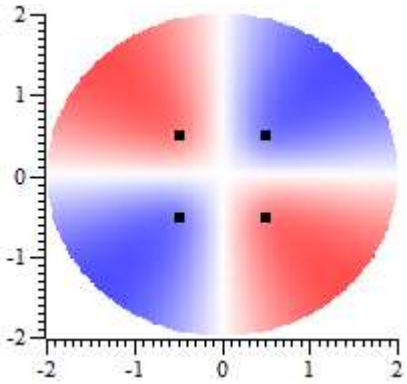
Finally, we create an XY view and set the newly created bitmap as the channel image. We then scale the image so that it spans the same x and y axis range as the bitmap and add data points for the positions where the data values are known.

```
var xy% := FileNew(12, 1);    ' New XY view
XRange(xLo, xHi);             ' Set axis range to span
YRange(1, yLo, yHi);          ' the data
ChanImage(1, path$);          ' Link bitmap to the channel and set...
ChanImage(1, 2, 1.00, xLo, yLo, xHi, yHi); '...to span the data range
XYAddData(1, p[0][], p[1][]); 'Mark data positions
```

The mapping operation is quite fast. On my computer it took around 8 milliseconds, so it is entirely possible to use this technique to generate an animated display. The time is proportional to the number of pixels and to the number of known data points. If you had 20 known points (a more reasonable number), it would likely take 5 times longer than this simple example with 4 known points.

Using a mask

If you want to use a mask to remove some of the result the following shows you how to do it. Insert the following code between the declaration of `var bmp%` and the call to `Spline2D()`:




```
for x% := 0 to 199 do
    for y% := 0 to 199 do
        xd% := x% - 100;     ' distance from centre in x
        yd% := y% - 100;     ' distance from centre in y
        if (xd%*xd% + yd%*yd% > 10000) then
            bmp%[x%][y%] := 0x100ffffff; 'Set white
        endif
    next
next
```

This scans every pixel of the bitmap and sets the data value of any pixel that is more that 100 pixels from the centre (deemed to be at 100,100) to be white (using the lower 32 bits, and sets bit 32 to 1 so that the pixel is excluding from the Spline calculation.

Alternative method generating a grid of data values

If you do not need to make use of masks and do not want to know anything about pixels, you can use the following method from [9.06] onwards. This has the advantage of generating a grid of interpolated values, which can be useful for further numeric analysis.

```
const p[][] := {{-0.5, 0.5}, {0.5, 0.5}, {0.5, -0.5}, {-0.5, -0.5}};
Spline2D(p);                    'set the initial point positions (square)
const v[] := {1, -1, 1, -1};    'the data values for each of the positions
Spline2D(v, 2);                 'set point values in "thin plate spline mode"

var vals[200][200];             'Grid of result values
const xLo := -2, xHi := 2, yLo := -2, yHi := 2;
Spline2D(vals, xLo, yLo, xHi, yHi); 'Generate the value at each grid point
var path$ := FilePath$(-4) + "test.bmp"; 'Where to save a bitmap
ArrMapImage(vals, path$, 0, 1, -2, 2); 'Generate the bitmap

var xy% := FileNew(12, 1);      ' New XY view to display the result
XRange(xLo, xHi);               ' Set x axis range to span the data
YRange(1, yLo, yHi);            ' Set y axis range to span the data
ChanImage(1, path$);            ' Link bitmap to the channel and set...
ChanImage(1, 2, 1.00, xLo, yLo, xHi, yHi); '...to span the data range
XYAddData(1, p[0][], p[1][]); 'Mark data positions
```

The start and end of this code is the same, but the central portion is somewhat simpler. We generate the result as a grid of real, interpolated values. This is converted to a bitmap by the `ArrMapImage()` command.

Sqrt()

Forms the square root of a real number or an array of real numbers. Negative numbers halt the script with an error when `x` is not an array. With an array, negative numbers are set to 0 and an error is returned.

```
Func Sqrt(x|x[]{[]...});
```

`x` A real number or a real array to replace with an array of square roots.

Returns With an array, this returns 0 if all was well, or a negative error code. With an expression, it returns the square root of the expression.

See also:

`Abs()`, `ATan()`, `Cos()`, `Exp()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Tan()`, `Trunc()`

SS...()

This family of commands gives you script control over the Spike Shape dialog. You open the dialog, or get the handle to an already open dialog with the `SSOpen()` command. You get and set the current channel with `SSChan()` and step or run with the `SSRun()` command. Use `SSParam()` to access the template parameters dialog. The `SSTemp...()` command family gives you access to the templates. The `SSButton()` command lets you set and get button and check box states in the dialog that control how it operates. The `HCursor()` command controls the horizontal cursors. The `SSCol...()` commands give you script control over collision analysis. Use `FileClose()` to shut the dialog.

The following commands are also implemented for Spike shape views:

`ViewLink()`	Find the parent time view of the spike shape dialog.
`Optimise()`	Use with no arguments to optimise the display, equivalent to the Automatic Levels button.
`YRange()`	Sets the displayed y axis range or set the display to full range.
`YHigh()`, `YLow()`	Get the y axis limits.
`XRange()`	Sets the time view range to process, equivalent to Time Range button
`XLow()`, `XHigh()`	Gets the start and end of the time range to process.
`Cursor(0)`	Use this in the associated Time view to find the time of the current spike.
`HCursor()`	Horizontal cursors 1-4 can be used in the spike shape view
`WindowVisible()`	Show, hide and maximise the window.
`Window()`	Position the dialog
`WindowGetPos()`	Find the dialog position
`WindowTitle$()`	Get and set the dialog title

SSButton()

This function sets and reads the toolbar and check box states of the current spike shape window. See the documentation of the spike shape dialogs for a description of the items. See `SSClassify()` for the Reclassify, New Channel and Online Update buttons.

```
Func SSButton({item%{, new%}});
```

`item%` If omitted, all items are reset to 0. Otherwise this identifies the item:

0	Overdraw spikes	0=no overdraw, 1=overdraw
1	Show template boundaries	0=hide, 1=show
2	Show non-matching spikes	0=hide, 1=show
3	Play sound for each spike	0=no sound, 1=sound
4	Scroll time view to track cursor 0	0=no track, 1=track
5	At end mode (online)	0=not at end, 1=at end
6	Circular replay	0=stop at end, 1=circular
7	Make templates	0=no, 1=yes
8	Build by code	0=by shape, 1=by code
9	Number of horizontal cursors	0=2, 1=4
10	Set which marker code to use	0-3 for layers 0-3
11	Overlay drawing of traces	0=no overlay, 1=overlay
12	Set size of displayed templates	0=small, 1=medium, 2=large
13	Collision analysis mode	0=no, 1=yes: Edit WaveMark only
14	Collision analysis use Mean width	0=no, 1=yes
15	Collision analysis Split as ideal	0=no, 1=yes: memory channel only

new% If present, this sets the state of the item. Use 1 to select the feature (depress the button or check the box) and 0 to deselect it. Item 10 supports values 0-3.

Returns The item state at the time of the call or -1 if the item does not exist or no item%.

See also:
HCursor(), SSChan(), SSClassify(), SSOpen(), SSParam(), SSRun(), SSTempDelete(), SSTempGet(), SSTempInfo(), SSTempSet(), SSTempSizeGet(), SSTempSizeGet(), XRange()

SSChan()

This function sets and gets the current channel in the current spike shape dialog.

```
Func SSChan({ch%});
```

ch% If omitted or 0, the return value is the current channel. If greater than 0, ch% sets the channel and the return value is the ch% (or the original channel if channel ch% is not suitable). If the channel changes, the old channel settings are saved.

If ch% is -1, the return value is the number of interleaved traces in the dialog. In the New Stereotrode or New Tetrode dialogs, this is 2 or 4. In the Edit WaveMark dialog it is the number of interleaved traces for the current channel.

If ch% is -2, this forces the channel configuration to be saved; the return value is 0. Use this before FileClose() as that does not save the current settings.

In a New Stereotrode or New Tetrode dialog ch% is a zero based channel index into the list of channels to combine and the return value is the channel number.

Returns The return value depends on the ch% argument.

See also:
SSButton(), SSClassify(), SSOpen(), SSParam(), SSRun(), SSTempDelete(), SSTempGet(), SSTempInfo(), SSTempSet(), SSTempSizeGet(), SSTempSizeSet(), XRange()

SSClassify()

This function is equivalent to the New Channel button in the New WaveMark dialog and to the Reclassify and Online Update button in the Edit WaveMark dialog. The arguments from new% onward are ignored in the Edit WaveMark dialog.

```
Func SSClassify({speed%{, new%{, type%{, noQu%}}}});
```

speed% If this is omitted, the function returns 1 if the dialog is currently reclassifying or creating a new channel and 0 if it is not. If present, use the values:

- -2 Equivalent to clicking the Online Update button when sampling.
- -1 To cancel any ongoing reclassification or writing to a new channel.
- 0 Start reclassifying or writing a new channel with display updates. This takes place in idle time, so you must provide some with Yield() or by using Toolbar(), Interact() or DlgShow().
- 1 Reclassify or write a new channel in fast mode, with no display updates. The operation will complete before control returns to the script.

new% The channel number to create in the New WaveMark dialog.

type% The channel type to create. You can use types: 2 (Event-), 3(Event+), 4(Level), 5(Marker) or 6 (WaveMark) only. If omitted, type 6 (WaveMark) is used.

noQu% If the channel set by new% is already in use and this argument is omitted or 0, the user is asked if they want to overwrite it. Set non-zero for no query.

Returns If speed% is omitted, the return value is 1 if reclassification or writing a new channel with display updates is in progress, otherwise 0. If speed% is present, the return value is 1 if the action was completed, 0 if it could not be done.

See also:
HCursor(), SSButton(), SSChan(), SSOpen(), SSParam(), SSRun(), SSTempDelete(), SSTempGet(), SSTempInfo(), SSTempSet(), SSTempSizeGet(), SSTempSizeSet(), XRange()

SSColApply()

This function applies the result of collision analysis to the current spike in an Edit WaveMark dialog. The dialog must be in Collision analysis mode.

```
Func SSColApply({act%});
```

act% This optional variable determines the action to perform. If the value is omitted it is taken as 0. The possible values are:

- 0 Test if it is possible to split the waveforms (equivalent to using the Split spike button in the collision analysis dialog). This can only be done if the current channel of spikes is stored in a memory channel. The return value is 1 if splitting the current spike is possible, 0 if not, -1 if this is not a memory channel.
- 1 Equivalent to the Copy codes button in the collision analysis dialog. The return value is the number of codes that were copied. This will normally be 2 or 1, being the number of templates that contributed to the best match.
- 2 Equivalent to the Split spike button. The return value is -1 if it is not possible to split the current spike, otherwise the return value is the number of spikes that the current spike is split into (typically 2, but can be 3 if two templates match and do not cross the x axis 0 point in the dialog).

Returns If the current view is an Edit WaveMark dialog, but it is not in collision analysis mode, the return value is -1. If it is any other form of WaveMark dialog, the return value is always 0. Otherwise, the return value is as described above.

Example

The following code opens a spike shape window on the first suitable channel, changes to collision mode, then iterates through 100 spikes and copies matching codes to the second and third marker codes for all events that match 2 templates and for which the error value is less than a defined value. This assumes you have suitable templates set. The script lists the positions and matching values for all the events that satisfied the conditions. Note the use of `View(ViewLink()).Cursor(0)` to get the position of the current spike in the data file.

```
var ss%;                    ' handle of spike shape dialog
ss% := SSOpen(1,1);         ' Open an Edit WaveMark dialog
if ss% < 0 then halt endif; ' Oops, not possible for some reason
SSButton(13,1);             ' Set collision analysis mode

const minErr := 0.7;        ' some arbitrary minimum error
var n%,n1%,n2%,p1,p2,err;
var i%;
var nCode% := SSButton(10, 1);   ' Make code 1 the current one
for i% := 1 to 100 do
    n% := SSColInfo(err,n1%,n2%,p1,p2);
    if (n% = 2) and (err < minErr) then ' if acceptable
        SSColApply(1);          ' copy codes starting at code 1
        PrintLog("%10.5f %5.2f %2d %2d %5.2f %5.2f\n",
           View(ViewLink()).Cursor(0), err, n1%, n2%, p1, p2);
    endif;
    SSRun(1);                   ' step to the next item
next;
SSButton(10, nCode%);       ' restore the original code
```

See also:

SSButton(), SSColArea(), SSColInfo(), SSRun()

SSColArea()

This function gets and sets the *Important area* of the template used for collision analysis. The command may only be used when the current view is a spike shape dialog in collision analysis mode.

Note that this area is reset to lie within the template region each time the dialog enters collision analysis mode - use `SSButton(13,1);` to do this. The initial important area is set to match the template area. Then, if this area is more than 4 points, the area is inset by two points at start and end.

The command has two variants:

Get area

```
Func SSColArea(&start%)
```

`start%` This is an integer variable, that is returned holding the zero-based offset from the start of the displayed data to the collision analysis area.

Returns The return value is the original size of the area used for collision analysis, or -1 if the spike shape dialog is not in collision analysis mode.

Set area

```
Func SSColArea(newSt%, newSz%);
```

`newSt%` This sets the start offset of the collision analysis area.

`newSz%` This sets the size of the area (in data points) used for collision analysis.

Returns The return value is the original size of the area used for collision analysis, or -1 if the spike shape dialog is not in collision analysis mode.

Changing the area size will trigger a new matching process. Each time you enter collision analysis mode, the Important area is set to start two points after the start of the template area and end two points before the end of the template area unless this would make the area of zero size, when it is set to match the template area. In most cases, this does what you want.

Example

The following example increases the size of the current important area by one point in each direction

```
var start%,size%, ss%;
ss% := SSOpen(1,1);          'Open an Edit WaveMark dialog
if ss% < 0 then halt endif;  'Oops, not possible for some reason
SSButton(13,1);              'Collision analysis mode
size% := SSColArea(start%);  'get current size
SSColArea(start%-1, size%+2);'set new size
```

See also:

`SSButton()`, `SSColInfo()`

SSColInfo()

This function can be used only when a Spike shape window is the current view and it is in Collision Analysis mode. The function gets information about the current state of the matching.

```
Func SSColInfo(&error{, &tmp1%{, &tmp2%{, &pos1{, &pos2}}}});
```

`error` This is a real variable that is returned holding the mean error per point. This is mean ratio of the difference between the waveform and the waveform templates with respect to the template width (either point by point, or to the mean width over all templates, depending on the Mean width check box).

`tmp1%` An integer variable that is returned holding the 0-based index of the template that matches furthest to the left.

`tmp2%` An integer variable that is returned holding the 0-based index of the template that matches furthest to the right. If only one template matches, this is negative.

`pos1` A real variable that is returned holding the point position in the display at which the first point of the first template starts. This will, in general, be a fractional position as the best match is arrived at by interpolation.

`pos2` This returns the position of the second matching template in the same format as `pos1` . If there is no second template, the value is set the same as `pos1`.

Returns The number of matching templates (0, 1 or 2) or -1 if the spike shape dialog is not in collision analysis mode.

Example

The following code opens a spike shape window on the first suitable channel, changes to collision mode, then iterates through 100 spikes and displays the matching information for all the ones it finds that match 2 templates. This assumes you have suitable templates set. Note the use of `View(ViewLink()).Cursor(0)` to get the position of the current spike in the data file.

```
var ss%;                    ' handle of spike shape dialog
ss% := SSOpen(1,1);         ' Open an Edit WaveMark dialog
if ss% < 0 then halt endif; ' Oops, not possible for some reason
SSButton(13,1);             ' Set collision analysis mode

var n%,n1%,n2%,p1,p2,err;
var i%;
for i% := 1 to 100 do
    n% := SSColInfo(err,n1%,n2%,p1,p2);
    if n% = 2 then          ' if we match two item
        PrintLog("%10.5f %5.2f %2d %2d %5.2f %5.2f\n",
           View(ViewLink()).Cursor(0), err, n1%, n2%, p1, p2);
    endif;
    SSRun(1);               ' step to the next item
next;
```

See also:

`SSButton()`, `SSColArea()`, `SSOpen()`

SSOpen()

This function opens and returns information about spike shape dialogs. The current view can be any view that has an associated time view (including a spike shape dialog). Close this dialog with `FileClose()`. Prior to version 9, the current view had to be a time view.

```
Func SSOpen({type%{, mode%});
```

`type%` This argument is assumed to be zero if omitted. Allowed values are:

- -1 The function returns the type of any open spike shape dialog associated with the current view or 0 if there is no open spike shape dialog. Returned values are 1=Edit WaveMark, 2=New WaveMark, 3=New n-trode.
- 0 The return value is the window handle of any open spike shape dialog associated with the current view, or 0 if there is no open dialog.
- 1 Open the Edit WaveMark dialog and return its handle or 0.
- 2 Open the New WaveMark dialog and return its handle or 0.
- 3 Open the New n-trode dialog with selected channels. Return its handle or 0.

`mode%` If this is zero or omitted, the dialog is created invisibly. Set 1 to make it visible. This is ignored unless `type%` is greater than 0.

Returns The return value depends on the `type%` argument.

When you open a spike shape dialog, it will behave exactly as if you had opened it interactively, so the state will depend on the previous usage; it is up to you to set the dialog features you require with `SSParam()`, `SSButton()`, `SSTempInfo()`, `SSTempSet()` and the like.

See also:
HCursor(), SSButton(), SSChan(), SSClassify(), SSParam(), SSRun(), ViewLink(), SSTempDelete(), SSTempGet(), SSTempInfo(), SSTempSet(), SSTempSizeGet(), SSTempSizeSet(), XRange()

SSParam()

This function sets the template parameters for the current spike shape dialog and reads back the state of individual parameters. This is equivalent to the template parameters dialog; see that dialog for a full description of the arguments. The command has two variants. In the first you supply all the arguments (use the value -1 to leave an argument unchanged). In the second variant you can get or set the value of a single item.

```
Func SSParam(new%,wide,rare%,AMax,PMin,flg%,mode%,nAT%,int%,tc%);
Func SSParam(item%{, value});
```

`new%` If `new%` is greater than 0, this sets number of spikes for a new template.

`wide` The width of a new template as a percentage of the template amplitude.

`rare%` Templates with spikes rarer than 1 in `rate%` are discounted.

`AMax` Maximum amplitude change for tracking or 0 for no amplitude tracking.

`PMin` Minimum percentage of point in the template for a possible match.

`flg%` This is the sum of: 1 = use minimum percentage only when building templates, 2 = Remove DC value from spike data before matching, 4 = No Independent triggering with multiple traces.

`mode%` The mode for adding spikes to the template: 0=All, 1=Autofix, 2=track.

`nAT%` The number of spikes for Autofix and track modes.

`int%` The interpolation method to use: 0=linear, 1=parabolic, 2=cubic spline.

`tc%` The high-pass time constant to apply to the data expressed as 2 to the power `tc%` data samples with `tc%` in the range 1 to 30. Use 31 for no high pass filter.

`item%` Used in the second command variant. Set -2 to copy the current settings to all channels. Set -1 to reset the values to a standard state. Set 0 to close the parameters dialog if it is open. Otherwise set 1 to 10 to select one of the arguments `new%` (1) to `tc%` (10) and the return value will be the value of that item at the time of the call.

`value` If present, this sets the new value of the parameter when `item%` is greater than 0.

Returns The first command variant returns 0. The second variant, with `item%` greater than 0, returns the item value at the time of the call and `item%` less than zero returns 0. For `item%`=0, the return value is 1 if the parameter dialog was open.

See also:
HCursor(), SSButton(), SSChan(), SSClassify(), SSOpen(), SSRun(), SSTempDelete(), SSTempGet(), SSTempInfo(), SSTempSet(), SSTempSizeGet(), SSTempSizeSet(), XRange()

SSRun()

This function gets and sets the run state of the current view, which must be a spike shape window. Use `Cursor(0)` to get the current run position. You can use `ViewLink()` to find the time view associated with the spike shape dialog. If the spike shape view is currently reclassifying, changes to the run state are ignored.

```
Func SSRun({run%});
```

`run%` If omitted, no change is made, otherwise negative values are backwards and positive are forwards. 0 means stop, 1=step, 2=run as fast as possible, 3= run in real time, 4= run at 91 Hz, 5=31 Hz and so on up to 13=1 Hz. Impossible requests (such as run backwards in the New WaveMark dialog) are treated as stop. For `run%` values greater than 1, the minimum interval between searches is approximately `5 + 6.9 * (run%-1)*(run%-1)` milliseconds. Negative values of `run%` generate the same rates as `-run%` with backwards searches, where allowed.

Returns The run state at the time of the call.

A request to step will do one search forwards/backwards and then the run state will be stopped. A request to run will do one search, then leave things in the requested run mode expecting to do additional searches in idle time. If the search hits the end of the search region and Circular replay mode is off, the run mode will be cancelled.

See also:
`HCursor(), SSButton(), SSChan(), SSClassify(), SSOpen(), SSParam(), SSTempDelete(), SSTempGet(), SSTempInfo(), SSTempSet(), SSTempSizeGet(), SSTempSizeSet(), XRange()`

SSTempDelete()

This function deletes one or more templates for the current channel of the current view, which must be a spike shape window. If this command is used with no arguments, or with both arguments set to -1, the entire template system is cleared, equivalent to clicking the clear all templates button in the dialog.

```
Func SSTempDelete({n%{, code%}});
```

`n%` The zero-based index of a template or -1 for all templates. Templates that match this index and the `code%` argument are deleted. If this is omitted, -1 is used.

`code%` If this is omitted or set to -1, all templates codes are deleted. Otherwise only templates selected by `n%` that match `code%` are deleted.

Returns The number of deleted templates or -1 if `n%` is not a template index.

See also:
`SSButton(), SSChan(), SSClassify(), SSOpen(), SSParam(), SSRun(), SSTempGet(), SSTempInfo(), SSTempSet(), SSTempSizeGet(), SSTempSizeSet(), XRange()`

SSTempGet()

This function gets information about the current set of templates for the current channel of the current view, which must be a spike shape window. It can also return the current raw data displayed in the template window.

```
Func SSTempGet(n%{, temp[]{[]}{, what%{, &count%}});
```

`n%` This is the zero-based index of the template to return information from or to -1 to get the last triggered waveform or -2 to get the currently displayed waveform (triggered or not). Use `SSTempSizeGet()` to get the number of displayed points and the number of points in the template. The -2 code did not exist before version 6.11.

`temp` This optional array is filled with template data. If there is insufficient data to fill it, unused entries are unchanged. An integer or real array can be used. If the template has multiple traces, use `temp[points%][traces%]` to get real data and `temp%[points%][traces%]` to get integer data. See `ChanScale()` for an explanation of the use of integer and real data for waveform values.

`what%` Omit this or set it to 0 to return the mean template waveform. It is ignored if `n%` is 1. 1 = return the template upper limit, 2 = return the lower limit, 3 = return the template width. The width is half the distance between the upper and lower template limits.

`count%` This optional integer variable is returned set to the number of events that were accumulated into this template. It is ignored if `n%` is 1.

Returns If `n%` is positive, the return value is the template code or -1 if the template does not exist. If `n%` is -1, in Edit WaveMark dialogs, the return value is the currently selected sort code of the spike or -1 if you are on-line and the current data did not trigger (background data). In Create WaveMark dialogs the return value is 1 for triggered data and 0 for background data.

See also:
`ChanScale(), SSButton(), SSChan(), SSClassify(), SSOpen(), SSParam(), SSRun(), SSTempDelete(), SSTempInfo(), SSTempSet(), SSTempSizeGet(), SSTempSizeSet(), XRange()`

SSTempInfo()

This function gets and sets template information for the current channel of the current view, which must be a spike shape window.

```
Func SSTempInfo({n%, item%{, value%{, noLim%}});
```

`n%` If this is omitted, the return value is the number of templates. If present, it is the zero-based index of a template or -1 for all templates (if appropriate). When returning a value, if -1 is used, the value for template 0 is returned.

`item%` This selects the information to get or set. Items 0-2 operate on all templates. Items 3-6 operate on visible templates only (`n%` in the range 0 to 19).

- 0 Get the index of the first template, starting at `n%`, that has a code of `value%`.
- 1 Get the code for template `n%`. It sets the code if `value%` is greater than 0 and less than 256.
- 2 Get the count of spikes added into template `n%`.
- 3 Get the event counter (spikes that matched the template). If `value%` is positive, it sets the event count.
- 4 Get the lock state of template `n%`. `value%`=0 to clear the lock and >0 to set it.
- 5 Change the template width by `value%` pixels. Set `value%` to 1 and -1 to mimic the increase and decrease width buttons. `noLim%` can be used.
- 6 Change the template width by `value%` channel ADC units. `noLim%` can be used.
- 7 In collision analysis mode, get or set the exclude template check box state. `value%` is 1 to exclude, 0 to include and omitted or negative for no change. You cannot exclude all templates, at least one must be left included.

`value%` Use this argument when setting values and to locate codes when `item%` is 0.

`noLim%` Set non-zero with item codes 5 and 6 to remove the template width limits that are usually enforced to stop the template becoming too wide or too narrow. Each point of a template has a mean value and a width. It also has a minimum width, which is initialised to the original width when the template was created. The width is allowed to increase to up to 4 times the minimum width, but not to become less that the original. If you allow the width to override these limits, the minimum width is adjusted, as required.

Returns The value selected by `item%` at the time of the call. The return value is 0 for items 5 and 6. The return value is -1 if a template is not found.

See also:
`HCursor(), SSButton(), SSChan(), SSClassify(), SSOpen(), SSParam(), SSRun(), SSTempDelete(), SSTempGet(), SSTempSet(), SSTempSizeGet(), SSTempSizeSet(), XRange()`

SSTempSet()

This function creates a template or adds the current spike in the window to a nominated template in the current channel of the current view, which must be a spike shape view. The first command variant creates a new template from user-supplied data.

The second variant adds the currently displayed data in the window to a nominated template or creates a new template from it. In the Edit WaveMark dialog this also sets the classification code of the displayed data. If this code is not in the current Marker Filter for the source channel, the displayed data changes to the next available spike. It can also renumber and sort the displayed templates and to update online templates in the 1401 to match the templates in the dialog.

```
Func SSTempSet(const temp[]{[]}, code%, wide{, count%});
Func SSTempSet({n%{, code%}});
```

`temp` Used when creating a new template from arrays of data. This is a one or two dimensional array holding the template shape. An integer or real array can be used. If the template has multiple traces, use `temp[points%][traces%]` for real data and `temp%[points%][traces%]` for integer data. See

ChanScale() for an explanation of the use of integer and real data for waveform values. If the array is too short or has too few traces, missing values are set to 0.

code% This is used when creating a new template. Set it to 0 (or omit it in the second command variant) to use the lowest unused code or set it to the code to use for the new template in the range 1-255.

count% This optional argument has a default value of 1, and sets the number of spikes that contributed to the template.

wide A number or an array holding the template width, which is half the distance between the upper and lower template limits. If you supply an array, the shape of this array should match the temp array.

If you supply a number, all points are given the same width. This is in user units if temp is a real array and is in channel units if temp is an integer array.

n% Optional. If omitted, the currently displayed waveform is added to the best-fit template. Set it to the zero-based index of an existing template to add the current waveform to the template. Set to -1 to add a new template, in which case code% determines the code of the new template. Set -2 to renumber the existing templates, in which case code% sets the lowest numbered code used. Set -3 to sort the templates into ascending code order. Set -4 to update online templates during sampling.

Returns The zero-based index of the template that was added or modified or -1 if there was a problem (no matching template, unknown template index or no current waveform). The second variant with n% less than -1 returns 0.

See also:
ChanScale(), SSButton(), SSChan(), SSClassify(), SSOpen(), SSParam(), SSRun(), SSTempDelete(), SSTempGet(), SSTempInfo(), SSTempSizeGet(), SSTempSizeSet()

SSTempSizeGet()

This function gets the template size and the displayed data area for the current channel. The current view must be a spike shape window. All arguments are integer variables that are set to the template and display sizes.

```
Func SSTempSizeGet({&start%{, &show%{, &pre%}}});
```

start% If present, set to the point offset to the template start in the raw data display area.

show% If present, set to the number of data points in the raw data display area.

pre% If present, set to the number of pre-trigger points in the raw data display area.

Returns the number of data points in the template.

See also:
SSButton(), SSChan(), SSClassify(), SSOpen(), SSParam(), SSRun(), SSTempDelete(), SSTempGet(), SSTempInfo(), SSTempSet(), SSTempSizeSet()

SSTempSizeSet()

This sets the template size and the displayed data area. The template must lie within the displayed data area with at least 2 data points before and after the template. If you set a template area that exceeds these limits, the command attempts to increase the display area (if this is possible). Use YRange() and Optimise() to set the displayed y range,

```
Proc SSTempSizeSet(start%{, n%{, show%{, pre%}}});
```

start% This is the offset to the beginning of the template from the start of the displayed area. It must be at least 2. Set -1 to leave the start point unchanged.

n% This sets the number of data points in the template. To keep the current number of points, either omit this argument or set it to -1.

show% The requested number of points to display in the main data window; it has no effect in the Edit WaveMark dialog. Omit show% or use -1 for no change. If odd, show% is reduced by 1. It is also adjusted if it is too large or too small.

pre% This sets the number of pre-trigger points to display in the main data window in the range 0 to show%-1. This has no effect in the Edit WaveMark dialog. Omit pre% or set -1 for no change. Out of range values are set to the appropriate limit.

See also:
Optimise(), SSButton(), SSChan(), SSClassify(), SSOpen(), SSParam(), SSRun(), SSTempDelete(), SSTempGet(), SSTempInfo(), SSTempSet(), SSTempSizeGet(), YRange()

Str$()

This converts a number to a string.

```
Func Str$(x {,width% {,sigd%}});
```

x A number to be converted.

width% Optional minimum field width. The number is right justified in this width. From version [10.00], when x is real, this can be set to -1 and sigd% omitted to generate the minimum number of significant figures to represent the value such that when read back, it would give the same value.

sigd% Optional number of significant figures in the result (default is 6) or set a negative number to set the number of decimal places as -sigd%.

Returns A string holding a representation of the number.

See also:
Asc(), Chr$(), DelStr$(), InStr(), LCase$(), Left$(), Len(), Mid$(), Print$(), Print(), Replace$(), Reverse$(), Right$(), UCase$(), Val()

StrToChanY()

This function lets you evaluate a text string in exactly the same way that a dialog expression evaluates a channel expression. It is here to allow us to test dialog expressions from a script, but it could occasionally be useful from a script.

```
Func StrToChanY(expr$, chan%, &value);
```

expr$ The channel dialog expression to evaluate.

chan% The channel to evaluate it for or 0 if no channel is needed. Expressions involving only horizontal cursors do not need a channel as each horizontal cursor is attached to a channel.

value A real variable returned holding the result of a successful evaluation.

Return 1 for success, 0 if the expression could not be evaluated.

See also:
StrToViewX()

StrToViewX()

This function lets you evaluate a text string in exactly the same way that a dialog expression evaluates a view expression. It is here to allow us to test dialog expressions from a script, but it could occasionally be useful from a script.

```
Func StrToViewX(expr$, &value);
```

expr$ A view x axis expression to be evaluated.

value A real variable returned holding the result of a successful evaluation.

Return 1 for success, 0 if unable to evaluate the expression.

See also:
StrToChanY()

Sweeps()

This function returns the number of items accumulated in the result view.

```
Func Sweeps({set%});
```

`set%` If present, this sets the number of sweeps held by the view. You might use it to sum waveform averages in a result view so that the sweep count is correct.

Returns The value returned depends on the type of the result view (see the `SetXXXX` commands). It is an error to use this in any type of view other than a result view.

See also:
`SetAverage(), SetEvtCrl(), SetINTH(), SetPhase(), SetPower(), SetPSTH(), SetWaveCrl()`

System()

This function returns the operating system version as a number and gets information about desktop screens and about the time consumed by the GUI thread. Use the `App()` command to get the version number of Spike2.

```
Func System({get%{, scr%{, sz%[]}}});
Func System(3, &user, &kernel);
```

`get%` A value that determines which information to return. If omitted, the value is taken as 0. Values 2 and 3 were added at Spike2 version 9.01.

- 0 The return value is the operating system revision times 100: 351=NT 3.51, 400=95 and NT 4, 410=98, 490=Me, 500=NT 2000, 501=XP, 502=Windows Server 2003, 600=Vista, Windows Server 2008, 601=Windows Server 2008 R2, Windows 7, 602=Windows 8, Windows Server 2012, 1000=Windows 10. Spike2 version 9 requires Windows 7 or later (it may work in Vista with the correct libraries installed).
- 1 The function returns information about installed desktop monitors (see `scr%` and `sz%[]`)
- 2 Set the base time for collecting the time used by the current GUI thread. There must only be one argument. This is for CED diagnostic use. If you do not use this, the value read back by setting `get%` to 3 will be relative to the start of Spike2 (not to the start of the current script).
- 3 Read back the time used in user and kernel mode of the GUI thread in seconds. There must be three arguments. This is for CED diagnostic use.

`scr%` Set 0 to return the number of desktop monitors; `sz%[]` gets the pixel co-ordinates of the desktop. Set to n (>0) to get the pixel co-ordinates of screen n; returns 1 for the primary monitor, 0 if not and -1 if it does not exist. Add 1000 to `scr%` to get the pixel co-ordinates of screen n with any areas reserved by the system (for example the task bar) removed.

`sz%[]` Optional array of at least 4 elements to return pixel positions. Elements 0 and 1 hold the top left x and y, 2 and 3 hold bottom right x and y.

`user` Returned holding the time in seconds that the GUI thread has executed for in user mode since the last call to `System(2)`. The result is meaningless if you do not call `System(2)` first.

`kernel` Returned holding the time in seconds that the GUI thread has executed for in kernel mode since the last call to `System(2)`.

Returns A value that depends on `get%` as described above.

See also:
`App(), System$(), Window(), WindowVisible()`

System$()

This returns the operating system name and accesses Spike2 environment variables and the program command line. The environment holds a list of strings of the form "name=value". You can get or set the value associated with name. You can also read all the strings into a string array.

Get operating system name

To get information about the operating system use the command with no arguments:

```
Func System$()
```

Returns With no arguments, it returns: "Windows SS build n" where SS is the operating system and n is the build number.

Get and optionally set an environment variable

Each process has its own environment. A process started with ProgRun() inherits a copy of the Spike2 environment, so you can pass it information. However, you cannot see environment changes made by the new process.

```
Func System$({var$ {,value$}});
```

var$ If present, this is the name of an environment variable (case insensitive).

value$ If present, the new value. An empty string deletes the environment variable.

Returns The value of the environment variable identified by var$ or an empty string if this variable does not exist.

Get a list of environment variables

This command variant reads the environment strings into a string array.

```
Func System$(list$[] {,&n%});
```

list$ An array of strings to fill with environment strings of the form "name=value".

n% An optional integer that is returned holding the number of elements copied.

Returns An empty string.

Get elements of the command line that started Spike2, Spike2 version

This command variant get elements of the command line used to start Spike2. From version [10.13] you can use it with a negative argument to get additional information.

Spike2 can be started with a command line and it can be useful to find out what this held. In particular, Spike2 ignores command line switches that it does not recognise, so a command line that commands a script to run can then interpret command line switches that are defined by the user.

```
Func System$(cmd%);
```

cmd% -2 to get the Spike2 version information as displayed in the About Spike2 box.

-1 to get the operating system name (the same as omitting cmd%).

Positive values return the elements that make up the program command line used to start Spike2. The first element is 0, the second 1 and so on. If a command element is a flag (introduced by - or /), it starts with a /. The return value is an empty string if the element does not exist. The command line variant was added at Spike2 version [9.01].

Beware: negative values of cmd% cause a fatal error in versions of Spike2 that do not support it. You can check the version with _Version.

Returns The item requested by cmd% or an empty string if there is no such element.

Get Spike2 version string and build information

This was added at version [10.13] to collect the Spike2 version information

```
Func System$(-2);
```

Returns Text to identify the Spike2 version.

Examples of use

```
var list$[200], value$, n%, i%;
PrintLog("%s\n", System$());  'Print OS name
System$("fred","good");       'Assign the value "good" to fred
PrintLog("%s\n", System$("fred")); 'get value of fred
System$("fred","");           'Delete fred from the environment
System$(list$[], n%);         'Print all environment strings
for i%:=0 to n%-1 do PrintLog("%s\n",list$[i%]) next;
```

See also:

ProgRun(), System()

T

TabSettings()

This sets and gets the tab settings for a text view. Any changes you make apply to the current view only. If you want to change the tab settings for all views, open the Edit menu preferences General tab and click the appropriate button in the Text view settings. It is possible to do this using a script with the Profile() command.

```
Func TabSettings({size%{, flags%}});
```

size% Tab sizes are set in units of the width of the space character in the Default style set for the view (style 32). Values in the range 1 to 100 set the tab size. If size% is 0 or omitted, no change is made. If size% is -1, the return value is the current flags% value for the text view.

flags% If omitted, no change is made to the flags. Otherwise, this is the sum of flag values: 1=Keep tab characters (the alternative is replace tabs with spaces), 2=show indents.

Returns If size% is positive, the return value is the tab size at the time of the call. If size% is -1, the return value is the flags% value at the time of the call.

See also:

FontGet(), FontSet(), Profile()

TalkerChanInfo()

Get information about a talker channel. If the talker is not currently loaded, this gets that last-known set of information. The arrays passed to the command can be of any length. Only the indices that hold data are updated.

```
Func TalkerChanInfo(talk$, chan%, info$[]{, info%[]{, info[]}});
```

`talk$` The name of the talker we want information about. Matching of names is case insensitive.

`chan%` The 0-based channel index.

`info$` An array of strings used to return text information. Data at each index is:

- 0 Channel title.
- 1 Channel description (comment).
- 2 Unique channel ID (may be blank).
- 3 Channel units (where applicable).

`info%` An array of integer values holding channel information. Data at each index is:

- 0 Flags indicating channel options (see the Talker documentation). This is the `TCF_Xxxx` set of flags. Likely of interest only to CED engineers.
- 1 Count of data items attached to a marker channel, otherwise 0.
- 2 Number of pre-trigger points for a WaveMark channel.
- 3 Number of data traces in a WaveMark channel.

`info` An array of real information, for rates and data ranges. Data at each index:

- 0 Data item rate (or expected rate) on the channel.
- 1 Waveform sample rate of a waveform or WaveMark channel.
- 2 Maximum waveform data value (or 0 if unset).
- 3 Minimum waveform data value (or 0 if unset).

Return The channel type or -1 if unknown Talker or -2 if unknown channel. Channel types are:

- 1001 16-bit waveform channel.
- 1002 Event channel (time stamps).
- 1003 Marker channel (time stamp plus codes).
- 1004 WaveMark (16-bit waveform plus marker codes).
- 1005 RealMark (list of 32-bit real data plus marker codes).
- 1006 TextMark (text string plus marker codes)
- 1007 RealWave (waveform of 64-bit real data), but seen by Spike2 as 32-bit reals.
- 1008 Level data (time stamps for rising and falling edges).
- 1009 RealWave (waveform of 32-bit floating point data).

Types 1007 and 1009 are the same, as far as Spike2 is concerned, but differ in how they are transmitted from the talker to Spike2.

See the `TalkerInfo()` description for an example of using this command.

See also:

`TalkerInfo()`, `TalkerReadStr()`, `TalkerSendStr()`, `SampleTalker()`

TalkerInfo

This function gets information about talkers known to Spike2. It was added at version [10.21]. There are several variants:

Get list of known talkers

This variant gets the number of talkers known to Spike2 and fills a vector with the talker names.

```
Func TalkerInfo({list$[]});
```

`list$` If present, this is an array of strings. If there are `n` known talkers, the first `n` elements are set to the names of the talkers. If this array has fewer elements, excess names are ignored. If the array has more elements, the extra elements are unchanged.

Returns The number of talkers that are known to Spike2.

Get talker state, load talker

This variant gets information about a specific Talker or loads an unloaded Talker.

```
Func TalkerInfo(talk$[, opt%]);
```

`talk$` The name of the talker. Matching of names is case insensitive.

`opt%` Determines the action to take with the talker.

- 0 (or omitted) to get flags associated with the talker. These flags are the sum of: 1 if the talker is currently loaded, 2 if the talker can be run as location known, 4 if the talker is remote (not running on the same computer as Spike2), 8 if the talker is active (in use for sampling).
- 1 Start the talker load process but do not wait for the talker to be loaded. The return value is 2 if already loaded, 1 if the talker loaded, 0 if not (yet) loaded, -1 if unknown talker, -2 if not found, -3 if timed out, -4 if not runnable, -5 if remote. You must release system time with `Yield()` or some user interaction with idle time to allow loading to be reported.
- 2 Load the talker and wait until it is loaded. Return values are the same as for option 1.
- 3 Get the number of channels that you can get information about with `TalkerChanInfo()`.

Return -1 if the talker is unknown, or one the values described above.

Get talker information

This variant gets talker details. We use arrays for each argument to allow us to add additional information in future versions. The arrays passed to the command can be of any length. Only the indices that hold data are updated. Much of this information is of interest only to CED Engineers.

```
Func TalkerInfo(talk$, info$[], {info%[]{, info[]}});
```

`talk$` The name of the talker. Matching of names is case insensitive.

`info$` An array of strings used to return text information. Data at each index is:

- 0 Description of the talker.
- 1 Text used to differentiate talker drift settings. For example, if a talker links to a device that determines the sample timing, the drift settings depend on the linked device. This field identifies the device. This can be blank.
- 2 Full path to the Talker `.exe` file. Can be blank if unknown (in which case Spike2 cannot start the talker).
- 3 Talker machine name or blank if local (running on same machine as Spike2). If this is not blank, Spike2 cannot start the talker.
- 4 The command line string used to run the talker.

`info%` An array of integer values holding talker information. Data at each index is:

- 0 The number of data channels known to the talker.
- 1 Talker-specific version number *100 (121 means version 1.21).
- 2 Earliest version of the talker that this one is compatible with.
- 3 Talker specification that this talker is written to.
- 4 A Unique ID for configuration information, or 0.
- 5 Flags indicating talker options (see the Talker documentation). This is the `TKF_Xxxx` set of flags.
- 6 1 if drift information is locked, 0 if it is not.

`info` An array of real information, currently all holding time drift information. Data at each index:

- 0 Saved drift rate for use at the start of sampling in seconds/second.
- 1 Standard deviation of drift rate or 0 if not set.
- 2 Length of data over which the drift was measured.

Returns 0 for OK or -1 if the talker is unknown.

Example code

This example forms a list of talkers, finds one that is loadable and not loaded, requests that it should load without waiting (so that other script actions are possible during the wait), and reports if the talker loaded. Finally, it gets and displays information about the Talker and all the known channels.

```
var Talkers$[0];                    ' Space for a list of talkers
var nTalk% := TalkerInfo();         ' Get number of talkers
resize Talkers$[nTalk%];            ' Make list big enough
TalkerInfo(Talkers$);               ' Collect all known talker names
'List talkers and talker flags
var i%, flags%, load$;
for i% := 0 to nTalk%-1 do
    flags% := TalkerInfo(Talkers$[i%], 0);
    PrintLog("%10s: %d\n", Talkers$[i%], flags%);
    if (flags% & 3) = 2 then        ' find an unloaded, but loadable talker
        load$ := Talkers$[i%];      ' Save name of last loadable talker
    endif
next

flags% := TalkerInfo(load$, 1);     ' Tell it to load, no wait
if (flags% = 0) then                ' if no error and not loaded
    Seconds(0, 1);                  ' set timer 0
    repeat
        Yield();                    ' allow idle time
        'Do other stuff here
        flags% := TalkerInfo(load$, 0);
    until ((flags% > 0) && (flags% & 1)) || (Seconds() > 4);
endif

if (flags% >= 0) && (flags% & 1) then
    PrintLog("%s is loade\n", load$);
else
    PrintLog("%s did not load, code %d\n", load$, flags%);
endif
flags% := TalkerInfo(load$, 3);     ' get channel count
PrintLog("Talker %s has %d channels\n", load$, flags%);
TalkerDisplay(load$);  'Display talker details
```

A routine to display all the talker information:

```
'Display information about talker amd channels
Proc TalkerDisplay(talk$)
var info$[5], info%[7], info[3];
var ok% := TalkerInfo(talk$, info$, info%, info);
const titleStr$[5] := {"Description", "DriftID", "Path", "Machine", "Command"};
var i%;
for i% := 0 to 4 do
    PrintLog("%11s: %s\n", titleStr$[i%], info$[i%]);
next;

const titleInt$[7] := {"Channels","Version","Earliest","TalkSpec", "UniqueID", "Flags", "Locked"};
for i% := 0 to 6 do
    PrintLog("%11s: %d\n", titleInt$[i%], info%[i%]);
next;

const titleFlt$[3] := {"Drift s/s", "Drift SD", "Drift time"};
for i% := 0 to 2 do
    PrintLog("%11s: %g\n", titleFlt$[i%], info[i%]);
next;

const nChan% := info%[0];   'The number of channels
var chan%;
for chan% := 0 to nChan%-1 do
    TalkerChanDisplay(talk$, chan%);
next
end;
```

Routine to display channel information:

```
'Display information about a talker channel
Proc TalkerChanDisplay(talk$, chan%)
var info$[4], info%[4], info[4];
var kind% := TalkerChanInfo(talk$, chan%, info$, info%, info);
if (kind% < 1001) then
    PrintLog("Channel index %d is unknown\n", chan%);
    return;
endif

const kind$[9] :=
{"Wave (16-bit)", "Event", "Marker",
 "WaveMark", "RealMark", "TextMark",
 "Wave (Double)", "Level", "Wave (float)"};

PrintLog("Channel index %d, Type %s:\n", chan%, kind$[kind%-1001]);
const TitleStr$[4] := {"Title", "Description", "UniqueID", "Units"};
var i%;
for i% := 0 to 3 do
    PrintLog("%11s: %s\n", titleStr$[i%], info$[i%]);
next;

const titleInt$[4] := {"Flags","Items","PreTrig","Traces"};
for i% := 0 to 3 do
    PrintLog("%11s: %d\n", titleInt$[i%], info%[i%]);
next;

const titleFlt$[4] := {"Item rate", "Wave rate", "Maximum", "Minimum"};
for i% := 0 to 3 do
    PrintLog("%11s: %g\n", titleFlt$[i%], info[i%]);
next;
end;
```

See also:
`TalkerChanInfo()`, `TalkerReadStr()`, `TalkerSendStr()`, `SampleTalker()`

TalkerReadStr()

This function controls the reading of text strings and function codes from a named talker. You could use this together with the `TalkerSendStr()` command to configure a talker before sampling (but not in a way that would cause any data it transferred to be incompatible with the target channel) or to receive commands from the talker during sampling. The original talker specification (revision 1), does not support this function. Your talker must support at least talker interface specification revision 2 to support this command. You can get information about known talkers from the Sample menu Talker list command.

The talker support in Spike2 will queue up messages; messages are not timed out if you do not read them. There is currently a limit of 20 on the number of messages that can be queued. If you exceed this, the oldest message is lost. Each message holds a text string and a function code.

See the documentation of individual talkers to see if they support this feature and to see what messages and codes are available.

The command has three variants:

Get the number of pending messages

```
Func TalkerReadStr(name$);
```

`name$` The name of the talker, which must be connected. You can see the currently connected talkers from the Sample menu Talkers command or by using the `TalkerInfo()` command.

Returns The number of queued messages from the talker waiting to be read or a negative error code.

Read the oldest message and remove it from the queue

```
Func TalkerReadStr(name$, &msg$);
```

`name$` The name of the talker.

`msg$` A string variable to hold the text associated with the message. The text will be no more than 255 ASCII characters.

Returns A positive function code associated with the message or a negative error code.

Discard all pending messages, get cache size

```
Func TalkerReadStr(name$, opt%);
```

`name$` The name of the talker.

`opt%` Set 0 to flush the queued messages and return the number discarded.

Set to 1 to return the maximum number of queued messages or 0 if the talker is not revision 2 or not connected.

Returns The number of messages that were discarded or the maximum queue size or a negative error code.

See also:
`SampleTalker(), TalkerSendStr()`

TalkerSendStr()

This function transmits a text string and a function code to a named talker. You could use this to configure a talker before sampling (but not in a way that would cause any data it transferred to be incompatible with the target channel) or to control the talker during sampling. The original talker specification (revision 1), does not support this function. Your talker must support at least talker interface specification revision 2 to support this command. You can get information about known talkers from the Sample menu Talker list command.

```
Func TalkerSendStr(name$, code%, msg$);
```

`name$` The name of the talker, which must be connected. You can see the currently connected talkers from the Sample menu Talkers command.

`code%` A positive 32-bit code to send; the meaning is determined by the talker. If your talker only uses strings to communicate, we recommend that you set this to 0.

`msg$` A text string to be sent to the talker; the meaning of the text is determined by the talker. If your talker only uses codes to communicate, we recommend that you set this to `""`. In the future we will very likely use UTF-8 coded strings, but for now we recommend that you restrict yourself to ASCII characters with codes in the range 0 to 127.

Returns 0 is returned if the message has been passed to the talker for transmission as soon as possible. 1 if the talker is busy sending the previous message; you should try again later. Otherwise a negative error code is returned (unknown talker, not connected, timed out, talker version less than 2 so it does not accept strings).

See the documentation of individual talkers to see if they support this feature and to see what commands are supported.

See also:
`SampleTalker(), TalkerReadStr()`

Tan()

This calculates the tangent of an angle in radians or converts an array of angles into tangents. Tangents of odd multiples of pi/2 are infinite, so cause computational overflow. There are 2pi radians in 360degrees. pi is approximately3.14159265359 (`4.0*ATan(1))`.

```
Func Tan(x|x[]{[]...});
```

x The angle, expressed in radians, or a real array of angles. The best accuracy of the result is obtained when the angle is in the range -2pi to 2pi.

Returns For an array, it returns a negative error code (for overflow) or 0. When the argument is not an array the function returns the tangent of the angle.

See also:
Abs(), ATan(), Cos(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Trunc()

Tanh()

This calculates the hyperbolic tangent of a value or an array of values.

```
Func Tanh(x|x[]{[]...});
```

x The value or an array of real values.

Returns For an array, it returns 0. Otherwise it returns the hyperbolic tangent of x.

See also:
Abs(), ATan(), Cos(), Cosh(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sinh(), Sqrt(), Trunc()

Time$()

This function returns the current system time of day as a string. If no arguments are supplied, the returned string shows hours, minutes and seconds in a format determined by the operating system settings. To obtain the time as numbers, use TimeDate(). To obtain relative time and fractions of a second, use Seconds(). Default argument values are shown **bold**.

```
Func Time$({tBase%{, show%{, amPm%{, sep$}}}});
```

tBase% Specifies the time base to show the time in. You can choose between 24 hour or 12 hour clock mode. If this argument is omitted, 0 is used.

0 Operating system settings 2 12 hour format
1 24 hour format

show% Specifies the time fields to show. Add the values of the required options together and use that as the argument. If this argument is omitted or a value of 0 is used, **7** (1+2+4) is used for 24 hour format and **15** (1+2+4+8) for 12 hour format.

1 Show hours 4 Show seconds 1 Show milliseconds (new at [10.0]) 6
2 Show minutes 8 Remove leading zeros from hours

amPm% This sets the position of the "AM" or "PM" string in 12 hour format and has no effect in 24 hour format. If omitted, a value of zero is used. The string that gets printed ("AM" or "PM") is specified by the operating system.

0 Operating system settings 2 Show to the left of the time
1 Show to the right of the time 3 Hide the "AM" or "PM" string

sep$ This string appears between adjacent time fields. If sep$ = ":" then the time will appear as 12:04:45. If an empty string is entered or sep$ is omitted, the operating system settings are used.

See also:
Date$(), FileTime$(), Seconds(), TimeDate()

TimeDate()

This returns the time and date in seconds, minutes, hours, days, months, and years plus the day of the week. You can use separate variables for each field or an integer array. To get the data or time as a string, use `Date$()` or `Time$()`. To measure relative times, or times to a fraction of a second, see the `Seconds()` command. To get the current sampling time, see `MaxTime()`.

```
Proc TimeDate(&s%{, &m%{, &h%{, &d%{, &mon%{, &y%{, &wDay%}}}}}});
Proc TimeDate(now%[])
```

`s%` If `s%` is the only argument, it is set to the number of seconds since midnight. Otherwise it is set to the number of seconds since the start of the current minute.

`m%` If this is the last argument, it is set to the number of minutes since midnight. Otherwise it is set to the number of full minutes since the start of the hour.

`h%` If present, the number of hours since Midnight is returned in this variable.

`d%` If present, the day of the month is returned as an integer in the range 1 to 31.

`mon%` If present, the month number is returned as an integer in the range 1 to 12.

`y%` If present, the year number is returned here. It will be an integer such as 2002.

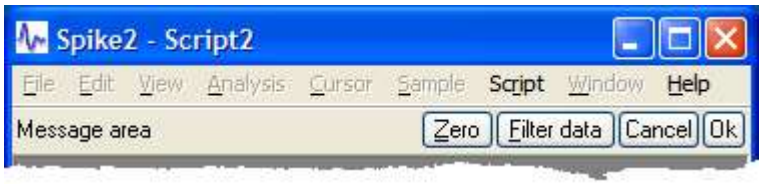
`wDay%` If present, the day of the week will be returned here as 0=Monday to 6=Sunday.

`now%[]` If an array is the first and only argument, the first seven elements are filled with time and date data. The array can be less than seven elements long. Element 0 is set to the seconds, 1 to the minutes, 2 to the hours, and so on. From version [10.0], if the array is eight elements long, `td%[7]` is set to the number of milliseconds, when available.

See also:
`Date$(), MaxTime(), Seconds(), Time$(), FileTimeDate()`

The toolbar

The toolbar is at the top of the screen, below the menu. The bar has a message area and can hold buttons that are used in the `Interact()` and `Toolbar()` commands. When you start a script, the toolbar is invisible and contains no buttons. When a script stops running, the toolbar becomes invisible (if it was visible). Do not confuse this with the system toolbar.



You can define up to 40 visible buttons, numbered from 1, in your toolbar. There is an invisible button 0, which sets an *idle* function that is called while the toolbar waits for a button to be pressed.

Buttons can be linked to the keyboard. However, any keys linked to buttons belong to the toolbar when it is active and waiting for a button press. If you link the A key to a button, each time you press A, the button is pressed, even if the text caret is in a text window.

Since version [10.06], we attempt to restore the Input focus to the view that held it at the time of the click. This restoration happens after any linked function runs. See `FocusHandle(-1)` for a way to stop the restoration.

You can also link mouse movement and mouse button presses in Time, Result and XY views to script functions and control the displayed mouse pointer.

See also:
`Interact(), MousePointer(), Toolbar(), ToolbarClear(), ToolbarEnable(), ToolbarMouse(), ToolbarSet(), ToolbarText(), ToolbarVisible()`

Toolbar()

This function displays the toolbar and waits for the user to click a button or press a linked key or perform a linked mouse action. If button 0 is defined with an associated *Idle* function, that function is called repeatedly while no button is pressed. If no buttons are defined or enabled, or if all buttons become undefined or disabled, the toolbar state is illegal and an error is returned. If the toolbar was not visible, it becomes visible when this command is given.

When the Toolbar is active, Spike2 ignores the `Esc` key unless an "escape button" has been set by `ToolbarSet()`.

```
Func Toolbar(text$, allow% {,help%|help$});
```

`text$` A message to display in the message area of the toolbar. The message area competes with the button area. With many buttons, the text may not be visible.

`allow%` A code that defines what the user can do (apart from pressing toolbar buttons). The values are the same as for `Interact()`.

1	0x0001	Can change application
2	0x0002	Can change the current window
4	0x0004	Can move and resize windows
8	0x0008	Can use the File menu
16	0x0010	Can use the Edit menu
32	0x0020	Can use the View menu
64	0x0040	Can use the Analysis menu
128	0x0080	Can use the Cursor menu and add cursors
256	0x0100	Can use the Window menu
512	0x0200	Can use the Sample menu
1024	0x0400	No changes to y axis
2048	0x0800	No changes to x axis
4096	0x1000	No horizontal cursor channel change

`help` This is either the number of a help item (CED internal use) or it is a help context string. This is used to set the help information that is presented when the user presses the F1 key. Set 0 to accept the default help. Set a string as displayed in the Help Index to select a help topic, for example `"Cursors: Adding"`.

Returns The function returns the number of the button that was pressed to leave the toolbar, or a negative code returned by an associated button or mouse function. If the return is due to a mouse up user-defined function that returns 0, the return value will be greater than the number or any toolbar button.

The buttons are displayed in order of their item number. Undefined items leave a gap between the buttons. This effect can be used to group related buttons together.

See also:
`Interact(), ToolbarClear(), ToolbarEnable(), ToolbarMouse(), ToolbarSet(), ToolbarText(), ToolbarVisible()`

ToolbarClear()

This function removes some or all of the toolbar buttons. If you delete all buttons, the `Toolbar()` function inserts an OK button, so you can get out of the `Toolbar()` function. Use `ToolbarText("")` to clear the toolbar message.

```
Proc ToolbarClear({item%});
```

`item%` If present, this is the button to clear. Buttons are numbered from 0. If omitted, all buttons are cleared. If the toolbar is visible, changes are shown immediately.

See also:
`Interact(), Toolbar(), ToolbarEnable(), ToolbarSet(), ToolbarText(), ToolbarVisible()`

ToolbarEnable()

This function enables and disables toolbar buttons, and reports on the state of a button. Enabling an undefined button has no effect. If you disable all the buttons and then use the `Toolbar()` function, or if you disable all the buttons in a function linked to the toolbar, and there is no idle function set, a single OK button is displayed. You can use the return value to detect if a button is defined and, from [10.05], if an Idle function is set.

```
Func ToolbarEnable(but% {,state%});
Func ToolbarEnable(const but%[], state%); (new at 8.03)
```

`but%` The number of the button or -1 for all buttons. Button 0 is always 'enabled' if an Idle function is set; you must enable and disable button 0 with `ToolbarSet()` and `ToolbarClear()`.

`but%[]` An array of button numbers. This allows you to set the state of several buttons with a single call. If you use this array variant of the command, you must provide the `state%` argument.

`state%` If present this sets the button state. 0 disables a button, 1 enables it, -1 leaves the state unchanged.

Returns The function returns the state of the button (or the first button in the list in the array variant of the command) prior to the call, as 0 for disabled and 1 for enabled. If `but%` or `but%[0]` was -1, the function returns 0 before [10.05] and the count of enabled buttons from [10.05]. If an undefined button is selected, the function returns -1. Before [10.05], if button 0 was specified this always returned 0, now it returns -1 if no function is set and 1 if one is.

See also:
`Interact(), Toolbar(), ToolbarClear(), ToolbarSet(), ToolbarText(), ToolbarVisible()`

ToolbarMouse()

This command gives you access to the mouse positions and left button mouse clicks in Time, Result and XY views when the mouse is over a data channel while a toolbar is active. There is an example, here. The description of this command applies also to `DlgMouse()` when used with 5 or more arguments.

```
Proc ToolbarMouse(vh%, ch%, mask%, want%, Func Down%{, Func Up%{, Func
Move%}});
```

`vh%` This is either the view handle of the view that you want to get mouse information from, or 0, meaning that you will accept mouse information from any suitable view. If you set this value, you will only get `Down%()` calls for this view. `Up%()` and `Move%()` calls do not depend on `vh%` matching.

`ch%` This is either a channel number in the view that you want mouse information for when you click on it or 0 to accept input from any channel. In an XY view, the display area is treated as belonging to channel 1, so setting 1 or 0 will work.

If you set a channel, you will only get `Down%()` calls for this channel. `Up%()` and `Move%()` calls do not depend on `ch%` matching.

If you set a channel number, once you have clicked on that channel, all values passed to you will be for that channel, even if you drag the mouse over a different channel. If you want to be able to click on a channel, then drag to another and be told about the other channel you must set `ch%` to 0. With `ch%` set to 0, if you drag to a place where there is no channel, you will be returned the last position that was over a channel.

`mask%` This, and the next argument (`want%`) are used when the left mouse button is clicked to decide if the script should be told about the mouse click. When the mouse button is clicked, and the conditions set here are met, the mouse becomes *owned* by the script and all mouse input will be given to the script until the mouse is released (or another application grabs the mouse). The mouse is said to be *captured*. The conditions set here are also used to decide if the script should be informed of mouse movements when the mouse is not owned by the script. Both `mask%` and `want%` are the sum of a set of values:

1	The left-hand mouse button is down.
2	The right-hand mouse button is down (releasing this button will normally display a context menu)
4	The `Shift` key is down.
8	The `Ctrl` key is down
16	The mouse middle button is down
32	Extra button 1 is down (this is the left-hand side button on my mouse)
64	Extra button 2 is down (this is the right-hand side button on my mouse)
128	The `Alt` key is down
256	There was a left mouse button double-click

The `mask%` value determines which of these items we care about. For example if you cared about the state of the `Shift` and `Ctrl` keys, you would set the value to 12.

`want%` This argument sets the desired state of the items that you have identified with `mask%`. For example, if you only want to be told when the `Shift` key is down and the `Ctrl` key is not down, set `mask%` to 12 and `want%` to 4. Another use would be to stop the script being told when the mouse was just being moved around but had not been clicked in an area we wanted. In this case you would set `mask%` to 1 and `want%` to 1 (only tell me when the left-hand mouse button is down).

`Down%` This is the name of a user-defined function that is called when the mouse left-hand button is clicked and the conditions implied by `vh%`, `ch%`, `mask%` and `want%` are satisfied. The arguments and return value of this function are described below.

`Up%` This is the name of an optional user-defined function that is called when the mouse button is released after it has been captured. You will always get a `Down%` function call before you get an `Up%` call. If another application (rudely) takes over the mouse by popping up a window, you will also get a call to the `Up%` function. This function is described below.

`Move%` This is the name of an optional user-defined function that is called when the mouse is moved after being captured. This function is also called when the mouse is moved when not captured by a mouse click and the conditions set by `mask%` and `want%` are satisfied (`vh%` and `ch%` are set to the actual view and channel). If you only want to be called during a drag operation, make sure you include the value for the left-hand mouse button in both `mask%` and `want%`. To be told about all mouse moves set both `mask%` and `want%` to 0.

The user-defined mouse functions

All three functions have exactly the same arguments. The function names do not have to be `Down%`, `Up%` and `Move%`, you can choose any names that are suitable. Ideally your mouse functions (especially mouse move) should not take a long time to run; if they do, the mouse movement will feel uncomfortable and jerky. If you must trigger a more time-consuming operation, set a flag and then service it in an idle routine. The mouse pointer and any screen drawing related to the mouse is not acted upon until after your function returns, so you may get odd screen effects if you have a time-consuming function that causes screen repainting. The return value has different uses in all three cases. The functions are:

```
Func Down%(vh%, ch%, x, y, flags%);
Func Move%(vh%, ch%, x, y, flags%);
Func Up%(vh%, ch%, x, y, flags%);
```

`vh%` The view handle of the view that the mouse is over. For `Down%()` and `Up%()`, if you set `vh%` to a view handle value in the `ToolbarMouse()` call, then this will be that value. Your function must not close this view as the view is required to handle the return value from this function. If you attempt to close the view with `FileClose()`, the call will report an unclosed view.

`ch%` The channel number that the values of `x` and `y` relate to or 0 if none. For `Down%()` and `Up%()`, if you specified a channel number in the `ToolbarMouse()` call, then this will be that value.

`x` The x-axis value in x-axis units in the view identified by `vh%`. If you click and drag you can get values that are outside the visible range of the x axis. If you want to scroll the view in response to this, you can do so.

`y` The y-axis value in y axis units for the channel identified by `ch%`. If there is no y axis or channel, the value will be 0.

`flags%` This holds the same information as held by `mask%` and `want%`. It gives you the state of the mouse buttons and `Shift`, `Ctrl` and `Alt` keys.

Return value

The return values have different uses for the three functions:

`Func Down%()`

If you decide that you do not want to do anything with the mouse, for example the click was not over anything interesting, then return 0 and Spike2 will decide what to do with the click. You will never get a mouse down call when the mouse is over the XY view key, or over a vertical or horizontal cursor. However, you do get priority over Spike2 for all other clicks in a view (for sizing, for instance). Return values greater that 0 select the mouse pointer to display (this is covered below) and mean that you want to capture the mouse for script use.

You can also choose to display an indication of a selection size or area by adding a value to the return value (only 1 value can be added). If you add any of the following values and you drag beyond the left or right edges of the data area, the area will scroll (if it is allowed to).

Value	Result
256	Display a selection rectangle, as you would for zooming in and out. If you return 256, the appropriate zoom cursor (16 or 17) is selected based on the state of the `Ctrl` key).
512	Display a measurement of the distance between the start of the drag and the current position. If you return 512, the measurement cursor (19) is selected.
1024	Display a line from the selection start to the current position. If you return 1024, the measurement cursor is selected.

`Func Move%()`

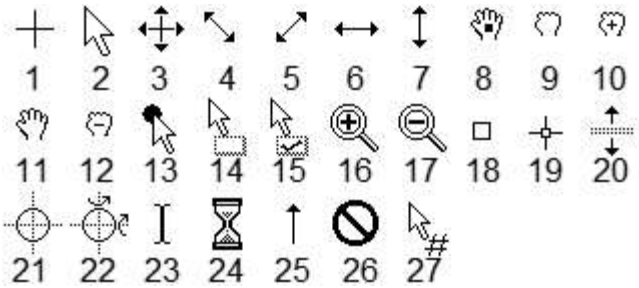
The return value from this sets the mouse pointer to be used (see below). A return value of 0 makes no change to the mouse pointer, which means that the pointer set by `Func Down%()` will be used for a drag operation and the cross-hair cursor (#1 in the list below) used when not dragging.

`Func Up%()`

The return value from this determines if the toolbar (or user-defined dialog for `DlgMouse()`) closes or not when the mouse button is released. Return a negative number or 0 to close the toolbar or dialog, 1 to keep running. With a dialog, a negative return acts as Cancel and 0 as OK. If the returned value closes the toolbar or dialog, the result of `Toolbar()` or `DlgShow()` is the negative returned value or a positive number greater than any button number if 0 was returned.

Mouse pointers set by return values

You have access to many of the mouse pointers that are available in Spike2, so you can use these to indicate that you are over an item or to show that you are dragging something. The values for the preset mouse pointers are:



Points 1-7 and 23-26 are system mouse pointers and may have a different appearance if you have chosen a custom set of mouse pointers in the Windows system Control Panel in the Mouse section. You should be aware that 3D and animated mouse pointers can be a lot slower on some systems than simple monochrome pointers. You can also define your own mouse pointers with the `MousePointer()` command. These are assigned numbers above the range of the built-in cursors.

Mouse double-click behaviour

If you double-click, you will get a mouse down, possibly followed by one or more mouse moves, followed by a mouse up for the first click. The second click generates another mouse down, but this time the double-click flag

will be set. It is up to you to decide if you want to undo anything that happened as a result of the initial mouse down and up. If your mouse down function returns 0, the double click will be handled by Spike2.

Things you should not do in a user-defined mouse function

The code in the mouse functions runs as part of the Windows code that decides which mouse pointer to display. There are a few things that you must not do inside the mouse functions:

1. Calling `FileClose()` on the view that is handling the mouse move or button notification will not close the view. This is because if it did close the view, when the mouse function returned control back to the deleted window, the program would crash.
2. Do not use up a lot of time, especially in a mouse move routine. Time consuming mouse routines will make mouse movement and button clicks feel awkward.
3. Mouse move calls after the mouse has been captured should not display user-defined dialogs or use the `Message()` or `Input()` or `Input$()` commands. If you do use these functions, you will find that the mouse is not available as it belongs to the window where the mouse was clicked. The keyboard can be used to navigate the dialog, but users will be very confused and may think that the program has crashed as mouse clicks in the dialog will have no effect.

Debugging user-defined mouse functions

You can set a break point in the mouse down, move and up functions. However, if you do, the function will not control the mouse pointer (as it is in use for debugging) and the mouse up function will not terminate the `Toolbar()` (or `DlgShow()`) call if it returns 0.

See also:

`DlgMouse()`, `MousePointer()`, `Toolbar()`, `ToolbarMouse()` example

ToolbarMouse() Example

This example assumes that there is a time, result or XY view open. The three mouse routines just print information to the log window. The `ToolbarMouse()` call is set up to accept any suitable window and any channel in that window. The `mask%` and `want%` arguments are set so that the mouse left-hand button must be down. This prevents the mouse move function being called unless the mouse is down.

The mouse down function requests mouse pointer 19 (the measurement pointer) and adds 1024, which causes a line to be drawn between the mouse down position and the current position.

The mouse move function returns 0, which will leave the mouse pointer unchanged. If there was no need to do anything in this routine it could be omitted. We have included it so we can print the mouse information to the log window.

The mouse up function returns 1, so that the toolbar continues to run.

```
func MouseDown%(vh%, chan%, x, y, flags%)
PrintLog("Down: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 19+1024; 'cursor 19 + a line linking start to end
end;

func MouseMove%(vh%, chan%, x, y, flags%)
PrintLog("Move: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 0;       'keep same cursor as for the mouse down
end;

func MouseUp%(vh%, chan%, x, y, flags%)
PrintLog("Up: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 1;       'do not close the toolbar
end;

ToolbarSet(1, "Hello");
ToolbarMouse(0, 0, 1, 1, MouseDown%, MouseUp%, MouseMove%);
Toolbar("Hello", 511);
```

ToolbarSet()

This function adds a button to the toolbar and optionally associates a call-back function with it. When a button is added, it is added in the enabled state; you can use `ToolbarEnable()` to change the state. Note that button 0 is not displayed; it defines an Idle time routine that runs when the toolbar is active and the system has nothing else to do. From version [7.01] you can also use the `ToolbarSet()` command with no arguments while the toolbar is displayed to get the last button pressed. There are two command variants:

```
Func ToolbarSet(item%, label$ {,func ff%()});
Func ToolbarSet();
```

`item%` The button number in the range 1 to 40 to add or replace or 0 to set or clear a function that is called repeatedly while the toolbar waits for a button press (an *Idle* function).

You can set an "escape" key as described in `Toolbar()`, by negating `item%`. For example `ToolbarSet(-2,"Quit");` sets button 2 as the escape key.

`label$` The button label plus an optional virtual key code and tool tip as `"Label|code|tip"`. Labels compete for space with each other; use tool tips for lengthy explanations. The label is ignored for button 0 (used to define an Idle function). Tool tips can be up to 79 characters long. To use a tool tip with no virtual key code use `"Label||A tooltip with no code field"` or set the code to 0. The label can contain any Unicode character supported by the Windows system font. You cannot include a vertical bar in the button label.

To link a key to a button, place `&` before a character in the label or add a vertical bar and a virtual key code in hexadecimal (e.g. 0x30), octal (e.g. 060) or decimal (e.g. 48) to the end of the label. Characters set by `&` are case insensitive. For example `"a&Maze"` generates the label aMaze and responds to `m` or `M`; the label `"F1:Go|0x70"` generates the label F1:Go and responds to the `F1` key. Useful virtual key codes include (`nk` = numeric keypad, `us` = US keyboard and some others).

#	Use	#	Use	#	Use	#	Use	#	Use	#	Use
0x08	Backspace	0x22	Page down	0x28	Down arr	0x6b	nk +	0xba	us ;:	0xc0	us ~
0x09	Tab	0x23	End	0x2e	Del	0x6c	nk ,	0xbb	=+	0xc1	us [{
0x0d	Enter	0x24	Home	0x30-39	0-9	0x6d	nk -	0xbc	,<	0xc2	us \ |
0x1b	Escape	0x25	Left arrow	0x41-5a	A-Z	0x6e	nk .	0xbd	-_	0xc3	us] }
0x20	Spacebar	0x26	Up arrow	0x60-69	nk 0-9	0x6f	nk /	0xbe	.>	0xc4	us ' "
0x21	Page up	0x27	Right arr	0x6a	nk *	0x70-87	F1-F24	0xbf	us /?		

Virtual key codes are not ASCII codes (though A-Z, 0-9 and a few more have the same codes); they are codes for the physical keys. Key with different shifted versions are shown with both items, but be aware that the combinations vary by keyboard type (for example on my UK keyboard, the single quote is paired with `@`, not `"`). If you use a keyboard that is radically different (for example a Japanese keyboard without the A-Z symbols) you will have access to much the same codes, but you need to experiment to find where they are on your keyboard.

Use of other virtual key codes or use of `&` before characters other than a-z, A-Z or 0-9 may cause unpredictable effects. Code `0x6c` appears on some keypads (e.g. Swedish) as the decimal separator (in place of `0x6e`). Code `0x0d` is also the keypad Enter code. You can find the full list of virtual key codes by searching the web for "virtual key codes windows".

Beware: when the toolbar is active, it grabs all keys linked to it. If the A virtual key code is linked, you cannot type a or A into a text window with the toolbar active.

ff%() This is the name of a function with no arguments. The name with no brackets is given, for example ToolbarSet(1,"Go",DoIt%); where Func DoIt%() is defined somewhere in the script. When the Toolbar() function is used and the user clicks on the button, the linked function runs. If the item% 0 function is set, that function runs while no button is pressed and is called the Idle function. The function return value controls the action of Toolbar() after a button is pressed.

If it returns 0, the Toolbar() function returns to the caller, passing back the button number. If it returns a negative number, the Toolbar() call returns the negative number. If it returns a number greater than 0, the Toolbar() function does not return, but waits for the next button. An item 0 function must return a value greater than 0, otherwise Toolbar() will return immediately.

If this argument is omitted, there is no function linked to the button. When the user clicks on the button, the Toolbar() function returns the button number.

Returns The call with no arguments returns the item number of the last button pressed. This call can be used to service multiple buttons with a single user-defined function. The call that creates a button returns 0.

Interaction with keyboard focus

When you click on a button, the normal Windows action is to give the button the keyboard focus. In Spike2, this can be a nuisance, especially when sampling when you may wish to have keystrokes recorded. To work around this, the Toolbar (and Interact bar) restore the keyboard focus. However, if the call-back function deliberately sets a new front view, you will usually want the keyboard focus to move and not be restored. From [10.14] you can use FocusHandle(-1) in the call-back function to stop the restoration.

See also:

Asc(), DlgButton(), Interact(), Toolbar(), ToolbarClear(), ToolbarEnable(), ToolbarText(), ToolbarVisible()

ToolbarText()

This can be used when a toolbar is active after a Toolbar() command, or when it is not. There are two command variants:

```
Func ToolbarText(msg${, allow%});
```

This replaces any message in the toolbar and if the toolbar is not active, makes the toolbar visible if it is invisible. It can be used to give a progress report on the state of a script that takes a while to run.

msg$ A string to be displayed in the message area of the toolbar.

allow% This is the same as the allow% argument to the Toolbar() command. It allows you to modify the allowed actions when the Toolbar() command is running, usually from a function linked to a toolbar button. If used when the toolbar is not active it will have no effect. This was added at revision [10.21].

Returns The bar state at the time of the call: 0 for hidden, 1 for visible. There was no return value before revision [10.21].

```
Func ToolbarText({show%{, &msg$});
```

This variant (added at revision [10.21]) can be used to show or hide the toolbar and to read back the text displayed in the bar. Note that you cannot hide the bar when it is active.

show% Set -1 for no change, 0 to hide, 1 to show the bar. This has no effect if the bar is active as it is always visible.

msg$ A string variable that is returned holding the current bar text.

Returns The bar state at the time of the call: 0 for hidden, 1 for visible.

See also:

Interact(), Toolbar(), ToolbarClear(), ToolbarEnable(), ToolbarSet(), ToolbarVisible()

ToolbarVisible()

This function reports on the visibility of the script toolbar, and can also show and hide it. You cannot hide the toolbar if the `Toolbar()` function is in use.

```
Func ToolbarVisible({show%});
```

`show%` If present and non-zero, the toolbar is made visible. If this is zero, and the `Toolbar()` function is not active, the toolbar is made invisible.

Returns The state of the toolbar at the time of the call. The state is returned as 2 if the toolbar is active, 1 if it is visible but inactive and 0 if it is invisible.

See also:
`Interact(), Toolbar(), ToolbarClear(), ToolbarEnable(), ToolbarSet(), ToolbarText()`

Trim()

This function removes leading and trailing white space (spaces, tabs and end of line characters) or user-defined characters from a string variable.

```
Proc Trim(&text${, chars$};
```

`text$` The string variable to remove characters from.

`chars$` An optional list of characters to remove. If omitted, `" \t\n\r"` is used.

This function and the similar `TrimLeft()` and `TrimRight()` are commonly used to help parse user input that may contain multiple spaces. For example:

Input to `Trim()`	After `Trim()`	After `Trim(text$, "1234 ");`
`"  12AB34  "`	`"12AB34"`	`"AB"`
`"  1234  "`	`"1234"`	`""`

See also:
`DelStr$(), InStr(), Left$(), Len(), Mid$(), Right$(), TrimLeft(), TrimRight()`

TrimLeft()

This function removes leading white space (spaces, tabs and end of line characters) or user-defined characters from a string variable.

```
Proc TrimLeft(&text${, chars$};
```

`text$` The string variable to remove characters from.

`chars$` An optional list of characters to remove. If omitted, `" \t\n\r"` is used.

This function and the similar `Trim()` and `TrimRight()` are commonly used to help parse user input that may contain multiple spaces. For example:

Input to `TrimLeft()`	After `TrimLeft()`	After `TrimLeft(text$, "1234 ");`
`"  12AB34  "`	`"12AB34  "`	`"AB34  "`
`"  1234  "`	`"1234  "`	`""`

See also:
`DelStr$(), InStr(), Left$(), Len(), Mid$(), Right$(), Trim(), TrimRight()`

TrimRight()

This function removes trailing white space (spaces, tabs and end of line characters) or user-defined characters from a string variable.

```
Proc TrimRight(&text${, chars$};
```

`text$` The string variable to remove characters from.

`chars$` An optional list of characters to remove. If omitted, `" \t\n\r"` is used.

This function and the similar `TrimLeft()` and `Trim()` are commonly used to help parse user input that may contain multiple spaces. For example:

Input to `TrimRight()`	After `TrimRight()`	After `TrimRight(text$, "1234 ");`
`"  12AB34  "`	`"  12AB34"`	`"  12AB"`
`"  1234  "`	`"  1234"`	`""`

See also:
`DelStr$()`, `InStr()`, `Left$()`, `Len()`, `Mid$()`, `Right$()`, `TrimLeft()`, `Trim()`

Trunc()

Removes the fractional part of a real number or array. To truncate a real number to an integer, assign the real to the integer. `ArrConst()` copies a real array to an integer array.

```
Func Trunc(x|x[]{[]...});
```

`x` A real number or a real array.

Returns 0 or a negative error code for an array. For a number it returns the value with the fractional part removed. `Trunc(4.7)` is `4.0`; `Trunc(-4.7)` is `-4.0`.

See also:
`Abs()`, `ATan()`, `Ceil()`, `Cos()`, `Exp()`, `Floor()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`

U

`U1401 commands`
`U1401Close()`
`U1401Ld()`
`U1401Open()`
`U1401Read()`
`U1401To1401()`
`U1401ToHost()`
`U1401Write()`
`UCase$()`

U1401 commands

The `U1401` commands give you direct access to a CED 1401 interface connected to your computer. Currently, the script language can have one 1401 open at a time. If you are sampling data with a 1401, the script language can open and use a *different* 1401 simultaneously. It is possible to communicate with the 1401 used for sampling; this requires detailed knowledge of the 1401 and Spike2 sampling process.

You must use the `U1401Open()` command first to take control of the 1401, and you should use the `U1401Close()` command to release the 1401 once you have finished with it. See the 1401 family programming manual for details of using the 1401; this is available from CED as part of the 1401 programming kit and also as a download from the CED web site (Downloads, 1401 Firmware, 1401 Windows installer).

See also:
U1401Close(), U1401Ld(), U1401Open(), U1401Read(), U1401To1401(), U1401ToHost(), U1401Write()

U1401Close()

This command closes the link between the script language and a 1401 interface generated by the U1401Open() command. If there is no open 1401, the command is ignored.

```
Proc U1401Close();
```

See also:
U1401Ld(), U1401Open(), U1401Read(), U1401To1401(), U1401ToHost(), U1401Write()

U1401Ld()

This command loads one or more 1401 commands with the option of nominating the folder to load the commands from. If no 1401 is open, the script halts.

```
Func U1401Ld(list$[, path$]);
```

list$ The list of commands to load separated by commas. Include KILL as the first item to clear all commands first. For example: "KILL,ADCMEM,MEMDAC". The file name for a command is the command name plus an extension that depends on the type of 1401. The extension is added automatically.

path$ Optional. The path to the folder to search for the commands. If omitted, or if the command is not found in this path, the 1401 folder in the Spike2 source folder is searched, then any path indicated by the 1401DIR environment variable, and finally, the \1401 folder on the current drive.

Returns 0 if all commands loaded. Otherwise the bottom 16 bits is a negative error code and the upper 16 bits is the 1-based index to the command in the list that failed to load. If v% is the non-zero return value, the command number is v%/65536 and the error code is (v% mod 65536 - 65536).

Note that loading a command resets the state of the 1401 and stops any interrupt driven processes.

See also:
U1401Close(), U1401Open(), U1401Read(), U1401To1401(), U1401ToHost(), U1401Write()

U1401Open()

This command attempts to open a 1401 for use by other U1401 commands and returns the type of the opened 1401 or a negative error code. It is not an error to call U1401Open() multiple times with no intervening U1401Close(), however, we suggest that you do not do this for future compatibility. Currently, if you do this, any unit number passed in is ignored and the script language continues using the 1401 that is already open.

```
Func U1401Open({unit%[, &got%]});
```

unit% Optional 1401 unit number(taken as 0 if omitted) in the range 1 to 8 or 0 for the first available unit. It is a fatal script error to request a unit number outside the allowed range.

CED only extension: negative values of unit% (-1 to -8) are reserved to provide access to a 1401 that is already open (for example, one in use for sampling). Without detailed knowledge of the sampling process, attempting to use this feature will damage data capture and could crash Spike2.

got% Optional argument to return the 1401 number that was opened. This argument was added at Spike2 [10.02].

Returns The return value is the type of the 1401 detected: 0=standard 1401, 1=1401*plus*, 2=micro1401, 3=Power1401, 4=Micro1401 mk II, 5=Power1401 mk II, 6=Micro1401-3, 7=Power1401-3 and 3A, 8=Micro1401-4. Otherwise it is a negative error code that can be decoded by Error$().

To identify a Power1401-3A when the return value of U1401Open() is 6, use the INFO,S,36 command after opening the 1401. For example:

```
var code%;
U1401Write("INFO,S,36;");
U1401Read(code%);
```

code% will have the value 6 for a Power1401-3A. INFO,S,36 is implemented for all Power1401-3A devices; other devices will return a different value (often 0 as this was a new feature introduced when we released the Power3A).

Programming the 1401

If you need to access the 1401 at a low level, you should read the 1401 programming documentation, in particular the *1401 family programming manual*, which is available as part of the 1401 Programming support from our CED web site, Downloads, 1401 Firmware. Click on *1401 Windows installer* to download the winsupp.exe file, which will install the support and documentation.

See also:

Error$(), U1401Close(), U1401Ld(), U1401Read(), U1401To1401(), U1401ToHost(), U1401Write()

U1401Read()

This command reads a text response from a 1401 and optionally converts it into one or more integer values or reports the number of available input lines. If no 1401 is open, the script halts. There are four variants:

Func U1401Read();	Get count of input lines
Func U1401Read(&text$);	Read an input line as text
Func U1401Read(&v1%{, &v2%, {...}});	Read a line and convert to integers
Func U1401Read(arr%[]);	Read a line, convert to integer array

text$ A text variable returned holding the entire response.

v1% An integer variable that is returned with the first integer number read.

vn% Optional integer variables (v2% up to v12%) returned with following values. Values for which no number is returned are unchanged.

arr%[] An integer array that is filled (starting at element 0) with converted values.

Returns The version with no arguments returns the number of available input lines. The other versions return the number of items that were converted from the input text and stored to a script variable or a negative error code. If you use a variant that reads a line and there is no text to read, the command times out after about 3 seconds and returns an error code.

See also:

U1401Close(), U1401Ld(), U1401Open(), U1401To1401(), U1401ToHost(), U1401Write()

U1401To1401()

This command transfers the contents of an integer array to memory in the 1401.

```
Func U1401To1401(const arr%[]{[]}, addr%{, size%});
```

arr% This is a one or 2 dimensional array to transfer. If you use a 2 dimensional array to interleave 4 channels of data, for example for MEMDAC, set the first dimension to 4 and the second to the number of points per channel.

addr% The start address of the block of contiguous memory in the 1401 user area to be filled with data.

size% Optional, the number of bytes in the 1401 that each array element is copied to. Acceptable values are 1, 2, 4 or 8. If size% is omitted, 4 is used.

Returns 0 if the data transferred without a problem, or a negative error code.

See also:
U1401Close(), U1401Ld(), U1401Open(), U1401Read(), U1401ToHost(), U1401Write()

U1401ToHost()

This command transfers a block of 1401 memory into an integer array.

```
Func U1401ToHost(arr%[]{[]}, addr%{, size%});
```

arr% This is a one or 2 dimensional array to receive the data. If you use a 2 dimensional array to interleave 8 channels of data, for example for ADCMEM, set the first dimension to 8 and the second to the number of points per channel.

addr% The start address of the block of contiguous memory in the 1401 user area to copy data from.

size% Optional, taken as 4 if omitted. The number of bytes of 1401 data used to set each array element. Use 1, 2 or 4 to read 1, 2, 4 or 8 bytes and sign extend to 32-bit integer. Use -1, -2 or -4 to read 1 or 2 or 4 bytes and zero extend to 64-bit integer.

Returns 0 if the data transferred without a problem, or a negative error code.

See also:
U1401Close(), U1401Ld(), U1401Open(), U1401Read(), U1401To1401(), U1401Write()

U1401Write()

This command writes a text string to the 1401.

```
Func U1401Write(text$);
```

text$ The text to write to the 1401. Commands to the 1401 are terminated by either a newline "\n" or a semicolon ";".

Returns 0 if the line was added to the 1401 device driver output buffer, or a negative error code.

See also:
U1401Close(), U1401Ld(), U1401Open(), U1401Read(), U1401To1401(), U1401ToHost()

UCase$()

This function converts a string into upper case. The upper-case operation may be system dependent. Some systems may provide localised upper casing, others may only provide the minimum translation of the ASCII characters a-z to A-Z.

```
Func UCase$(text$);
```

text$ The string to convert.

Returns An upper case version of the original string.

See also:
Asc(), Chr$(), DelStr$(), InStr(), LCase$(), Left$(), Len(), Mid$(), Print$(), ReadStr(), Replace$(), Reverse$(), Right$(), Str$(), Val()

V

Val()
VerticalMark()
View()
ViewColour()
ViewColourGet()
ViewColourSet()
ViewExtraTime()

```
ViewFind()
ViewKind()
ViewLineNumbers()
ViewLink()
ViewList()
ViewMaxLines()
ViewOverdraw()
ViewOverdraw3D
ViewStandard()
ViewTrigger()
ViewUseColour()
ViewZoom()
VirtualChan()
```

Val()

This converts a string to a number. The converter allows the same number format as the script compiler and leading white space is ignored.

```
Func Val(text${, &nCh%{, flag%}});
```

`text$` A string that starts with a floating point number to convert. The conversion stops at the first character that is not part of the number. From version 6.10, the function will also accept a hexadecimal number if `flag%` is set to 1. In ambiguous cases, the conversion uses the format that uses the most characters of the input string, so `"0xa"` has the value 10, not 0 and uses all the characters. The string `"0x"` is converted as 0 and uses 1 character as `"0x"` is not a valid hexadecimal number. The expected formats are (items in curly brackets are optional, a vertical bar means use one of the characters before or after the bar):

`{white space}{-|+}{digits}{.digits}{e|E{+|-}digits}` or
`{white space}0x|Xhexadecimaldigits`

`nCh%` If present, it is set to the number of characters used to construct the number.

`flag%` If present and set to 1, hexadecimal input is also acceptable.

Returns It returns the extracted number, or zero if no number was present. You should use the `nCh%` argument to decide if a number was processed.

See also:
`Asc()`, `Chr$()`, `DelStr$()`, `InStr()`, `LCase$()`, `Left$()`, `Len()`, `Mid$()`, `Print$()`, `ReadStr()`, `Replace$()`, `Reverse$()`, `Right$()`, `Str$()`, `UCase$()`

VerticalMark()

This sets up, cancels or gets information about vertical markers in the current time view. The View menu Vertical Marker command documentation has more information. Use the `FontGet()` and `FontSet()` commands to set the text font.

```
Func VerticalMark(chan%{, flags%{, just%{, hPos%{, dir%}}}});
```

`chan%` Either the channel number of an event, Marker or extended marker type to use or 0 to cancel any vertical marker or -1 to -5 to return the current values of arguments 1 (channel) to 5 (direction).

`flags%` Set this to -1 or omit for no change. Otherwise it is the sum of:

1	Draw on top of channel data
2	Use Marker code colours
4	Enable text for TextMark channels
8	No vertical line for text
16	No fill behind text

`just%` Sets the text justification as: 0=Left, 1=Centre, 2=Right. Use -1 for no change.

`hPos%` Sets the text baseline: 0=Below, 1=On, 2=Above the line. Use -1 for no change.

`dir%` Sets the text direction: 0=Upwards, 1=Downwards. Use -1 for no change.

Returns Calls with `chan%` greater than or equal to 0 return 0 for success or a negative error code. Calls with `chan%` less than 0 return the requested information.

See also:
`FontGet()`, `FontSet()`, Vertical markers

View()

The `View()` function sets the current view and returns the last view handle. A view handle is a positive integer > 0. Changing the current view does not change the focus or bring the view to the front, use `FrontView()` to do that.

```
Func View({vh%});
```

`vh%` An integer argument being:

- >0 A valid view handle of a view that is to be made the current view. Use `ViewKind()` to test for a valid handle.
- 0 (or omitted) no change of the current view is required.
- <0 If `vh%` is `-n`, this selects the `n` th duplicate of the time view that is associated with the current view. The current view can be a time, result or XY view. If the current view is a time view, this is equivalent to `Dup(n).View(-1)` in a result or XY view returns the time view from which it was created.

Returns the view handle of the view that was current at the time of the call. If an argument is passed in which is not valid, the script stops with an error.

View().x()

The `View().x()` construction overrides the current view for the evaluation of `x()`. For example, `View(vh%).Draw(1,2)` draws view number `vh%`. It is an error if the selected view does not exist or if the function closes the original view, and the script stops.

```
View(vh%).x()
```

`vh%` A view handle of an existing view, 0 for the current view, or -n for the n th duplicate of the time view associated with the current view.

The equivalent code to `View(vh%).x()` is:

```
var temp%;
temp% := View(vh%);  'Save the current view
x();                 'call the user-defined or built-in function
View(temp%);         'restore the original view
```

This means that `View(vh%).FileClose()` causes an error if `vh%` is the current view.

View().[]

The `View().[]` construction overrides the current view to give you access to the data arrays that form result view channels.

```
View(vh%{,ch%}).[]
```

`vh%` A view handle of an existing result view or 0 for the current result view.

`ch%` An optional channel number in the result view. If omitted, channel 1 is assumed.

For example:

```
ArrConst(View(0,2).[6:20], 0); 'zero 20 elements of channel 2
View(rv%).[4] := 3;            'set fifth element of channel 1
```

See also:
`App()`, `Dup()`, `FileClose()`, `FrontView()`, `SampleHandle()`, `ViewFind()`, `ViewKind()`, `ViewLink()`, `ViewList()`

ViewColour()

Deprecated. Use `ViewColourGet()` and `ViewColourSet()`. This function gets and sets the palette colour indices of time, result and XY view items, overriding the application-wide colours set by `Colour()`. Currently you can set the background colour.

```
Func ViewColour(item%{, col%});
```

`item%` The colour item to get or set; 0=background

`col%` If present, the new colour index for the item. There are 40 colours in the palette with indices 0 to 39. Use -1 to revert to the application colour for the item.

Returns The palette colour index that is nearest to the item colour at the time of the call or -1 if no view colour is set. **Beware**: if you set a colour index for an item and read it back you may not get the index you set if another index holds the same colour.

See also:
Colour dialog, `ChanColour(), Colour(), PaletteGet(), PaletteSet(), XYColour()`

ViewColourGet()

This function gets the RGB colour of a time, result or XY view item, indicating if it overrides the application-wide colours set by `ColourSet()`. This was added at Spike2 version 7.07.

```
Func ViewColourGet(item%{, &r, &g, &b});
```

`item%` The colour item to get or set; 0=background. See `ViewColourSet()` for other values.

`r g b` If present, set to the red, green and blue colour values in the range 0.0 to 1.0. If the colour is not overridden, these values will be 0.

Returns 1 if the returned colour overrides the application colour, 0 if it does not.

See also:
Colour dialog, `ChanColourGet()`, `ColourGet()`, `ViewColourSet()`

ViewColourSet()

This function sets the RGB colour of time, result, XY and grid view and Info window items, to override the application-wide colours set by `ColourSet()`. This was added at Spike2 version 7.07. Note that setting any view colour other than 0 did not work before [10.05].

```
Proc ViewColourSet(item%{, r, g, b});
```

`item%` The colour item to get or set. The number of items depends on the current view type:

Time, Result and XY views

0 Background colour

1 All Info window backgrounds (each Info window can override)

2 All Info window text (each Info window can override)

Grid views and Info windows

0 Background colour

1 Text colour

`r g b` If present, These set the red, green and blue colour values in the range 0.0 to 1.0. If omitted, the item colour is set to the default application colour (see `ColourSet()`).

Colour heirarchy

When a view is created, it inherits the colour state from the Application colours. This command allows you to override those colours for the particular view. A newly-created Info window inherits the Info windows settings of the view it belongs to. If you make the Info window the current view, you can override the inherited settings.

See also:

Colour dialog, `ChanColourSet()`, `ColourSet()`, `ViewColourGet()`

ViewExtraTime()

This is equivalent to the View menu X Axis Extra Time menu command and sets the extra time to display at the end of the X axis during sampling. This has no effect on the `MaxTime()` command.

```
Func ViewExtraTime({secs));
```

`secs` If present and greater than or equal to 0.0, this sets the extra time to display. If omitted or negative, no change is made.

Returns The value of the extra time at the time of the call (before any change).

During sampling, this command can be used to set up a scrolling time window that displays future time, usually so that a pre-written channel is visible. This can be used in situations where a subject is required to track a template signal and needs to view the future state of the signal (often on the same axis as a recorded trace).

ViewFind()

This function searches for a window with a given title and returns its view handle. This is mainly used by the script recorder to locate views that were not created as part of the recorded sequence and usually indicates code that will require editing in a finished script. Windows titles can be modified to indicate read only status and duplicated window numbers or by a script; `ViewList()` can be a more reliable method of locating a view that displays a known data file.

```
Func ViewFind(title$${, flags%});
```

`title$` A string holding the view title to search for. The comparison is not case sensitive. If a view has been modified, the '*' added to the displayed view title is not included in the comparison.

`flags%` Optional, default of 0. The same flags as for `WindowTitle$()`, used to suppress view numbers, modified and 32-bit file markers from the title used for matching. This argument was added at version [10.13].

Returns The view handle of a view with a matching title, or 0 if no view matches.

See also:

`WindowTitle$(), View(), ViewList()`

ViewKind()

This function returns the type of the current view or a view identified by a view handle. Types 5-7 are reserved. Type 11 windows include the sampling configuration window and control panel, the sequencer control panel and multimedia windows.

```
Func ViewKind({vh%});
```

`vh%` An optional view handle. If omitted, the type of the current view is returned. Use `-n` in a result or XY view to find the n^{th} duplicate of the time view from which the current view was created.

Returns The type of the current view or the view identified by `vh%`. View types are:

-2	Invalid handle	1	Text view	4	Result view	10	Application	17	Grid view
-1	Unknown type	2	Output sequence	8	External text file	11	Other types	19	Info window
0	Time view	3	Script	9	External binary file	12	XY view		

Note: In Signal (a CED application with a similar script language), the code for an Info window is 18.

See also:
`ChanKind(), SampleHandle(), ViewList()`

ViewLineNumbers()

This function is used in a text-based view to display or hide line numbers and to set the number of decimal digits of space to use for a line number.

```
Func ViewLineNumbers({show%});
```

`show%` Set -1 or omit for no change, 0 to hide line numbers, 1 to show them. Use 2-8 to set 2-8 digits of space and greater than 8 for a standard display (5 digits).

Returns 0 if line numbers were hidden at the time of the call, else the number of digits of display space allocated (2-8).

See also:
`Gutter()`

ViewLink()

This function returns the view handle of the view that owns the current window. For example, you can use this to get the time view that owns a multimedia, spike shape or spike monitor window or the time view that created a result or XY view or the view that owns an Info window. This is slightly different from `View(-n)`, which finds the n^{th} duplicate of the time view linked to the current time, result or XY view.

```
Func ViewLink();
```

Returns The handle of the linked view, or 0 if there is no such view or if it is the Desktop window. For example, **`View(App(0)).ViewLink()`** returns 0 as we do not have a view handle for the Desktop window.

This command can also be used to iterate through the process operations that are linked to the current time view. For example, if you create a new sampling document using `FileNew()` with the current sampling configuration, you may also open other result and XY documents and have additional channels present due to the sampling configuration. In this case, the command is:

```
Func ViewLink(n%{, mask%{, &name$}});
```

`n%` This can be 0, meaning count the number of processes and return it, or it can be the process number to report on.

`mask%` This value determines the types of processes we wish to count or report on and is the sum of: 1 for result views generated by `SetINTH()` and similar calls, 2 for XY views generated by `MeasureToXY()` and 4 for data channels in the current view generated by `MeasureToChan()`. If this argument is omitted, it takes the value 3, to report on result and XY views. Normally this argument will be set to 1, 2, 3 or 4. There is nothing to stop you using the values 5, 6 and 7, but you would need to interpret the returned `name$` argument to decide if the return value was a channel number or a view handle.

`name$` If present, and `n%` is greater than 0, this is set to the name of the command that will process the data.

Returns If `n%` is 0, this returns the number of processes that match the `mask%` argument or a negative error code if the current view is the wrong type. If `n%` is greater than 0, then this returns either the handle of the view or the channel number in the current view that is the target of the process operation. If `n%` is greater than 0 and there is no corresponding process, the return value is 0.

Note that this form of the command identifies active processes, that is processes for which the `Process()` command could have some effect.

Example

This example code lists the processes associated with the current view:

```
var vh%, n%,i%,name$;
n% := ViewLink(0, 3);   'count associated result and XY views
for i% := 1 to n% do
    vh% := ViewLink(i%, 3, name$);
    PrintLog("View Handle %d, name: %s\n", vh%, name$);
next;

n% := ViewLink(0, 4);   'count associated channels
for i% := 1 to n% do
    vh% := ViewLink(i%, 4, name$);
    PrintLog("Channel %d, name: %s\n", vh%, name$);
next;
```

See also:

App(), MMOpen(), SampleHandle(), SSOpen(), View(), ViewKind(), ViewList()

ViewList()

This function fills an integer array with a list of view handles. It never returns the view handle of the running script; use the App() command to get this.

```
Func ViewList(list%[]{, types%});
```

list% An integer array that is returned holding view handles. The first element of the array (element 0) is filled with the number of handles returned. If the array is too small to hold the full number, the number that will fit are returned.

types% The types of view to include. This is a code that can be used to filter the view handles. The filter is formed by adding the types from the list below. If this is omitted or if no types are specified for inclusion, all view handles are returned.

1	Time views	8	Script views	256	External text	2048	Other view types
2	Text views	16	Result views	512	External binary	4096	XY views
4	Sequencer	128	Grid views	1024	Application view		

You can also exclude views otherwise included and include views normally excluded by adding:

8192	Exclude views not directly related to the current view
16384	Exclude visible windows
32768	Exclude hidden windows
65536	Exclude duplicates
131072	Include the running script and any #included files (these are usually hidden)

Returns The number of windows that match types%.

The following example prints all the window titles into the log view:

```
var list%[100],i%;          'Assume no more than 99 views
ViewList(list%[]);          'Get a list of all the views
for i%:=1 to list%[0] do    'Iterate round all the views
   PrintLog(view(list%[i%]).WindowTitle$()+"\n");
   next;
```

See also:

App(), SampleHandle(), ViewKind()

ViewMaxLines()

This function gets and optionally sets the maximum number of lines that the current text view will retain if text is added by a script. For the Log view, this value is initially set by the Edit menu Preferences in the General tab, but a script can set or clear the line limit for any text-based view. `ViewStandard()` clears the limit except in a Log view, where it sets the value set in the preferences.

```
Func ViewMaxLines({max%});
```

`max%` If present, this sets the maximum lines to keep after a script `Print()` or `PrintLog()` statement. If this is omitted or negative, no change is made. Set 0 for no limit and greater than zero to set a limit.

Returns The value of the limit at the time of the call.

Each time the line limit is exceeded, the number of lines in the file is reduced to 90% of the maximum line count by deleting the lowest numbered lines. We reduce the line count to 90% of the maximum because deleting lines from the start of the view is a slow operation (in very large views); by allowing the view to grow normally for a while this minimises the time penalty for setting this option. For example, if `max%` is set to 1000, the next time that a line is added and more than 1000 lines are in the file, the line count is reduced to 900. After this 100 lines can be added before the limit is tripped.

See also:
Edit Preferences, `ViewStandard()`

ViewOverdraw()

All time views hold a list of overdraw trigger times sorted into ascending order. The use of this list is enabled by `ViewTrigger()`. This command adds new times to the list if overdraw mode is enabled. The command can also read back the list of trigger times. Whenever this command changes the list, the view is positioned to the last list time. It is the script equivalent of the View menu Overdraw List dialog. Trigger times are taken from a data channel, or can be supplied as times in seconds. There are two command variants:

```
Func ViewOverdraw(flag%, chan%, sTime, eTime);
Func ViewOverdraw({flag%{, time|times[]}});
```

`flag%` If -1 and the `times[]` array is present, times are read back from the list. Otherwise, it is the sum of the following values:

- 1 Empty the list before optionally adding any new values. If omitted, previous events are preserved unless adding new events causes them to be deleted.
- 2 Adding a time does not delete all later times in the list (Merge mode). If omitted (Normal mode), adding a time deletes all later times; the last time added becomes the current time.

`chan%` The channel number of an event, Marker, or Marker-derived channel in the time view to use as a source of trigger times.`ViewTrigger(-1)` returns the channel that the times in the list come from or -1 if there is a mix of channels.

`sTime` The start of the time range to search for times.

`eTime` The end of the time range to search for times.

`time` A single time to be added to the list.

`times` An array of times to be added to the list (`flag%` >= 0) or to hold times copied from the list (`flag%`=-1). Times are added in array order, so the times should be in ascending order if `flag%` is not 2 or 3. `ViewTrigger(-1)` will return -1 as there is no associated source channel.

Returns If `flag%` is -1 and `times[]` is present, the return value is the number of elements of `times[]` that were returned holding times from the list. Otherwise, it is the number of times in the list or -1 if overdrawing is disabled.

See also:
Overdraw list details, `ViewTrigger()`, `ViewOverdraw3D()`, Overdraw List dialog, Display Trigger dialog, Overdraw 3D dialog

ViewOverdraw3D()

This command controls 3D drawing (enabled by the `ViewTrigger()` command or the Display Trigger dialog). It is the equivalent of the View menu Overdraw 3D dialog. This command may only be used in a Time view. There are two command variants. The first sets the overdraw values, the second reads back the current settings.

```
Func ViewOverdraw3D(xProp, yProp{, xScale{, yScale{, flags%}}});
Func ViewOverdraw3D(get%)
```

`xProp` The proportion of the available x space (in the range 0 to 1.0) to use for the 3D effect. Values outside the range 0 to 1 are limited to this range.

`yProp` The proportion of the available y space to use for the 3D effect. Values outside the range 0 to 1 are limited to this range.

`xScale` This optional argument sets how much to shrink the display width when going from the front to the back to give a perspective effect. Values are limited to the range 0 (shrink to nothing) to 1 (no shrink). If omitted, no change is made.

`yScale` How much to shrink the display height when going from the front to the back to give a perspective effect, in the range 0 to 1. Values are limited to the range 0 (shrink to nothing) to 1 (no shrink). If omitted, there is no change.

`flags%` If this optional argument is omitted, no change is made. It the sum of:

- 1 Display a fixed count of frames or time range. This only has any effect if a maximum frame count or time range is set by `ViewTrigger()`. If not set, the latest frame is at the front and the oldest is at the back.
- 2 The position of a frame is determined by the frame time. If omitted, the position of a frame is set by the frame number.
- 4 Auto update on a change in the dialog (new at 9.04).

`get%` The variant of the command with one argument uses this value to indicate the value to read back: -1=`xProp`, -2=`yProp`, -3=`xScale`, -4=`yScale`, -5=`flags%`

Returns 0 when setting a value or the value requested by `get%`.

See also:
`ViewTrigger()`, `ViewOverdraw()`, Display Trigger dialog, Overdraw List dialog, Overdraw 3D dialog

ViewStandard()

This sets the current time, result or XY view to a standard state by making all channels and axes visible in their standard drawing mode, axis range and colour. All channels are given standard spacing and are ungrouped and the channels are sorted into the numerical order set by the Edit menu Preferences. In a time view, duplicate channels are deleted, triggered mode is disabled and any channel processing is removed and any extra time at the end of the x axis is removed. All XY, Sonogram and colour scale keys are deleted.

In a text-based view it removes any maximum line limit except in the Log view, where is applies the line limit set in the Edit menu Preferences option. It also hides line numbers, displays the gutter and the folding margin (for views that support folding). All the text styles are set to the default values (equivalent to opening the Font dialog and using Reset All) and any zooming is removed.

In a Grid view it removes any applied formatting.

```
Proc ViewStandard();
```

See also:
`ChanOrder()`, `ChanWeight()`, `DrawMode()`, `Gutter()`, `ViewMaxLines()`, `ViewTrigger()`, `XYDrawMode()`

ViewTrigger()

This controls the triggered drawing mode available for time views; it is a fatal error to use this command in a different view type. This command is the equivalent of the View menu Display Trigger dialog. See that for a detailed description of the arguments. The first command variant sets and enables the view trigger and clears all memory of overdrawing. The second variant enables and disables the view trigger and gets information about the current settings.

```
Func ViewTrigger(chan%, pre, hold, xZero%{, cur0%{, wait{, over%{, col%{,
     maxO%{, maxT}}}}}});
Func ViewTrigger({mode%});
```

`chan%` The event, marker, WaveMark, TextMark or RealMark trigger channel. Set this to 0 for a paged display on-line or during rerun or to step by one display page width on Next and Previous buttons.

`pre` The pre-trigger time to display, in seconds.

`hold` The minimum hold time for on-line displays, the minimum time to the next/previous trigger event for off-line use.

`xZero%` If this is set to 1, the x axis zero point (for display only) is moved to the trigger time. All time measurements are still relative to the start of the file.

`cur0%` This optional argument controls the cursor 0 action each time the view is triggered. 0=no action, 1=move cursor 0 without causing the active cursors to iterate, 2=move cursor 0 and update any active cursor positions (after a delay of `wait` seconds if this is an online trigger due to sampling a new trigger point). Set -1 for no change. If this argument is omitted, the value 0 is used.

`wait` This optional argument sets how long to wait online after a trigger before updating any active cursors when `cur0%` is 2. This allows a search that would hit the end of a file if done immediately the trigger value was detected. If this argument is omitted, the value 0 is used (for backwards compatibility). Set `wait` to -1 for no change to the current value. This argument was added at version 6.08.

`over%` Set 1 to enable overdraw, add 2 to enable 3D drawing, 0 to disable, -1 for no change. If omitted, 0 is used.

`col%` Overdrawn data colour. -1 or omitted=no change, 0=normal, 1=half intensity, 2=fade to background, 3=fade to secondary.

`maxO%` Maximum overdrawn traces from 1 to 4000, 0 for no limit or -1 or omitted for no change.

`maxT` Maximum time range of overdrawn traces up to 10000 seconds or 0 for no limit or -1 or omitted for no change.

`mode%` This command version gets information, moves to the next and previous trigger, and enables and disables the trigger. With no arguments, the command returns the enabled or disabled state as 1 or 0. If there is a single argument, it can be:

- 4 Get the x axis position displayed as "0.0". This is the current trigger position. Added at version [11.00]. This returns 0.0 if the view is not in triggered mode, or no trigger is set.
- 3 Move to the next trigger and return the trigger time or -1 if no trigger is found or if the trigger is not enabled.
- 2 Move to the previous trigger and return the trigger time or -1 if no trigger is found or if the trigger is not enabled.
- 1 Enable the View trigger with the current settings and return the previous trigger enable state.
- 0 Disable the trigger and return the previous trigger enable state.

Negative values return the current argument values:

-1	`chan%`	-3	`hold`	-5	`cur0%`	-7	`over%`	-9	`maxO%`
-2	`pre`	-4	`xZero%`	-6	`wait`	-8	`col%`	-10	`maxT`

Returns When the command is used with `mode%`, the command returns the requested information or -1 if the information is not available. All other use returns the enabled/disabled state as 1 or 0, or a negative error code.

See also:
Display Trigger dialog, Overdraw details, `DrawMode(), ViewOverdraw(), ViewOverdraw3D(), ViewStandard(), XYDrawMode()`

ViewUseColour()

This function can be used to force the display to use black and white only, or to use the colours set in the Colour dialog or by the `Colour()` command.

```
Func ViewUseColour({use%});
```

`use%` If present, a value of 0 forces Spike2 to display all windows in black and white. Any other value allows the use of colour. If omitted, no change is made.

Returns The current state as 1 if colour is in use, 0 if black and white is used.

See also:
`ChanColour(), Colour(), ViewColour(), XYColour()`

ViewZoom()

This function is used in a text-based view to get and optionally set the zoom factor. This is the number of points to add to the nominal size of the text.

```
Func ViewZoom({zoom%});
```

`zoom%` Omit for no change or a value in the range -10 to 20 to add to the point size of all the text. The resulting minimum text size is 2 point regardless of `zoom%`.

Returns The zoom value at the time of the call.

See also:
`ViewStandard()`, Zooming

VirtualChan()

This command controls virtual channels in the current time view. Virtual channels are defined by an algebraic expression that can include other channels (but not virtual channels with the same or a higher channel number). This command has the functionality of the Virtual Channel dialog. There are two command variants, described below. It is not a fatal error to set a channel number that is in the possible range of virtual channels and that does not exist; this causes the command to return an error code. However, trying to use a channel number outside the virtual channel range (apart from 0 or -1 in the first variant) is a fatal error and will stop the script.

Create a new virtual channel or modify an existing channel

Depending on the value of the channel number, this command variant either creates a new virtual channel and returns the number, or it modifies an existing virtual channel.

```
Func VirtualChan(chan%, expr$[, match%[, binsz[, align]]]);
```

`chan%` This is either -1 or 0 to create a new virtual channel, or the channel number of an existing virtual channel to modify it. If `chan%` is 0, the created channel has the lowest available virtual channel number. If `chan%` is -1, the created channel is set to the channel number of the highest existing virtual channel plus 1. If the highest allowed channel is already in use, the lowest unused channel is returned. You can use the `Chan("v1")` command to find the channel number of the first virtual channel. Using -1 is a new feature added at Spike2 version 9.02.

In the first command variant, set this to 0 to create a new virtual channel at the lowest available channel or to -1 to create at and return the channel number. The remaining arguments set the initial channel settings, otherwise default values are used. If not 0, this is the number of an existing virtual channel to modify or from which to read back the settings.

`expr$` This is a string expression defining the virtual channel expression that creates the channel output. An illegal expression does not prevent the channel being created, but no data will be displayed.

`match%` This is the number of an existing waveform, RealWave or WaveMark channel to match for sample interval and alignment. If 0, the sample interval and alignment are set by the `binsz` and `align` arguments. If negative, no change is made.

`binsz` If `match%` is 0, this sets the sample interval of the channel. Values of `binsz` less than the file time resolution are ignored, as are values less than or equal to 0.

`align` If `match%` is 0, this sets the channel alignment. Values less than 0 are ignored.

Returns When creating a new channel, the return value is the new channel number or a negative error code. When modifying an existing channel, the return value is a negative code if the `match%`, `binsz` or `align` arguments are illegal, a positive code (below) if the expression contains an error and 0 if there was no error.

Read back virtual channel information

You can use this variant to read back the state of an existing virtual channel.

```
Func VirtualChan(chan%, get%{, &expr$});
```

`chan%` This is the channel number of an existing virtual channel from which you want to read back information.

`get%` This determines the returned value. 0 = the state of the expression parsing, 1 = `match%`, 2 = `binsz`, 3 = `align`.

`expr$` This is an optional string variable that is returned holding the current expression for the channel.

Returns The requested information or a negative error code. If `get%` is 0, the return value is negative if the channel is not a virtual channel, 0 if the expression is acceptable and a positive code (below) if not.

Error codes if expression is invalid

The current error codes when the virtual channel expression is invalid (at version [10.20]) are:

Code	Meaning
1	Expected a closing bracket
2	Expected `,` or `)` when parsing argument list
3	Item number out of range
4	Did not recognise a field as a number or a value
5	Nothing found
6	Division by constant of 0
7	Bad channel number
8	No channel in the expression
9	Problem building the channel list
10	Attempted square root of negative number
11	Logarithm of negative or zero number
12	Out of range result
13	Expression is empty
14	Expected a value, got a channel
15	Value is incorrect
16	Not enough arguments
17	Internal logic error or error reading data
18	Generated wave had too short a time
19	Argument was not a vector
20	The cursor number was illegal
21	Wanted a value, got a vector

See also:

More about virtual channels, `MemChan(), Chan()`

W

```
Window()
WindowDuplicate()
WindowGetPos()
WindowSize()
WindowTitle$()
WindowVisible()
```

Window()

This sets the position and size of the current view. Normally, positions are percentages from the top left-hand corner of the application window size. You can also set positions relative to a monitor. This can also be used to position, dock and float dockable toolbars.

```
Func Window(xLow, yLow{, xHigh{, yHigh{, scr%{, rel%}}}});
```

`xLow` Position of the left hand edge of the window in percent. When docking a dockable toolbar, the `xLow` and `xHigh` values correspond to the position of the top left corner of the window when dropped with the mouse.

`yLow` Position of the top edge of the window in percent.

`xHigh` If present, the right hand edge. If omitted the previous width is maintained. If the window is made too small, the minimum allowed size is used. If the current view is a dockable control and `yHigh` is 0, values less than 1 or greater than 4 float the window at `(xLow, yLow)`, otherwise `xHigh` sets the docking state:

1 Docked to the left window edge
2 Docked to the top window edge
3 Docked to the right edge
4 Docked to the bottom edge

`yHigh` If present, the bottom edge position. If omitted the previous height is maintained. If the window is made too small, the minimum allowed size is used.

If the Window is dockable and `yHigh` is 0, this command sets the docked state of the window (see `xHigh`). Otherwise the window is floated with the nearest allowed width that is no more than `xHigh-xLow`. If `xHigh-xLow` is 0 or negative `YHigh` sets the height of the dockable window.

`scr%` Optional screen selector for views, dialogs and the application window. If omitted or -1, positions are relative to the application window. Otherwise, 0 selects the entire desktop rectangle and greater values select a particular screen rectangle (but see `rel%`). Add 1000 to `scr%` to get a screen rectangle with areas reserved by the system removed. See `System()` for more screen information.

`rel%` Ignored unless `scr%` is 0 or greater. Set 0 or omit for positions relative to the intersection of the rectangle set by `scr%` and the application window, 1 for positions relative to the `scr%` rectangle. If there is no intersection, there is no position change. When positioning the application window, `rel%` is treated as if it were 1.

Returns 1 if the position was valid, or -1 if the rectangle set by `scr%` and `rel%` is of zero size. Before Spike2 version 6.06 this was a `Proc` with no return value.

Examples:

```
View(App()).Window(0,0,100,100,0);  'Spike2 uses all desktop
View(App()).Window(0,0,100,100,2);  'Spike2 uses all second monitor
Window(0,0,100,100);                'Current view uses all application space
View(SSOpen(0)).Window(0,0,50,50);  'position a spike shape window
```

See also:

`App()`, `System()`, `WindowDuplicate()`, `WindowGetPos()`, `WindowSize()`, `WindowTitle$()`, `WindowVisible()`

WindowDuplicate()

This duplicates the current time window, creating a new window that has all the settings of the current window. It does not duplicate channels; these are shared with the existing window. The new window becomes the current view and is created invisibly.

```
Func WindowDuplicate();
```

Returns The view handle of the new window, a negative error code or 0 if there are no free duplicates. There is a limit of 64 duplicates per window.

See also:
`Window()`, `WindowGetPos()`, `WindowSize()`, `WindowTitle$()`, `WindowVisible()`

WindowGetPos()

This gets the window position of the current view with respect to the application window, the desktop, a particular display screen or the intersection of the application window and one of these rectangles. Positions are measured from the top left-hand corner of the reference rectangle as a percentage. The `scr%` and `rel%` arguments were added at version 6.06. To get the position of a user dialog use the `DlgGetPos()` command.

```
Func WindowGetPos(&xLow, &yLow{ ,&xHigh ,&yHigh{, scr%{, rel%}});
```

`xLow` A real variable that is set to the position of the left hand edge of the window.

`yLow` A real variable that is set to the position of the top edge of the window.

`xHigh` A real variable that is set to the position of the right hand edge of the window or that returns a docking code for a docked control bar if `yHigh` is returned as 0.

1 Docked to the left window edge
2 Docked to the top window edge
3 Docked to the right edge
4 Docked to the bottom edge

`yHigh` A real variable that is set to the position of the bottom edge of the window or to 0 if the window is docked.

`scr%` Optional screen selector for views, dialogs and the application window. If omitted or -1, positions are relative to the application window. Otherwise, 0 selects the entire desktop rectangle and greater values select a particular monitor (but see `rel%`). See `System()` for more screen information.

`rel%` Ignored unless `scr%` is 0 or greater. Set 0 or omit for positions relative to the intersection of the rectangle set by `scr%` and the application window, 1 for positions relative to the `scr%` rectangle.

Returns 1 if the position was valid, or -1 if the rectangle set by `scr%` and `rel%` is of zero size. Before Spike2 version 6.06 this was a `Proc` with no return value.

Note
We observe that top level windows (windows that can be positioned outside the application window) can have a semi-transparent border that is used to generate a shadow. The `x,y` position that is returned allows for any such border, so the returned values may appear a little smaller than you would expect

.

See also:
`DlgGetPos()`, `System()`, `Window()`, `WindowDuplicate()`, `WindowSize()`, `WindowTitle$()`, `WindowVisible()`

WindowSize()

This resizes the current view without changing the top left-hand corner position. There are no errors from this function. There are two command variants:

Set window size

Setting a negative size causes no change. Setting a size less than the minimum or greater than the maximum allowed sets the appropriate limit.

```
Proc WindowSize(width, height);
```

`width` The width of the window as a percentage of the available area. Negative values preserve the current width.

`height` The height of the window as a percentage of the available area. Negative values preserve the current height.

You can also use this to resize the application window or a dockable control bar when it is floating; use `App()` to get the handles. For control bars, if `width` is greater than zero, its sets the width, otherwise `height` is used to set the height. If the control bar can be resized, it will use the width or the height and will calculate the other dimension itself.

Size to contents ([10.05] onwards)

In a grid view, using the command with no arguments sizes the view so that there is no unused space and so that the bottom right cell is visible, if possible. This does not change the top left cell of the grid. The application window determines the maximum size of the grid view.

```
Proc WindowSize();
```

This command variant has no effect on other view types.

See also:

`App(), Window(), WindowDuplicate(), WindowGetPos(), WindowTitle$(), WindowVisible()`

WindowTitle$()

This function gets and sets (where allowed) the current window title. Most windows can return a title. If you change a title, dependent window titles change, for example, cursor windows belonging to time views track the title of the time view. You can set the titles of time, result, XY, grid and text-based views interactively with the Window menu Window Title command.

```
Func WindowTitle$({new$|flags%});
```

`new$` If present, this sets the new window title. Window titles must follow any system rules for length or content. Illegal titles (for example titles containing control characters) are mangled or ignored at the discretion of the system. If you set an empty string as the title to a time, result, XY, grid or text-based view, the title will revert to the original default title.

`flags%` This argument was added at version [10.09]. Used to modify the returned string. If omitted, taken as 0, otherwise the sum of:

- 1 Add `" *"` to the end of the title if the associated view data has been modified.
- 2 Suppress the addition of `":n"`, where `n` is the view number, when the view is duplicated.
- 4 Suppress `"[32-bit]"` at the end for `.smr` data files. New at version [10.13].

Returns The window title as it was prior to this call. Beware that this returned value includes duplicate numbers (unless `flags%` is set to suppress then) if the window is duplicated, for example `"fred.smrx:1"`. If you set this as a title, the window title will become: `"fred.smrx:1:1"`. This happens because the duplicate number is not really part of the title and is added just before the window is displayed.

The Window Title is *not* the same as the name of any associated file. For example, when you open a data file that has no title set in the associated resource file, the window title is set based on the file name, but can be set

independently of it. If you set the window title, it is saved in the resource file and restored when the resource file is used.

See also:
Window(), WindowDuplicate(), WindowGetPos(), WindowSize(), WindowVisible()

WindowVisible()

This function is used to get and set the visible state of the current window. This function can also be used on the application window, however the effect will vary with the operating system.

```
Func WindowVisible({code%});
```

code% If present, this sets the window state. The possible states are:

- 0 Hidden, the window becomes invisible. A hidden window can be sent data, sized and so on, the result is just not visible.
- 1 Normal, the window assumes its last normal size and position and is made visible if it was invisible or iconized.
- 2 Iconized, an iconized window can be sent data, sized and so on; the result is not visible. This will dock a dockable window at its last docked position.
- 3 Maximise, make it as large as possible or float a dockable window.
- 4 Application window only; extend over all available desktop monitors.

Returns The window state prior to this call. A hidden application window may return 2, not 0.

See also:
FrontView(), Window(), WindowDuplicate(), WindowGetPos(), WindowSize(), WindowTitle$()

X

XAxis()
XAxisAttrib()
XAxisMode()
XAxisStyle()
XHigh()
XLow()
XRange()
XScroller()
XTitle$()
XToBin()
XUnits$()
XYAddData()
XYColour()
XYCount()
XYDelete()
XYDrawMode()
XYGetData()
XYInChan()
XYInCircle()
XYInRect()
XYJoin()
XYKey()
XYOffset()
XYRange()
XYSetChan()
XYSetData()
XYSize()
XYSort()

XAxis()

This turns on and off the x axis of the current view and returns the state of the x axis.

```
Func XAxis({on%});
```

`on%` Set the axis state. If omitted, no change is made. 0=Hide axis, 1=Show axis.

Returns The axis state at the time of the call (0 or 1, as above) or a negative error code. It is an error to use this function on a view that has no concept of an x axis.

Changes made by this function do not cause a redraw immediately. The affected view is drawn at the next opportunity.

See also:
`XAxisMode(), XAxisStyle(), XHigh(), XLow(), XRange(), YAxis()`

XAxisAttrib()

This function controls the choice of logarithmic or linear axis in result and XY views and control over the automatic adjustment of axis units in time, result and XY views. This command is equivalent to the controls at the bottom of the X Axis dialog. You cannot set a logarithmic axis in a time view.

```
Func XAxisAttrib({flags%});
```

`flags%` A value of 0 sets a linear axis with no auto-adjust of units for high zoom. Add 1 for logarithmic. Add 2 to display powers on the logarithmic axis (you must have added 1 as well for this to take effect). Add 4 to cause a linear axis to auto-adjust its units in increments of 10^3 at high zoom around 0, add 8 (in addition to 4) to indicate scale changes by using SI prefixes on the units. Omit this argument for no change to the attributes.

Returns The sum of the current flags set for the x-axis.

See also:
`XAxis(), XAxisMode(), XAxisStyle(), XHigh(), XLow(), XRange(), YAxisAttrib()`

XAxisMode()

This function controls what is drawn in an x axis. This command matches some of the controls in the Show/Hide Channel dialog. See the `Grid()` command for placing an x axis in the data area.

```
Func XAxisMode({mode%});
```

`mode%` Optional argument that controls how the axis is displayed. If omitted, no change is made. Possible values are the sum of the following:

1	Hide all the title information.
2	Hide all the unit information.
4	Hide small ticks on the x axis. Small ticks are hidden if big ticks are hidden.
8	Hide numbers on the x axis. Numbers are hidden if big ticks are hidden.
16	Hide the big ticks and the horizontal line that joins them.
32	Scale bar axis. If selected, add 4 to remove the end caps.

Returns The x axis mode value at the time of the call or a negative error code.

See also:
`XAxis(), XAxisAttrib(), XAxisStyle(), XHigh(), XLow(), XRange(), YAxisMode()`

XAxisStyle()

This function controls the x axis display style and the major and minor tick spacing for all views that have an x axis. Tick spacing values that would cause illegible or unintelligible axes are stored but not used unless the axis range or scaling changes to make the values useful. This command matches controls in the X Axis Range dialog.

```
Func XAxisStyle({style%{, nTick%{, major}}});
```

`style%` In a time view, set 1 for an axis in seconds, 2 for hours minutes and seconds, 3 for time of day and 4 for milliseconds. In a result or XY view set 1 for the standard axis. If the view has units of "s" or "seconds", you can also set 4 for a milliseconds axis. Omit `style%` or use 0 to leave the style unchanged. Set -1 to return the minor tick subdivisions value, -2 to return the major tick spacing value.

`nTick%` The number of minor tick subdivisions or 0 for automatic spacing. Omit `nTick%` or set it to -1 for no change.

`major` If present, values greater than 0 sets the major tick spacing. Values less than or equal to 0 select automatic major tick spacing.

Returns If `style%` is positive or omitted it returns the style at the time of the call. See the description of `style%` for negative values.

See also:
`XAxis(), XAxisAttrib(), XAxisMode(), XHigh(), XLow(), XRange(), YAxisStyle()`

XHigh()

This returns the x axis value at the right hand side of the current time, result or XY view, the end of the time range in the associated time view to process for a spike shape view, the line past the last fully visible line in a text view or the right-most visible column in a grid view. Use with other views causes a fatal script error.

```
Func XHigh()
```

Returns In a time view or a spike shape dialog, the result is in seconds. In a result view, the result is in bins and can be fractional. In an XY view the result is in x axis units. It is in lines in a text view.

In a grid view it is the 0-based column number of the column that intersects with the right-hand edge of the view. If there is grey space to the right of the grid, from [10.05] the return value is the number of columns (previously it was 0).

In a text view, the value is the first visible line number plus the number of fully visible lines. At the end of a file, the returned value can be greater than the number of lines in the file.

This example pages through a time or result view from the current position to the end.

```
while (xHigh() < MaxTime()) do Draw(XHigh()) wend;
```

See also:
`Draw(), XRange(), BinToX(), XToBin(), XLow()`

XLow()

This returns the x axis value at the left hand side of the current time, result or XY view, the start of the time range in the associated time view to process for a spike shape view, the first visible line in a text view or the left-most visible column in a grid view. It is a fatal error to use this in an inappropriate view.

```
Func XLow()
```

Returns In a time view or a spike shape dialog, the result is in seconds. In a result view, the result is in bins and can be fractional. In a text view, this returns a line number (the first line number in a text view is 1). In a grid view it returns the left-most visible column number.

For example, this code pages a time or result view from the current position to the start of the file. In an XY view it moves the view to the left if the current position is past 0.

```
while XLow()>0 do Draw(XLow()-(XHigh()-XLow())) wend;
```

See also:
`Draw(), XRange(), BinToX(), XToBin(), XHigh()`

XRange()

This sets the x axis range to display in a time result or XY view in x axis units (not bins for a result view). In a Grid view, it sets the first column to display. It can also be used as the equivalent of the X Axis Range dialog Show All button. Unlike `Draw()`, the view does not update immediately; updates wait for the next `Draw()`, `DrawAll()`, `Yield()` or some interactive activity.

In the Create and Edit WaveMark spike shape dialogs it sets the time range in the associated time view to process.

Set X axis range or position or Grid view column

```
Proc XRange(from {,to});
```

`from` The left hand edge of the view in x axis units (seconds for a time view, 0-based columns for a grid view). You can set `from` to -1 in a time view that is sampling or re-running to make the view scroll automatically to show the most recent data at the right-hand edge.

`to` The right hand edge of the view. If omitted, the view stays the same width.

Values are limited to the axis range for time and result views; there is no limit in an XY view. Without `to`, it preserves the width, adjusting `from` if required. If the resulting width is less than the minimum allowed, no change is made.

Show all X axis

From Spike2 version 9.02, the command has a new variant that is equivalent to the X Axis Range dialog Show All button.

```
Proc XRange()
```

In time and result views, this expands the x axis to display the full data range. In an XY view, this sets the x axis to display the range of all the visible data channels; it does not change the y axis to make all the points visible.

Time view in Triggered mode

If the view is in Triggered mode with a trigger set, the x axis displays times relative to the trigger, but the operation of this command is unchanged. If you want to display an x axis that starts at the displayed value of -1.0, from [11.00] you can use `XRange(ViewTrigger(4) - 1.0);`

See also:
`Draw(), XLow(), XHigh(), Yield()`

XScroller()

This function gets and optionally sets the visibility of the x axis scroll bar and controls.

```
Func XScroller({show%});
```

`show%` If present, 0 hides the scroll bar and buttons, non-zero shows it.

Returns 0 if the scroll bar was hidden, 1 if it was visible.

See also:
`ChanShow()`, Show/Hide channel, `XAxisMode()`, `YAxisMode()`

XTitle$()

This gets and sets the x axis title in a result or XY view. In a time view, it has no effect and returns an empty string. The window updates with a new title at the next opportunity.

```
Func XTitle$({new$});
```

New$ If present, this sets the new x axis title in a result view.

Returns The x axis title at the time of the call.

See also:
ChanTitle$()

XToBin()

In a result view, this converts between bin numbers and x axis units; in a time view, it converts between time in seconds and the underlying Spike2 time units.

```
Func XToBin(x);
```

x An x axis value. If it exceeds the x axis range it is limited to the nearer end.

Returns In a result view it returns the bin position that corresponds to x. In a time view, it converts seconds to the underlying time units used for the file.

See also:
BinToX(), BinSize()

XUnits$()

This function gets the units of the x axis. You can also set the units in a result or XY view. The units of a Time view are always seconds. The window will update with the new units at the next opportunity.

```
Func XUnits$({new$});
```

New$ If present, this sets the new x axis units in a result view.

Returns The x axis units at the time of the call.

See also:
ChanUnits$(), XTitle$()

XY...()

XYAddData()
XYColour()
XYCount()
XYDelete()
XYDrawMode()
XYGetData()
XYInChan()
XYInCircle()
XYInRect()
XYJoin()
XYKey()
XYOffset()
XYRange()
XYSetChan()
XYSetData()
XYSize()
XYSort()
XYZOrder()

XYAddData()

This adds data points to an XY view channel. If the axes are set to automatic expanding mode by `XYDrawMode()`, they will change when you add a new data point that is out of the current axis range. If the channel is set to a fixed size (see `XYSize()`), adding new points causes older points to be deleted once the channel is full. The first form of the command allows unrestricted x and y positions. The second form is for data that is equally spaced in the x direction.

```
Func XYAddData(chan%, x, y);
Func XYAddData(chan%, const x{%}[], const y{%}[]);
Func XYAddData(chan%, const y[], xInc{, xOff});
```

`chan%` A channel number in the current XY view. The first channel is number 1.

`x` The x co-ordinate(s) of the added data point(s). In the first form of the command, both x and y must be single values to add one point. In the second version they are both real or integer vectors and the number of data points added is equal to the size of the smaller vector.

`y` The y co-ordinate(s) of the added data point(s). In the third form of the command, this is an array of equally spaced data in x.

`xInc` Sets the x spacing between the y data points in the second form of the command.

`xOff` Sets the x position of the first data point in the second form of the command. If omitted, the first position is set to 0.

Returns The number of data points which have been added successfully.

See also:
`XYColour()`, `XYCount()`, `XYDelete()`, `XYDrawMode()`, `XYGetData()`, `XYInCircle()`, `XYInRect()`, `XYJoin()`, `XYKey()`, `XYRange()`, `XYSetChan()`, `XYSetData()`, `XYSize()`, `XYSort()`, `XYZOrder()`

XYColour()

Deprecated. This gets or sets a channel line colour or a channel fill colour in the current XY view using a palette colour index.

```
Func XYColour(chan%{, col%{, item%}});
```

`chan%` A channel number in the current XY view. The first channel is number 1.

`col%` The index of the colour in the colour palette. There are 40 colours in the palette, numbered from 0 to 39. If omitted or -1, there is no colour change.

`item%` Set 1 for the line colour, 2 for the fill colour. Taken as 1 if omitted.

Returns The colour index in the colour palette of the nearest colour to the colour of the item at time of call or a negative error code.

This function is now deprecated. You should use `ChanColourSet()` and `ChanColourGet()` unless you need backward compatibility with older versions of Spike2.

See also:
Colour dialog, `ChanColour()`, `Colour()`, `ViewColour()`, `ViewUseColour()`, `XYAddData()`, `XYCount()`, `XYDelete()`, `XYDrawMode()`, `XYGetData()`, `XYInCircle()`, `XYInRect()`, `XYJoin()`, `XYKey()`, `XYRange()`, `XYSetChan()`, `XYSize()`, `XYSort()`, `XYZOrder()`

XYCount()

This gets the number of data points in a channel in the current XY view. To find the maximum number of data points, see the `XYSize()` command.

```
Func XYCount(chan%);
```

`chan%` A channel number in the current XY view. The first channel is number 1.

Returns The number of data points in the channel or a negative error code.

See also:
`XYAddData(), XYColour(), XYDelete(), XYJoin(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYKey(), XYRange(), XYSetChan(), XYSize(), XYSort(), XYZOrder()`

XYDelete()

This command deletes a range of data points or all data points from one channel of the current XY view. Use `ChanDelete()` to delete the entire channel.

```
Func XYDelete(chan% {,first% {,last%}});
```

`chan%` A channel number in the current XY view. The first channel is number 1.

`first%` The zero-based index of the first point to delete. Omit to delete all points.

`last%` The zero-based index of the last data point to delete. If omitted, data points from `first%` to the last point in the channel are deleted. If `last%` is less than `first%` no data points are deleted.

Returns The function returns the number of deleted data points.

The index number of a data point depends on the current sorting method of the channel set by `XYSort()`. For different sorting methods, a data point may have different index numbers. The data points in a channel have continuous index numbers. When a point has been deleted the remaining points re-index themselves automatically.

See also:
`ChanDelete(), XYAddData(), XYColour(), XYCount(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYKey(), XYRange(), XYSetChan(), XYSize(), XYSort(), XYZOrder()`

XYDrawMode()

This gets and sets the drawing and automatic axis expansion modes of a channel in the current XY view. It is an error to use this command with any other view type.

```
Func XYDrawMode(chan%, which% {,new%});
```

`chan%` A channel number in the current XY view. The first channel is number 1. This is ignored when `which%` is 5, as all XY channels share the same axes. -1 can also be used, meaning all channels.

`which%` The drawing parameter to get or set in the range 1 to 5. When setting parameters, the new value is held in the `new%` argument. The values are:

1 Get or set the data point draw mode. The drawing modes are:

0	dots (default)	3	crosses x	6	diamonds
1	boxes	4	circles O	7	horizontal line
2	plus signs +	5	triangles	8	vertical line

2 Get or set the size of the data points in points (units of approximately 0.353 mm). The sizes allowed are 0 to 100; 0 is invisible. The default size is 5.

3 Get or set the line style. If the line thickness is greater than 1 all lines are drawn as style 0. Styles are:

0	solid (default)	1	dotted	2	dashed

4 Get or set the line thickness in points. Thickness values range from 0 (invisible) to 10. The default is 1.

5 Get or set the XY Autoscale mode. This applies to the entire view, so the `chan%` argument is ignored. If `new%` is present it sets the new mode. Bits 0-3 select the data to use to set the range, bits 4-7 set the x axis scale more and bits 8-11 set the y axis scale mode.

Automatic scaling is done each time you add data to the XY view (but not when data is removed). The data in the XY view is divided into two classes: the new points and all visible data in the XY view. You can choose to auto scale based on new data in X or Y or on all visible data. Add one value from each table to generate the `new%` value:

#	Data selected for automatic scaling
0	None. The axes do not change automatically when new data points are added.
1	All visible points in the Y direction, only new points in the X direction.
2	All visible points in the X direction, only new points in the Y direction.
3	All visible and new data points.
4	New data points only.

#	How the X axis uses the data
0	Expand the x axis to make the selected data visible. No zoom in.
16	Track the selected data, zooming in or out as required to make the selected data fill the x axis range.
32	Fixed size x axis centred on the selected data.
48	Track the selected data, with a minimum size.
64	No automatic scale change to x axis

#	How the Y axis uses the data
0	Expand the y axis to make the selected data visible. No zoom in.
256	Track the selected data, zooming in or out as required to make the selected data fill the y axis range.
512	Fixed size y axis centred on the selected data.
768	Track the selected data, with a minimum size.
1024	No automatic scale change to x axis

The axis modes with a fixed or minimum size take the current axis size. If the user changes the axis size interactively or with a script command, the new size becomes the fixed size of limit.

Before Spike2 version 8.10, the only values for `new%` that could be used were 0 (no autoscale) and 1 (expand only). Note that if you are processing to an XY view with the Optimise display after process option enabled, it is more efficient to disable the XY Autoscale option.

`new%` New draw mode or axis expanding mode. If omitted, no change is made.

Returns The value of the relevant channel draw mode or axis expanding mode at the time of the call or a negative error code.

See also:
`XYAddData(), XYColour(), XYCount(), XYDelete(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYKey(), XYRange(), XYSetChan(), XYSize(), XYSort(), XYZOrder()`

XYGetData()

This gets data points between two indices from a channel in the current XY view. It is an error to use this command with any other view type.

```
Func XYGetData(chan%, &x|x[], &y|y[] {,first% {,last%}});
```

`chan%` A channel number in the current XY view. The first channel is number 1.

`x|x[]` The returned x co-ordinate(s) of data point(s). When arrays are used, either both `x` and `y` must be arrays or neither can be. The smaller of the two arrays sets the maximum number of data points that can be returned.

`y|y[]` The returned y co-ordinate(s) of data point(s).

`first%` The zero-based index of the first data point to return. If omitted, 0 is used.

`last%` The zero-based index of the last data point returned, used when `x` and `y` are arrays. If omitted or greater than or equal to the number of data points, the final data point is the last one in the channel. If `last%` is less than `first%`, no data points are returned.

Returns The number of data points copied. If the x or y arrays are not big enough to hold all the data points from `first%` to `last%`, the return value is the array size. If x or y are not arrays, if a data point with index `first%` exists, 1 is returned.

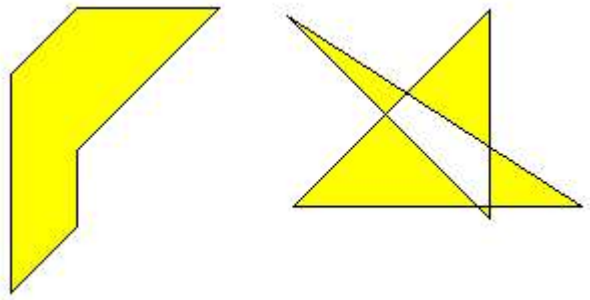
The index number of a data point depends on the current sorting method (see `XYSort()`).

See also:
`XYAddData(), XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYInCircle(), XYInRect(), XYJoin(), XYRange(), XYKey(), XYSetChan(), XYSize(), XYSort(), XYZOrder()`

XYInChan()

This function returns the number of data points in a channel of the current XY view that lie inside another channel that is treated as a joined up shape. The points in the channel that defines the shape are not sorted (regardless of the sorting mode set for that channel); they are always considered in the order in which the points were added to the channel. You might use this function to let a user draw a shape (using `ToolbarMouse()`) in an XY view around data points that they want to select for a particular purpose.

For a data point to lie inside the channel, we count the number of times that a line drawn from the data point to infinity in any direction crosses a line of the channel. If it crosses an even number of times it is outside (0 is even). If it crosses an odd number of times, it is inside. This is obvious for simple shapes, but less so for complex ones. In the example picture to the right, the shaded sections are the inside and the non-shaded are the outside. Points that lie exactly on the boundary may be inside or outside; however, if you have a set of shapes that exactly tessellate to fill an area, any point placed in that area will be in only one of the shapes. If there are `n` points to check, and the channel that defines the shape has `m` points, then this algorithm takes a worst case time of O(`n`*`m`). This means that it pays to search inside shapes with relatively few points to define them. The algorithm will be relatively quick if you are searching for points inside an area that is small compared to the area that the data points cover.



Inside and outside

```
Func XYInChan(chan%, shCh%{, list%[]});
```

`chan%` A channel number in the current XY view that defines the points to test against the shape.

`shCh%` The channel in the current XY view that defines the shape that the points must be inside.

`list%` An optional integer array that is returned holding the indices of the data points that were inside. If the array is too small, the function returns the number that would have been returned had the array been large enough. The index values returned are the index positions in the current sort mode for the channel. You can read the points back with `XYGetData()`.

Returns The number of data points inside the rectangle or a negative error code.

Channel offset

This command does not make any allowance for a channel offset (`XYOffset()`). For this to work as documented, both channels must have the same offset values.

See also:
`XYAddData(), XYColour(), XYInCircle(), XYInRect(), XYDelete(), XYDrawMode(), XYGetData(), XYJoin(), XYKey(), XYRange(), XYSetChan(), XYSize(), XYSort(), XYZOrder()`

XYInCircle()

This gets the number of data points inside a circle defined by `xc`, `yc`, and `r` in the current XY view. A general point `(x,y)` is considered to be inside the circle if:

$(x\text{-}xc)^2 + (y\text{-}yc)^2 <= r^2$

Points lying on the circumference are considered inside, but owing to floating-point rounding effects they may be indeterminate.

```
Func XYInCircle(chan%, xc, yc, r{, list%[]});
```

`chan%` A channel number in the current XY view. The first channel is number 1.

`xc,yc` These are the x and y co-ordinates of the centre of the circle.

`r` This is the radius of the circle. `r` must be >= 0.

`list%` This is an optional integer array that is filled in with the indices of the points in the channel that were inside the circle. Points are filled from index 0 of the array until the array is full or there are no more points. The index values returned are the index positions in the current sort mode for the channel. You can read the points back with `XYGetData()`.

Returns The number of data points inside the circle or a negative error code.

See also:
`XYAddData(), XYColour(), XYCount(), XYDrawMode(), XYGetData(), XYInChan(), XYInRect(), XYJoin(), XYRange(), XYKey(), XYSetChan(), XYSize(), XYSort(), XYZOrder()`

XYInRect()

This function returns the number of data points in a channel of the current XY view that lie inside a rectangle. To be inside, a data point must lie between the low rectangle coordinate up to, but not including the high coordinate. This is so that two rectangles with a common edge will not both count a data point on the boundary.

```
Func XYInRect(chan%, xl, yl, xh, yh{, list%[]});
```

`chan%` A channel number in the current XY view.

`xl,xh` The `x` co-ordinates of the left and right hand edges of the rectangle. `xh` must be greater than or equal to `xl`.

`yl,yh` The `y` co-ordinates of the bottom and top edges of the rectangle. `yh` must be greater than or equal to `yl`.

`list%` This is an optional integer array that is filled in with the indices of the points in the channel that were inside the rectangle. Points are filled from index 0 of the array until the array is full or there are no more points. The index values returned are the index positions in the current sort mode for the channel. You can read the points back with `XYGetData()`.

Returns The number of data points inside the rectangle or a negative error code.

See also:
`XYAddData(), XYColour(), XYInChan(), XYInCircle(), XYDelete(), XYDrawMode(), XYGetData(), XYJoin(), XYKey(), XYRange(), XYSetChan(), XYSize(), XYSort(), XYZOrder()`

XYJoin()

This function gets or sets the data point joining method of a channel in the current XY view. Data points can be separated, joined by lines, or joined by lines with the last point connected to the first point (making a closed loop), filled or filled and framed.

```
Func XYJoin(chan% {,join%});
```

`chan%` A channel number in the current XY view. The first channel is number 1. -1 is also allowed, meaning all channels.

`join%` If present, this is the new joining method of the channel. If this is omitted, no change is made. The data point joining methods are:

- 0 Not joined by lines (this is the default joining method)
- 1 Joined by lines. The line styles are set by `XYDrawMode()`.
- 2 Joined by lines and the last data point is connected to the first data point to form a closed loop.
- 3 Not joined by lines, but the channel data is filled with the channel fill colour.
- 4 Joined by lines and filled with the channel fill colour.
- 5 Draw as a histogram. Each consecutive pair of points (in the current sort order), defines a histogram bin. The start and end of each bin are set by the x positions and the bin contents are defined by the first point. For n bins you need n+1 points; the last point defines the end of the last bin and the y value is ignored (we recommend that you set it to 0). Drawing is most efficient when the points are ordered in increasing x order.

Returns The joining method at the time of the call or a negative error code.

Example

The following example takes the current result view and copies it as a histogram into an XY view:

```
if (ViewKind() <> 4) then Message("Needs current result view"); halt endif;
var rv% := View();            'Save the result view handle
var xl := BinToX(0);          'start point of first bin
var xh := BinToX(MaxTime());  'end point of last bin
var xinc := BinSize();        'Width of each bin
var xy% := FileNew(12, 1);    'make a visible XY view
if (xy% <= 0) then Message("Create xy view failed"); halt endif;
XYAddData(1, View(rv%,1).[], xinc, xl); 'copy data values
XYAddData(1, xh, 0);          'add right hand side of last bin
XYJoin(1, 5);                 'set histogram mode
XRange(xl,xh);                'show the data range
```

See also:

`XYAddData()`, `XYColour()`, `XYCount()`, `XYDelete()`, `XYDrawMode()`, `XYGetData()`, `XYInCircle()`, `XYInRect()`, `XYRange()`, `XYKey()`, `XYSetChan()`, `XYSize()`, `XYSort()`, `XYZOrder()`

XYKey()

This gets or sets the display mode and positions of the channel key for the current XY view. The key displays channel titles (set by `ChanTitle$()`) and drawing symbols of all the visible channels. It can be positioned anywhere within the data area. The key can be framed or unframed, transparent or opaque and visible or invisible. `ChanKey()` can also be used to control XY keys and is the preferred way to control all keys.

```
Func XYKey(which%{, new});
```

`which%` This determines which property of the key we are interested in. Properties are:

- 1 Visibility of the key. 0 if the key is hidden (default), 1 if it is visible.
- 2 Background state. 0 for opaque (default), 1 for transparent.
- 3 Draw border. 0 for no border, 1 to draw a border (default)
- 4 Key left hand edge x position. It is measured from the left-hand edge of the x axis and is a percentage of the drawn x axis width in the range 0 to 100. The default value is 0.
- 5 Key top edge y position. It is measured from the top of the XY view as a percentage of the drawn `y` axis height in the range 0 to 100. The default is 0.

`new` If present it changes the selected property. If it is omitted, no change is made.

Returns The value selected by `which%` at the time of call, or a negative error code. The value returned for `which%` values 4 and 5 is quantised to the pixel position of the key, so will often differ from the value set.

See also:
View menu Options command, ChanKey(), ChanTitle$(), XYAddData(), XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYRange(), XYSetChan(), XYSize(), XYSort(), XYZOrder()

XYOffset()

This command sets and gets a channel offset. The offset moves the origin of the x,y co-ordinate system when the channel is drawn. This allows you to generate a grid of histograms or to generate waterfall displays.

```
Proc XYOffset(chan%, x, y{, opt%});
Proc XYOffset(chan%, &x, &y{, opt%});
```

chan% A channel in the current XY view.

x,y Used when setting the channel offset. This is the offset in the current *axis units*.

&x,&y Used when returning the offset. Returns the offset in the current *axis units*.

opt% Omit or set to 0 to set the offset. Set -1 to return the offset.

Axis units
In Spike2 you have the choice of drawing axes in linear or logarithmic mode. In linear mode, the axis units are the same as the user units. In logarithmic mode, the axis units are Log(user units).

This command is somewhat experimental (it was added at version 7.09) and more opt% values may be added in the future.

See also:
XYRange(), XYZOrder()

XYRange()

This function gets the range of data values of a channel or channels in the current XY view. This is equivalent to the smallest rectangle that encloses the points.

```
Func XYRange(chan%, &xLow, &yLow, &xHigh, &yHigh{, offs%});
```

chan% A channel number in the current XY view or -1 for all channels or -2 for all visible channels. The first channel is number 1.

xLow A variable returned with the smallest x value found in the channel(s).

yLow A variable returned with the smallest y value found in the channel(s).

xHigh A variable returned with the biggest x value found in the channel(s).

yHigh A variable returned with the biggest y value found in the channel(s).

offs% If this is present and non-zero, the returned values allow for any channel offset set by XYOffset(). Otherwise, any channel offset is ignored.

Returns 0 if there are no data points, or the channel does not exist, 1 if values found.

See also:
XYAddData(), XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYKey(), XYSetChan(), XYSize(), XYSort(), XYZOrder()

XYSetChan()

This function creates a new channel or modifies an existing channel in the current XY view. It is an error to use this function if the current view is not an XY view. This function can modify all properties of an existing channel without calling the XYSize(), XYSort(), XYJoin() and XYColour() commands individually.

```
Func XYSetChan(chan% {,size% {,sort% {,join% {,col%}}}});
```

`chan%` A channel number in the current XY view. If `chan%` is 0, a new channel is created. Each XY view can have maximum of 2000 channels, numbered 1 to 2000. The first channel is created automatically when you open a new XY view with `FileNew()`. If `chan%` is not 0, it must be the channel number of an existing channel to modify or -1 to modify all channels. Before Spike2 version 9 the maximum number of channels was 256.

`size%` This sets the number of data points in the channel and how and if the number of data points can extend. The only limits on the number of data points are the available memory and the time taken to draw the view.

A value of zero (the default) sets no limit on the number of points and the size of the channel expands as required to hold data added to it.

If a negative size is given, for example `-n`, this limits the number of points in the channel to `n`. If more than `n` points are added, the oldest points are deleted to make space for the newer points. If you set a negative size for an existing channel that is smaller than the points in the channel, points are deleted.

If a positive value is set, for example `n`, this allocates storage space for `n` data points, but the storage will grow as required to accommodate further points. Using a positive number rather than 0 can save time if you know in advance the likely number of data points as it costs time to grow the data storage.

`sort%` This sets the sorting method of data points. The sorting method is only important if the points are joined. If they are not joined, it is much more efficient to leave them unsorted as sorting a large list of points takes time. The sort methods are:

- 0 Unsorted (default). Data is drawn and sorted in the order that it was added. The most recently added point has the highest sort index.
- 1 Sorted by x value. The index runs from points with the most negative x value to points with the most positive x value.
- 2 Sorted by y value. The index runs from points with the most negative y value to points with the most positive y value.

If this is omitted, the default value 0 is used for a new channel. For an existing channel, there is no change in sorting method.

`join%` If present, this is the new joining method of the channel. If this is omitted, no change is made to an existing channel; a new channel is given mode 0. The data point joining methods are:

- 0 Not joined by lines (this is the default joining method).
- 1 Joined by lines. The line styles are set by `XYDrawMode()`.
- 2 Joined by lines and also connect the first and last data points to form a loop.
- 3 Not joined by lines, but the channel data is filled with the channel fill colour.
- 4 Joined by lines and filled with the channel fill colour.
- 5 Draw as a histogram. See `XYJoin()` for details and an example.

`col%` If present, this sets the index of the colour in the colour palette to use for this channel. There are 40 colours with indices 0 to 39. If omitted, the colour of an existing channel is not changed. The default colour for a new channel is the colour that a user has chosen for an ADC channel in a time window. As an alternative to using a colour index, you can use `ChanColourSet()` to set the channel colour.

Returns The highest channel number that was affected or a negative error code. When you create a channel, the value returned is the new channel number.

See also:

`ChanColourSet(), XYAddData(), XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYKey(), XYRange(), XYSize(), XYSort(), XYZOrder()`

XYSetData()

This command is used to modify the data in an XY channel. Use `XYAddData()` to add new data points. This command was added at Spike2 version [9.02].

```
Func XYSetData(chan%, x, y, index%);
```

chan% A channel number in the current XY view. The first channel is number 1.

x The new x co-ordinate of the data point.

y The new y co-ordinate of the data point.

index% The 0-based index (in the current sort mode) of the data point in the channel to be modified. If the data is sorted, the modified point will not (in general) have the same index as the original. This can cause problems if you wish to modify multiple points as this may change the indices of all the other points.

Returns 1 if the point was modified, 0 if the index does not exist and a negative error code if the channel does not exist.

See also:
XYAddData(), XYCount(), XYDelete(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYRange(), XYSize(), XYSort(), XYZOrder()

XYSize()

This function gets and sets the limits on the number of data points of a channel in the current XY view. Channels can be set to have a fixed size, or to expand as more data is added. The only limit on the number of data points is the available memory and the time taken to draw data.

```
Func XYSize(chan% {,size});
```

chan% A channel number in the current XY view. The first channel is number 1.

size% This sets the number of data points in the channel and how and if the number of data points can extend. A value of zero sets no limit on the number of points and the size of the channel expands as required to hold data added to it.

If a negative size is given, for example -n, this limits the number of points in the channel to n. If more than n points are added, the oldest points are deleted to make space for the newer points. If you set a negative size for an existing channel that is smaller than the points in the channel, points are deleted.

If a positive value is set, for example n, this allocates storage space for n data points, but the storage will grow as required to accommodate further points. Using a positive number rather than 0 can save time if you know in advance the likely number of data points as it costs time to grow the data storage.

If this is omitted, there is no change to the size.

Returns If the number of points for the channel is fixed at n points, the function returns n. Otherwise, the function returns the maximum number of points that could be stored in the channel without allocating additional storage space.

See also:
XYAddData(), XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYKey(), XYRange(), XYSetChan(), XYSort(), XYZOrder()

XYSort()

In the current XY view, gets or sets the sorting method of the data in a channel.

```
Func XYSort(chan% {,sort%});
```

chan% The channel number in the current XY view. The first channel is number 1.

sort% This sets the sorting method of data points. The sorting method is only important if the points are joined. If they are not joined, it is much more efficient to leave them unsorted as sorting a large list of points takes time. The sort methods are:

0 Unsorted (default). Data is drawn and sorted in the order that it was added. The most recently added point has the highest sort index. This is the fastest (most efficient) sorting method.

1 Sorted by x value. The index runs from points with the most negative x value to points with the most positive x value.

2 Sorted by y value. The index runs from points with the most negative y value to points with the most positive y value.

If this is omitted, there is no change in sorting method.

Returns The function returns the sorting method at time of call or a negative error code.

See also:
XYAddData(), XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYKey(), XYRange(), XYSetChan(), XYSize(), XYZOrder()

XYZOrder()

In an XY view, this commands allows you to override the standard channel drawing order, which is channel number order; higher numbered channels draw on top of lower numbered channels. From version [10.17] you can assign an XY channel a ZOrder value with this command and from the XY View Channel Information dialog. Channels are drawn from lowest ZOrder value to highest. In the event of a tie, the channel with the lower channel number draws first.

```
Func XYZOrder(chan%{, zOrder})
```

chan% The channel number in the current XY view. The first channel is number 1. You can set the zOrder value of all channels to the same value by setting the channel number to 0. You can set chan% to -1 to get the minimum zOrder value set for any channel and to -2 to get the maximum value set for any channel.

zOrder If present, this value determines the order in which the channel is drawn with respect to all the other channels. If omitted, the command reports the channel number value, unless chan% is 0, when all channels are reset to 0. When a channel is created it has a ZOrder value of 0.0 and the channel number determines the drawing order. If chan% is -1 or -2, this value is ignored.

Returns If chan% is a valid channel number, the channel zOrder value before any change is made. If chan% is -1 or -2 the return value is the minimum or maximum zOrder value for any channel. All other values of chan% return 0.

See also:
XYAddData(), XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYKey(), XYRange(), XYSetChan(), XYSize(), XYSort()

Y

YAxis()
YAxisAttrib()
YAxisLock()
YAxisMode()
YAxisStyle()
YHigh()
Yield()
YieldSystem()
YLow()
YRange()

YAxis()

This function is used to turn the y axes on and off, in the current view and to find the state of the y axes in a view.

```
Func YAxis({on%});
```

on% Optional argument that sets the state of the axes. If omitted, no change is made. Possible values are:

0 Hide all y axes in the view.

1 Show all y axes in the view.

Returns The state of the y axes at the time of the call (0 or 1) or a negative error code.

See also:
`ChanOffset(), ChanScale(), Grid(), Optimise(), XAxis(), XScroller(), YAxisMode(), YHigh(), YLow(), YRange()`

YAxisAttrib()

This function controls the options available through check boxes in the Y Axis dialog for all views that have y-axes. If you set values that would cause illegible or unintelligible axes, they are stored but not used unless other axis attributes change to make the values useful. You cannot set logarithmic axes in a time view.

```
Func YAxisAttrib(cSpc{, flags%});
```

`cSpc` A channel specifier for the channel(s) to modify or report. When multiple channels are specified, returned values are for the first channel.

`flags%` A value of 0 sets a linear axis with no auto-adjust of units for high zoom. Add 1 for logarithmic. Add 2 to display powers on the logarithmic axis (you must have added 1 as well for this to take effect). Add 4 to cause a linear axis to auto-adjust its units by powers of 10^3 at high zoom around 0. Add 8 (in addition to 4) to use an SI prefix on the units to indicate scales. Omit this argument for no change to the attributes.

Returns The sum of the current flags set for the y-axis of the first channel in the channel specification. If the channel does not exist, the result is 0.

See also:
`YAxis(), YAxisMode(), YAxisStyle(), YHigh(), YLow(), YRange(), XAxisAttrib()`

YAxisLock()

This function locks and unlocks the axes of grouped channels and reports on the locked state of grouped channels. If you lock a group, the grouped channels keep their own axis ranges, but display using the axis of the first channel in the group. The `YRange()`, `YHigh()` and `YLow()` commands operate on the information stored with a channel. To change the displayed range of grouped and locked channels, you must use `YRange()` on the first channel in a group.

```
Func YAxisLock(chan%{, opt%{, vOffs}});
```

`chan%` A channel that is in the group that you wish to address.

`opt%` If present, values of 1 and 0 set and unset the locked state and return the original locked state. A value of -1 returns the visual offset per channel for the group. If omitted, no change is made and the current lock state returned.

`vOffs` If present, this sets the y axis display offset to apply between channels in the group. The n^{th} channel has a visual offset of `(n-1)*offs`.

Returns The current locked state of the group unless `opt%` was -1, when it returns the y axis visual offset per channel for the group.

See also:
`ChanOrder(), YAxisMode(), YHigh(), YLow(), YRange()`

YAxisMode()

This function controls what is drawn in a y axis and where the y axis is placed with respect to the data.

```
Func YAxisMode({mode%{, hal%{, hsp%{, hvp%}}}});
```

`mode%` Optional argument that controls how the axis is displayed. If omitted or negative, no change is made. Positive values are the sum of :

1 Hide all the title information.
2 Hide all the unit information.
4 Hide y axis small ticks. They are also hidden when big ticks are hidden.
8 Hide y axis numbers. They are also hidden when big ticks are hidden.
16 Hide the big ticks and the vertical line that joins them.
32 Scale bar axis. If selected add 4 to remove the end caps.
4096 Place the y axis on the right of the data.
8192 Horizontal text for the title and units.

`hal%` The horizontal label alignment as -1 or omitted for no change, 0 for centred, 1 for aligned to the edge.

`hsp%` Set the horizontal label character space in the range 2-33 or -1 or omitted for no change.

`hvp%` Set the horizontal label vertical position as 0 for centred, 1 for top, 2 for bottom or -1 or omitted for no change.

Returns If `mode%` is positive, omitted or -1 it returns the `mode%` value at the time of the call or a negative error code. Set `mode%` to -2 to return the value of `hal%`, -3 for `hsp%` or -4 for `hvp%` (all returned values are at the time of the call).

See also:
`ChanNumbers(), XAxisMode(), YAxis(), YAxisStyle(), YHigh(), YLow(), YRange()`

YAxisStyle()

This function controls the y axis major and minor tick spacing. If you set values that would cause illegible or unintelligible axes, they are stored but not used unless the axis range or scaling changes to make the values useful.

```
Func YAxisStyle(cSpc, opt%{, major});
```

`cSpc` A channel specifier for the channel(s) to modify/report. When multiple channels are specified, returned values are for the first channel.

`opt%` Values greater than 0 set the number of subdivisions between major ticks. 0 sets automatic small tick calculation. Use -1 for no change. Values less than -1 return information, but do not change the axis style. The maximum number of subdivisions is 25.

`major` If present and `opt%` is greater than -2, values greater than 0 sets the major tick spacing. Values less than or equal to 0 select automatic major tick spacing.

Returns If `opt%` is -2 this returns the current number of forced subdivisions or 0 if they are not forced. If `opt%` is -3 this returns the current major tick spacing if forced or 0 if not forced. Otherwise the return value is 0 or a negative error code. If multiple channels are specified the return value is for the first channel in the list.

See also:
Channel specifiers, `YAxis(), YAxisMode(), YHigh(), YLow(), YRange(), XAxisStyle()`

YHigh()

This function returns the upper y axis limit for the y axis of a channel in a time, result or XY view. The return value ignores the grouped or locked state of the channel. In a grid view it returns the 0-based bottom visible line.

```
Func YHigh({chan%});
```

`chan%` The channel number, which must be present and valid for a time or result view. Use 0 or omit for an XY view or a grid view.

Returns The value at the appropriate end of the axis or the bottom visible line in a grid view. In a grid view, the first row is 0.

See also:
YAxisLock(), YLow(), YRange(), ChanScale(), ChanOffset(), Optimise()

Yield()

This function suspends script operation for a user defined time and allows the system to idle. During the idle time, invalid screen areas update, you can interact with the program and Spike2 has the opportunity to do housekeeping such as releasing temporary system resources. If your script runs for long periods without using Interact(), Toolbar() or DlgShow(), adding an occasional Yield() can make it feel more responsive and stop the system marking Spike2 as "not responding". However, Yield() uses up time and slows down script execution, even with wait set to 0; avoid calling it inside loops, particularly deeply nested ones.

```
Func Yield({wait{, allow%}});
```

wait The minimum time to wait, in seconds. If omitted or 0, Spike2 gives the system one idle cycle. If greater than 0 and there is still waiting time left after an idle cycle completes, other processes are given the opportunity to run between idle cycles. If set negative, there is no idle cycle but the allow% argument is applied.

allow% Optional and set to 0 if omitted. This defines what the user can do during the wait period. See Interact() for the allowed values. The allow% value is cancelled after the command unless wait is 0 or negative. From [10.12] if the allow% value is negative no change is made to the allow actions.

Returns From [10.12] if wait is negative, the function returns the allow% value in effect at the time of the call. Otherwise it returns 1.

See also:
Interact(), MaxTime(), Seconds(), Toolbar(), YieldSystem()

YieldSystem()

The YieldSystem() command causes Spike2 to surrender the current time slice and suspends the user interface and script thread for a user-defined time or until a new message arrives in the Spike2 input queue. It has no effect on sampling, which runs in a separate thread. Unlike Yield(), it does not allow Spike2 to idle.

To share the system CPU(s) among competing processes, the operating system allocates time slices of around 10 milliseconds based on process priorities and recent process activity. A process can surrender a time slice if it has nothing to do. A typical application spends most of its time waiting for user input, which appears as messages in the application message queue; it will surrender a time slice if the message queue is empty unless it has a task to work on. Spike2 normally surrenders time slices, but if you run a script, it runs for the full time slice unless it is in Yield(), Interact(), or you use ToolBar() or DlgShow() without an idle routine.

```
Proc YieldSystem({wait});
```

wait The time to wait, in seconds, before resuming the thread. Values are rounded to the nearest millisecond. Values greater than 10 are treated as 10 seconds; values less than –10 are treated as –10 seconds.

For wait values greater than 0, the wait is ended by unserviced messages; keyboard and mouse activity, timers for screen updates and the like cause messages. If wait is 0 or omitted, the current time slice is surrendered, but if Spike2 is the highest priority task it will be re-scheduled immediately. Negative values suspend Spike2 for –wait seconds regardless of messages.

YieldSystem() with wait values greater than 0 returns immediately if there are messages in the input queue. Unless you allow Spike2 to idle, either with a Yield() call or with Toolbar() or DlgShow(), there will always be pending messages, so it will have no effect. If you have a script loop that causes 100% CPU usage, inserting:

```
Yield();YieldSystem(0.001);
```

will give other processes a chance to run. Increasing the wait time up to 0.05 will further reduce the CPU usage. Larger values have little additional effect due to timer messages ending the wait early. To give as much system time as possible to other tasks without allowing Spike2 to idle, you can use:

```
YieldSystem(-0.001);
```

In this case, increasing the time to –10.0 will have an effect; Spike2 will feel completely unresponsive until the time period has elapsed.

See also:
`Interact(), DlgShow(), Seconds(), Toolbar(), Yield()`

YLow()

This function returns the lower y axis limit for the y axis of a channel in a time, result or XY view. The return value ignores the grouped or locked state of the channel. In a grid view it returns the 0-based row number of the top visible row.

```
Func YLow(chan%);
```

`chan%` The channel number. Use 0 for an XY view or a grid view.

Returns The value at the appropriate end of the axis or the top row in a grid view.

See also:
`YAxisLock(), YHigh(), YRange(), ChanScale(), ChanOffset(), Optimise()`

YRange()

This sets the displayed y axis range for a channel in a Time, Result, XY or Grid view. It can also be used in a Spike Shape dialog.

Time, result and XY views and Spike Shape dialogs

It has no effect if the current channel display mode has no y axis. If the y range changes, the display updates at the next opportunity. If you omit `low` and `high`, Spike2 displays the "full" range of the channel, equivalent to the Show All button in the Y Axis dialog. In a Spike Shape dialog, the value of `low` or `high` farthest from zero sets the displayed range.

```
Proc YRange(cSpc{, low, high});
```

`cSpc` A channel specifier for the channel(s) to modify. This is ignored in a Spike Shape dialog and should be set to 0.

`low` The value for the bottom of the y axis. If omitted, and the channel type has a known range, Spike2 sets `low` and `high` to suitable limits. For example, for a waveform channel displayed as a waveform the limits are those set by the 16-bit nature of the data. For a sonogram channel, the limits are 0 and half the sampling rate. For an event channel the limits are 0 and the maximum sustained event rate set in the sampling configuration dialog.

`high` The value for the top of the y axis. If `low` and `high` are the same, or too close, the range is not changed. It is an error to supply `low` and omit `high`.

Grid view

In a grid view, this is used to set to top line to display in the grid. The channel specification is currently unused and should be set to 0 for future compatibility. Use the `Draw()` command or `XRange()` to set the first column.

```
Proc YRange(0, line);
```

`line` The top row number to display. The first row is 0.

See also:
Channel specifiers, `YAxisLock(), YHigh(), YHigh(), YLow(), ChanScale(), ChanOffset(), Optimise()`

Z

ZeroFind()

ZeroFind()

Find a zero (root) of a user-defined continuous function using Brent's method. You must supply a range to search that contains a zero. The method is iterative with each iteration reducing the search range. Iterations stop when a zero is found, the search range becomes smaller than a supplied tolerance or an iteration limit is reached. The need to supply a range that brackets a zero means that you must have some understanding of the general shape of the function before using `ZeroFind()`.

```
Func ZeroFind(&x, f(x), a, b{, tol{, maxIt%}});
```

`x` A real variable that is returned with a value that is within `tol` of a position for which `f(x)` is zero.

`f` The name (no brackets or argument) of a user-defined function that takes a single real argument and that returns a real value.

`a,b` These values define a range to be searched for a zero. `f(a)` and `f(b)` must both be non-zero and must have different signs.

`tol` This specifies how close the returned x must be to the exact result and must be positive. If omitted or zero, the tolerance is set based on the root position and floating point precision.

`MaxIt%` The maximum number of iterations of the algorithm in the range 1 to 200. If omitted, a maximum of 100 iterations are set.

`Return` The number of iterations left after the tolerance is achieved, or 0 if it was not, or -1 if the initial range (`a`, `b`) does not include a zero.

Examples

The following example calculates the roots of the equation x^2-2=0.

```
Func root2(x) return x*x-2 end;

var r1,r2;
ZeroFind(r1, root2, 0, 2);    'Find a root between 0 and 2
ZeroFind(r2, root2, 0, -2);   'Find a root between 0 and -2
PrintLog("Roots at %g and %g\n", r1, r2);
```

The result of this example could have been obtained by simple algebra; the roots are obviously ±√2. However, in some cases the function may not be analytic. For example, suppose we want to demonstrate the often quoted result that 95% of normally distributed values lie within two standard deviations of the mean. The indefinite integral of a Gaussian with unit standard deviation can be obtained with:

```
Func NormProp(x) return GammaP(0.5, 0.5*x*x); end;
```

This returns the fraction of the data that lies within `x` standard deviations of the mean of a normal distribution. We want the value of `x` for which `NormProp(x) = 0.95`. Put another way, we want `NormProp(x)-0.95` to be zero. The following script does this:

```
Func NormProp(x) return GammaP(0.5, 0.5*x*x); end;
Func Root(x) return NormProp(x)-0.95 end;

var sd;
ZeroFind(sd, Root, 0, 5);   'get 95% level
printlog("95%% at %4.2f\n", sd);
```

You can use this function in some circumstances to calculate the inverse of a function when no other method is available. That is, if $y = f(x)$, then $x = f^{-1}(y)$. For example, in the above case we were finding the inverse of the `NormProp(x)` function at the specific value 0.95. To tabulate the function for a range of values we would change the `Root()` function to:

```
var prop;
Func Root(x) return NormProp(x)-prop end;
```

Now, by setting `prop` to a value in the range between 0 and 1 (but not to 0 and 1), we can map the inverse function:

```
for prop := 0.01 to 0.99 step 0.01 do
    ZeroFind(sd, Root, 0, 100);
    printlog("%4.2f %5.3f\n", prop, sd);
next;
```

We have omitted 0 and 1 from our range because we cannot set a range that meets the criteria for the arguments `a` and `b` for these values (the result from `NormProp()` can only lie in the range 0 to 1, and only reaches 1 at infinity).

See also:
Normal distribution CDF

Curve fitting

Introduction

It frequently happens that you have a set of data values (x_1,y_1), (x_2,y_2) … (x_n,y_n) that you wish to test against a theoretical model $y = f(x, \mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2\ldots)$ where the $\mathbf{a}_i$ are coefficients that are to be set to constant values which give the *best fit* of the model to the data values.

For example, if we were looking at the extension of a spring (y) as it is loaded by weights (x), we might wish to fit the straight line $y = \mathbf{a}_0 + \mathbf{a}_1 x$ to some measured data points so that we could measure a weight by the extension it caused. A careful experimenter might also wish to know what the probable error was in $\mathbf{a}_0$ and $\mathbf{a}_1$ so that the probable error in any weight deduced from an extension would be known. An even more cautious experimenter might want to know if the straight-line formula was likely to model the measured data.

To avoid repeating definitions throughout the remainder of this chapter the following will be taken as defined. We apologise to the statisticians who may read this and shudder.

mean

Given a set of n values y_j, the mean is the sum of the n values divided by n.

variance

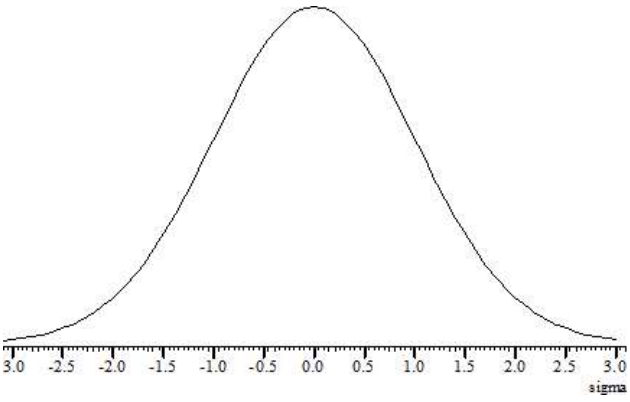
If the mean of a set of n data values y_j is y_m, then the variance (sigma squared) of this set of values is: the sum of the squares of $y_j - y_m$ divided by n if y_m is known independently of the data values and divided by $(n - 1)$ if y_m is calculated from the data values y_j. For a data set of any reasonable size, the use of n-1 or n in the denominator should make little difference.

standard deviation

The standard deviation (sigma) of a data set is the square root of the variance. Both the variance and the standard deviation are used as measures of the width of the distribution.

Normal distribution

If you measure a data value in any real system, there is always some error in the measurement. Once you have made a (very) large number of measurements, you can form a curve showing the probability of getting any particular value. One would hope that this error distribution would show a large peak at the "correct" value of the measurement and the width of this distribution would show the spread of likely errors.



There is a particular error distribution that often occurs, called the Normal distribution. If a set of measurements is normally distributed, with a mean y_m and standard deviation σ (sigma), then the probability of measuring any particular value y is proportional to:

$$P(y) \propto \exp(-\tfrac{1}{2}(y-y_m)^2/\sigma^2)$$

It is for this distribution of errors that we have the well-known result that 68% of the values lie within one standard deviation of the mean, that 95% lie within two standard deviations and that 99.7% lie within three standard deviations. Of course, **if the error distribution is not normal, these results do not apply**.

Chi-squared

The fitting routines given here define *best fit* as the values of $\mathbf{a}_i$ (the coefficients) that minimise the chi-squared value defined as the sum over the measured data points of the square of the difference between the measured and predicted values divided by the variance of the data point:

If the sigma of each data point is unknown, then the fitting routines can be used to minimise the sum of the squares of the differences between the actual data values and the values predicted by the fitted function which produces the same result as a chi-squared fit would produce if the variance of the errors at all the data points was the same. This is commonly called least-squares fitting (meaning that the fit minimises the sum of squares of the errors between the fitted function and the data).

Chi-squared fitting is also a maximum likelihood fit if the errors in the data points are normally distributed. This means that as well as minimising the chi-squared value, the fit also selects the most probable set of coefficients that model your data. If your data measurement errors are not normally distributed you can still use this method, but the fit is not maximum likelihood.

If your errors are normally distributed and if you know the variance(s) of the data points, you can form good estimates of the variance of the fitted coefficients, and you can also test if the function you have fitted is likely to model the data.

If your errors are normally distributed but you do not know the variance of the errors at the data points, you can make an estimate of the variance of the errors (based on the assumption that the variance is the same for them all and that the model does fit the data), by fitting your model and calculating the variance from the errors between the best fit and the data. Having done this, you cannot then use this variance to test if the fit is likely to model the data.

Residuals

Once your fit is completed, it is a good idea to look at the graph of the errors between your original data and the fitted data (the residuals or residual errors). If your errors are normally distributed and are independent, you would expect this graph to be more or less horizontal with no obvious trends. If this is not the case, you should consider if the correct model function has been selected, or if the fitting function has found the true minimum.

Linear fit, Non-linear fit

A linear fit is one in which the theoretical model $y = f(x, \mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2\ldots)$ can be expressed as $y = \mathbf{a}_0 f_0(x) + \mathbf{a}_1 f_1(x) + \mathbf{a}_2 f_2(x)$ … for example $y = \mathbf{a}_0 + \mathbf{a}_1 x + \mathbf{a}_2 x^2$. Linear fits are relatively quick as they are done in one step. Usually, the only thing that can cause a problem is if the functions $f_i(x)$ are not linearly independent. The methods we use can usually detect this problem, and can still give a useful result.

A non-linear fit means all other cases, for example, $y = \mathbf{a}_0 \exp(-\mathbf{a}_1 x) + \mathbf{a}_2$. We solve these types of problem by making an initial guess at the coefficients (and ideally providing a range of values that the result is known to lie in) and then improving this guess. This process repeats until some criterion is met. Each repeat is called an *iteration*, so we call this an iterative process.

Covariance array

Several of the script fitting routines return a covariance array. If you have n coefficients, this array is of size n by n and is diagonally symmetric. If the errors in the original data points are normally distributed, the diagonal elements of this array are the variances in the values of the fitted coefficients. The remaining elements are the co-variances of pairs of the fitting parameters and can be used to estimate errors in derived values that depend on the product of two of the coefficients. If the errors are not normally distributed, the further away from normal the errors are, the less useful is the covariance array as a direct indication of the variances in the fitted coefficients.

For example, in the case of the linear fit:

$y = \mathbf{a}_0 + \mathbf{a}_1 x + \mathbf{a}_2 x^2$

you might collect your three coefficients in the array `coef[]`, and the covariance in the array `covar[][]`. In this case, the $\mathbf{a}_0$ value is returned in `coef[0]` and its variance in `covar[0][0]`, the $\mathbf{a}_1$ value is returned in `coef[1]` and its variance in `covar[1][1]`, and the $\mathbf{a}_2$ value is returned in `coef[2]` and its variance in `covar[2][2]`.

Because the array is diagonally symmetric, `covar[i][j]` is equal to `covar[j][i]` and the off-diagonal elements are the expected variance in the product of pairs of the coefficients, so `covar[1][2]` is the variance of $\mathbf{a}_1\mathbf{a}_2$.

If you have not supplied the standard deviations of the errors in the data points, the covariance array is calculated on the assumption that all data points have a standard deviation of 1.0, and the covariance array is incorrectly scaled. In this case, if inspection of the residuals leads you to the conclusion that the function does indeed fit the data and that the errors are more or less the same for all values and not too far from normally distributed, then you can scale the covariance array to the correct values by multiplying all the elements of the array by the sum of squares of the errors between the data and the fitted values divided by the number of data points. If there are `nD%` data points and `nC%` coefficients and the sum of squares of the errors is `errSq`, then use `ArrMul(covar[][], errSq/(nD%-nC%));` to rescale the covariance (see Numerical Recipes, Modelling of Data, Fitting Data to a Straight Line).

What does the covariance mean?

Having fitted our data, we would like some idea of how the errors in the original data feed through to uncertainties in the values of the coefficients. The best way to do this is to obtain many sets of (x,y) data and fit our coefficient to each set. Then we can inspect the values of the coefficients and obtain a mean and standard deviation for each coefficient. However, this is very time consuming and may not be practical.

If the errors in the data are normally distributed (or not too far from this ideal case) and known, then the covariance array gives you some useful information. The square root of the covariance for a particular coefficient is the expected standard deviation in that value (given that the remaining coefficients remain fixed at optimum values). In script language terms, the standard deviation of `coef[i]` is `sqrt(covar[i][i])`.

In this case you would expect the coefficient to be within one standard deviation of the "correct" result 68% of the time, within 2 standard deviations 95% of the time and within 3 standard deviations 99.7% of the time.

Testing the fit

If the errors in the original data are normally distributed and known (not calculated from the fit), and you know the chi-squared value for the fitted data, you can ask the question, "Given the known errors in the original data, how likely is it that you would get a value of chi-squared at least this large given that the data is correctly modelled by the fitting function plus normally distributed noise?" The answer is (at least in terms of the script language) that the probability is: `GammaQ((nData% - nCoef%)/2.0, chiSq/2.0);` where `nData%` is the number of data points to be fitted, `nCoef%` is the number of coefficients that were fitted and `chiSq` is the chi-squared value for the fit. `GammaQ()` is the incomplete Gamma function.

If you want to follow this result up in a statistical textbook, you should look up *chi-squared distribution for n degrees of freedom*. In our case, we have `nData%-nCoef%` degrees of freedom.

If the fit is reasonable, you should expect a probability value between 0.1 and 1 (but be a bit suspicious if you always get values close to 1.0, as you may have overestimated the errors in the data). If the wrong function has been fitted or if the fit is poor you usually get a very small probability.

Intermediate values (0.0001 to 0.1) may indicate that the errors in the original data were actually larger than you thought, or they may indicate that the model does not explain all the variation in the data.

Fitting routines

The `ChanFit...()` family of commands give a script the same capability as the Analysis menu Fit Data command.

`ChanFit()` This command associates a fit with a data channel in a time, result or XY view, performs the fit and returns basic fit information.

`ChanFitCoef()` This gets and sets fit coefficients, coefficient limits and the hold flags. It is equivalent to the coefficient page of the Fit Data dialog.

`ChanFitShow()` This controls the display of the fitted data.

`ChanFitValue()` This returns the value at a given x position of the fitted function.

The remaining fitting functions allow you to fit curves to data in arrays:

`FitPoly()` This fits a polynomial of the form $y = \mathbf{a}_0 + \mathbf{a}_1 x + \mathbf{a}_2 x^2 + \mathbf{a}_3 x^3 \ldots$

`FitLine()` This fits a straight line to data in a time or result view.

`FitLinear()` This fits $y = a_0 f_0(x) + \mathbf{a}_1 f_1(x) + \mathbf{a}_2 f_2(x) \ldots$where the $f_n(x)$ are user-defined functions of x.

`FitExp()` This fits multiple exponentials of the form $y = \mathbf{a}_0 \exp(-x/\mathbf{a}_1) + \mathbf{a}_2 \exp(-x/\mathbf{a}_3) \ldots$

`FitGauss()` This fits Gaussian distributions of the form $y = \mathbf{a}_0 \exp(-½(x-\mathbf{a}_1)^2/\mathbf{a}_2^2) + \mathbf{a}_3 \exp(-½(x-\mathbf{a}_4)^2/\mathbf{a}_5^2) + \ldots$

`FitSin()` This fits multiple sinusoids of the form $y = \mathbf{a}_0 \sin(\mathbf{a}_1 x+\mathbf{a}_2) + \mathbf{a}_3 \sin(\mathbf{a}_4 x+\mathbf{a}_5) + \ldots$

`FitNLUser()` This fits a user-defined function of the form $y = f(x, \mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2 \ldots)$ to a set of data points where the $\mathbf{a}_i$ are constant coefficients to be determined. You must be able to calculate the differential of the function f with respect to each of the $\mathbf{a}_i$ coefficients. This is the most general fitting routine, and also the slowest and most complex to use.

`GammaP()`
`GammaQ()` Use these functions to calculate the error function *erf(x)*, the cumulative Poisson probability function and the Chi-squared probability function. These are very useful when considering probabilities of fits associated with the normally distributed data.

`LnGamma()` This calculates the natural logarithm of the Gamma function. It can be used to calculate the log of large factorials and is useful when working with probability distributions.

`BetaI()` The incomplete Beta Function comes up when you work with the Binomial Distribution, Student's T Distribution and the F-Distribution.

XY Views

You will find a brief introduction to XY views in Getting Started with Spike2: XY Views.

There are several example scripts included with Spike2 that illustrate some of the uses of XY views. You will find them in the `scripts` folder:

`FFTWater` This draw a “waterfall” type display showing the variation in the power spectrum of a waveform channel with time.

`Clock` An analogue clock for your Toolbar idle routine.

Creating an XY view

Although you can create an XY view from the File New menu command, the usual way to generate XY views is with the Analysis menu Measurements command or with the script language. The script language can generate an XY view for general use with the `FileNew()` command and as the target for measurements with a

Process() command with the MeasureToXY() command. See the documentation for MeasureToXY() for an example.

The following assumes that you have some familiarity with the script language. An XY view always has at least one data channel, so when you create a view, you also create a channel. The following script code shows you how to make an XY view:

```
var xy%;                    'handle for the XY view
xy% := FileNew(12,1); 'type 12=XY, 1=make visible now
```

If you want to add additional channels you can do this using the XYSetChan() command. You can also use this command to set a channel to a particular state. The following sets channel 1 (the first channel) to show data points joined by lines with no limit on the number of data points, drawn in the standard colour:

```
XYSetChan(1, 0, 0, 1);       'chan 1, no size limit, no sort, joined
XYDrawMode(1, 2, 0);  'set a marker size of 0 (invisible)
```

To add data points to a channel you use the XYAddData() command. You can add single points, or pass an array of x and y co-ordinates. The following code adds three points to draw a triangle:

```
XYAddData(1, 0, 0);          'add a point to channel 1 at (0, 0)
XYAddData(1, 1, 0);          'add a point at (1,0)
XYAddData(1, 0.5, 1); 'add a point at (0.5, 1)
```

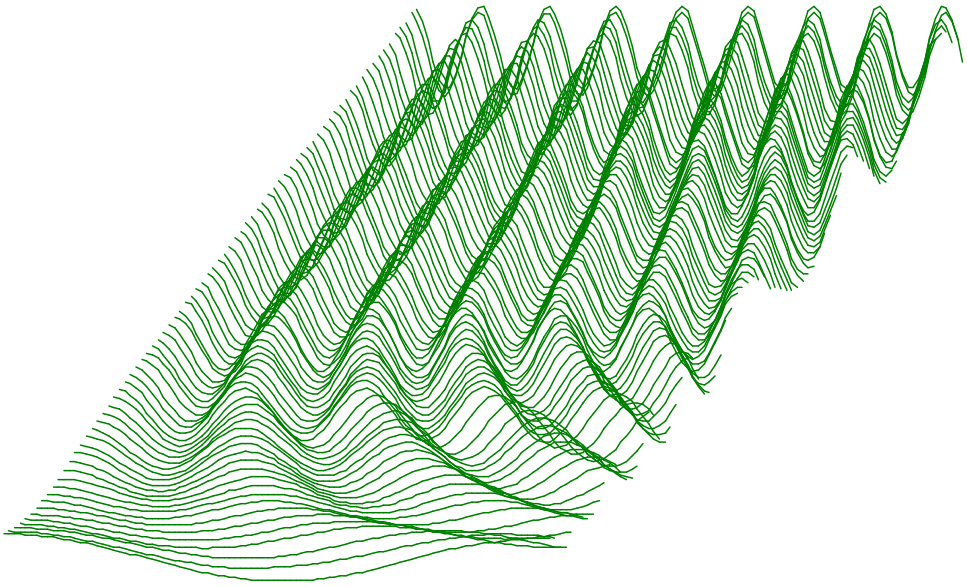
You will notice that the result of this draws only two sides of a triangle. We could complete the figure by adding an additional data point at (0,0), but it is just as easy to change the line joining mode to "Looped", and the figure is completed for you:

```
XYJoin(1,2);                 'set looped mode
```

See also:
MeasureToXY(), FileNew()

Overdrawing data

You can use the XY view to overdraw data. For example, suppose we have a waveform channel and another channel of trigger markers and we want to superimpose the first 100 data points after each trigger. The following example does this. To make it a little easier to see, we have displaced each triggered data sweep slightly to the right and up. We have also turned off the x and y axes.



The code required to produce this image is quite short and can be easily adapted to display other types of data. This example expects to run with the demo.smr data file in the Spike2\Data folder created when you installed the program. To run with other files you would need to adjust the channel numbers.

The first line declares variables to remember the time and XY window handles (identifiers) and a variable to hold the number of points read. The second line checks that the current view is a time view and gives up if it is not. The third line remembers the time view handle.

```
var tv%, wh%, np%;          'time view, XY view, points read
if ViewKind() then Message("Not a time view");Halt endif;
tv% := View();              'save view handle
```

The next three lines declare an array to hold 100 data points, a variable to hold the trigger time (and we set it to -1 so we see the first event, even if it is at time 0), and we call the `MakeWindow%()` function (see below) that creates the XY window for us. We set the first argument to the time interval between data points in out time view so that the x spacing of the data points is correct. The second and third arguments to this function set the distance each "slice" of data is moved to the right and up.

```
var wave[100];              'space for 100 waveform data points
var t := -1;                'start time for trigger search
wh% := MakeWindow%(BinSize(1), 0.01, 0.5); 'start waterfall view
```

To generate the data we run round a loop (`repeat…until (t<0) or (t>20);`) that finds the time of the next trigger on channel 2 , reads in 100 data points into the array `wave[]` and then as long as the data read correctly, adds the data into the XY window.

```
repeat
   View(tv%);               'Make time view current
   t := NextTime(2, t);     'Get next trigger time from channel 2
   np% := ChanData(1, wave[], t, MaxTime()); 'get channel 1 data
   if (np% > 0) then AddSlice(wave[:np%]) endif; 'add to picture
until (t < 0) or (t>20);    'until end or we reach 20 seconds
```

The final task is to make the XY window visible and halt.

```
View(wh%).WindowVisible(1); 'make XY window visible
halt;
```

The remaining code can be copied from the `FFTWater.s2s` script. The variables with names starting `WF` are used to remember the initial waterfall settings. There are two functions in this code. `MakeWindow%()` prepares for the waterfall display by creating the XY window and storing the information needed to position the channels. `AddSlice()` takes an array of data points and adds it to the display as an additional channel.

```
'================== Waterfall display code =====================
var WFxInc,WFyInc,WFbinSz,WFSlices%,WFvh%;
'
'xBinSz Width of each bin (sample interval)
'xInc   Add to each slice x co-ords to give waterfall effect
'yInc   Add to each slice y co-ords to give waterfall effect
Func MakeWindow%(xBinSz, xInc, yInc)
WFSlices% := 0;             'no slices yet
WFxInc := xInc;             'save x increment per slice
WFyInc := yInc;             'save y increment per slice
WFbinSz:= xBinSz;           'save data point separation
WFvh% := FileNew(12);       'create a new XY window (hidden)
return WFvh%;               'return the XY window handle
end;

Func AddSlice(y[])          'Add data to the waterfall
View(WFvh%);                'select the waterfall view
var ch%:=1;                 'true if this is the first channel
if WFSlices% = 0 then       'if first channel no need to create
   XYSetChan(1,-Len(y[]),0,1);          'set original channel
else
   ch% := XYSetChan(0, -Len(y[]), 0, 1);'create new channel
   if (ch% <= 0) then return ch% endif; 'No more channels
endif;
WFSlices% := ch%;           'number of slices
XYDrawMode(ch%,2,0);        'Hide the markers (set size of 0)
var x[Len(y[])];            'space for x values, same size as y[]
ArrConst(x[], WFbinSz);     'generate x axis values, set the same
x[0]:=(ch%-1)*WFxInc;       'set x offset as first value
ArrIntgl(x[]);              'form the x positions
ArrAdd(y[],(ch%-1)*WFyInc); 'add the y offset to the y array
```

```
XYAddData(ch%, x[], y[]);   'add the (x,y) data points
return 1;                   'return >0 means all OK
end;
```

Channel offsets

At Spike2 version 7.09 we added the ability to shift a channel in an XY view with the `XYOffset()` command. This command moves the XY view origin to wherever you specify when the channel is drawn. You set the offset in axis units. That is, if the y axis is in Logarithmic mode and the x axis is linear, and you want the origin to be at the point where (x,y) would appear, you must set the offset to (x, Log(y)). This means that the offsets must be recalculated if the axis mode is changed.

The `XYInRect()`, `XYInCircle()` and `XYInChan()` functions can be used when channels are offset and tell you which visible points lie inside the rectangle, circle or channel. If you do not want to take offsets into account you must set the channel offset to (0,0). The `XYRange()` function now has an extra argument to specify if the offset is to be considered or ignored.

The previous example of a waterfall plot can be done using channel offsets by changing the `AddSlice()` function:

```
Func AddSlice(y[])          'Add data to the waterfall
View(WFvh%);                'select the waterfall view
var ch%:=1;                 'true if this is the first channel
if WFSlices% = 0 then       'if first channel no need to create
   XYSetChan(1,-Len(y[]),0,1);          'set original channel
else
   ch% := XYSetChan(0, -Len(y[]), 0, 1);'create new channel
   if (ch% <= 0) then return ch% endif; 'No more channels
endif;
WFSlices% := ch%;           'number of slices
XYDrawMode(ch%,2,0);        'Hide the markers (set size of 0)
var x[Len(y[])];            'space for x values, same size as y[]
ArrConst(x[1:], WFbinSz);   '[0] left as 0, rest set the same
ArrIntgl(x[]);              'form the x positions
XYOffset(ch%, (ch%-1)*WFxInc, (ch%-1)*WFyInc));
XYAddData(ch%, x[], y[]);   'add the (x,y) data points
return 1;                   'return >0 means all OK
end;
```

In this version, the x co-ordinates are the same for each histogram (and could be calculated once, saving a little time). The y co-ordinates can be left as read; they do not need offsetting. If you wanted to dynamically change the angle of the waterfall, you could remove the `XYOffset()` call from the `AddSlice()` routine and instead have a separate routine:

```
Proc SetAngle(xInc, yInc, nCh%)
var ch%;
for ch% := 1 to nCh% do
    XYOffset(ch%, (ch%-1)*xInc, (ch%-1)*yInc);
next;
end;
```

The way this example is written, the slices are added to the front first, working backwards. If you wanted to draw the data as a histogram (another new XY view feature in version 7.09, see `XYJoin()`), you would want to draw the data from the back to the front as channels are drawn in order and you would want the last channel to be at the front so that the filled histograms overwrite each other in a sensible way. If you did this, then in `SetAngle()`:

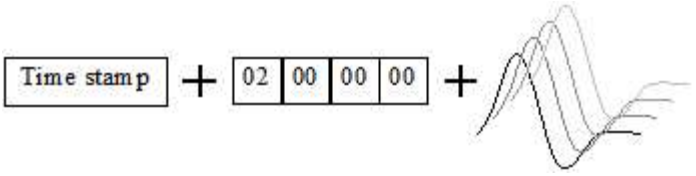
```
    XYOffset(ch%, (nCh%-ch%)*xInc, (nCh%-ch%1)*yInc);
```

so that the last channel (at the front) has no offset and the first channel (at the back) has the largest.

16: Spike sorting

Spike sorting

Introduction



Spike2 stores spike shapes as WaveMark channels. Each spike shape has a time stamp, 4 identifying codes and 1, 2 or 4 traces of up to 1000 waveform data points that define the spike (on-line you can use up to 126 points). The codes store the spike classification, obtained by matching the spike shape against shape templates or by using cluster analysis; normally only the first code is used. The time stamp is the time of the first saved data point.

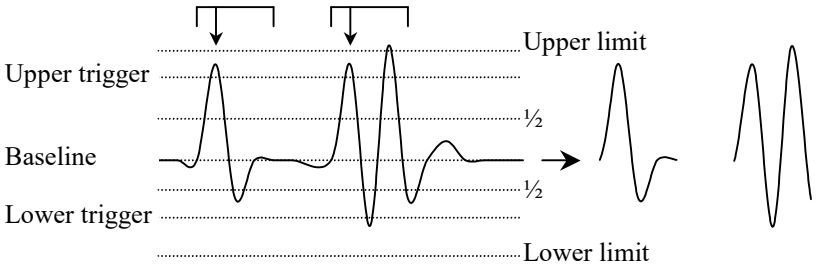
On-line, spikes can be sorted in real-time with up to 8 spike classes per channel and an optional digital output as each spike is classified. Spike2 captures up to 32 channels of spikes with a Power1401 and up to 16 channels with a Micro1401. Off-line Spike2 can extract spikes from a waveform channel and resort and edit spikes extracted on-line or off-line; you can divide spikes into almost any number of classes. Off-line spike sorting can be script-controlled.

The first part of this section is organised as reference information. This is followed by Getting *started with spike shapes and templates* which is more discursive. If you are a new user you may prefer to start there and come back to the reference material later.

Spike detection

Spikes are detected by the input signal crossing a trigger level. There are two trigger levels, one for positive-going and one for negative-going spikes. Captured spikes are aligned on the first positive or negative peak. To avoid problems due to baseline drift, the data capture routines pass the data through a high pass filter. The time constant of this filter can be adjusted to suit the data. You can define the number of data points captured before and after the peak. The spike detection algorithm is as follows:

1. Wait for the signal to lie within half the trigger levels. When it does, go to step 2.
2. Wait for the signal to cross either trigger level. If it crosses the upper trigger level go to step 3. If it crosses the lower level, go to step 4.
3. Track the positive peak signal value. If the signal falls below the peak and there are sufficient post-peak points to define a spike, go to 5. If the signal falls below half the upper trigger level, we ignore further peaks (as for the second spike in the diagram).
4. Track the negative peak signal value. If the signal rises above the peak, see if we have sufficient post-peak points to define the spike. If so, go to 5. If the signal rises above half the lower trigger level, we ignore further peaks.
5. Save the waveform and first data point time and go to step 1 for the next spike.

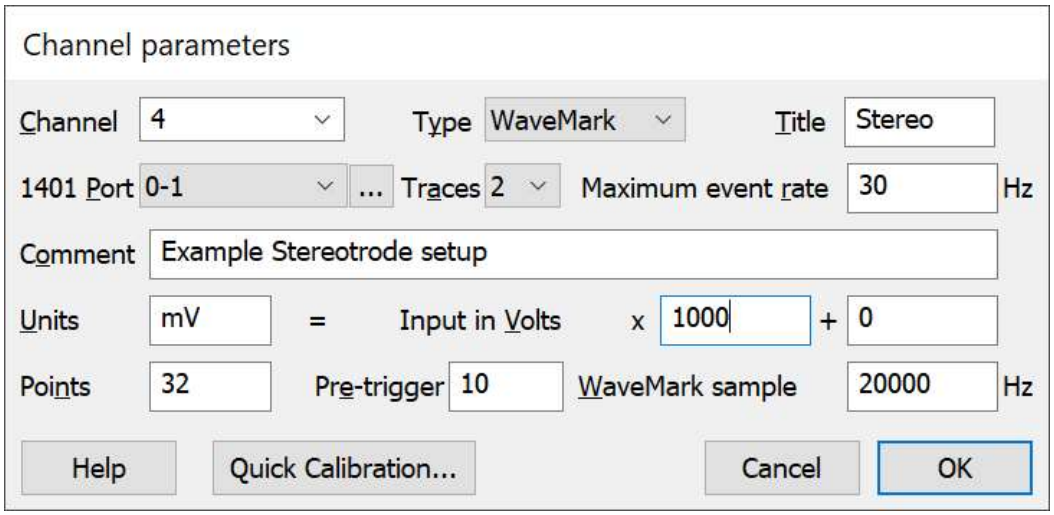


With multiple traces, each trace is tracked separately and the first trace to trigger saves the data for all traces.

When working offline and online with any 1401 except the Micro1401-2, you can choose to set two additional limit levels outside the trigger levels. Any detected spikes with a peak amplitude outside these limits are rejected. In this case the trigger and associated limit act as an amplitude window on the data. The limit applies

to the initial peak amplitude, so the second spike in the diagram is acceptable, even though some of the data points lie outside the limits.

WaveMark channel settings



To sample spike shapes you must create a WaveMark channel in the Sampling configuration dialog. The Title, Port, Channel comment, Units and Scaling fields are the same as for Waveform data. The Maximum event rate field is the total number of spikes (of all classes) that you expect per second on this channel.

The Traces field sets the number of waveforms that are combined in this channel. This should be set to 1 for a single electrode, 2 for a stereotrode and 4 for a tetrode. The 1401 Port sets the input channel number of the first trace. The remaining traces are taken from sequential channel numbers. For example, if the 1401 Port field was set to 7 and Traces was 4, the 1401 input channel numbers 7, 8, 9 and 10 would be used for the traces.

The bottom line of the dialog box sets the number of points to save per trace per spike in the range 10 to 126 (Points) and the number of points to display before the peak (Pre-trigger). You can alter the number of points before the peak and the total number of points in the template set up section, so it is not essential to get this exactly right.

The WaveMark sample rate field sets the ideal sample rate for all WaveMark channels. A sampling rate of 20 kHz per channel is often used so that spikes (of typical duration 1 to 2 milliseconds) can be usefully discriminated. The actual sampling rate depends on the number of waveform and WaveMark channels. See the Sampling configuration Resolution tab for more information on setting waveform and WaveMark sample rates.

Template matched signal

You can use digital output bits 0-7 (on the rear panel of the Micro1401 and Power1401) to signal that a spike shape has matched a template. There are two output formats (set in the on-line Template Setup). In One bit mode, a single bit pulses from low to high, then back to low for each of up to 8 templates. Bit 0 is for the first template, bit 1 for the second, and so on up to bit 7 for the eighth template. One bit mode is not useful when you have more than one channel of spikes (WaveMark data) as you cannot tell which channel produced the output.

In Coded mode, pin 23 of the digital output port pulses for each matched template, and the digital output bits 3-7 hold the ADC port number (0 to 31) and bits 0-2 hold the template (0-7 for the 8 possible templates). The data is valid on both edges of the pulse. The pulse is short, less than 1 microsecond for the Power1401, more than 1 microsecond for other 1401s. If you have more than 32 ADC channels, the bottom 5 bits of the port number are presented on bits 3-7.

Digital output connections

Data bit	7	6	5	4	3	2	1	0	Gnd	Strobe
Output pin	5	18	6	19	7	20	8	21	13	23

Coded mode uses all 8 bits, One bit mode uses one bit per template. The Micro1401 and Power1401 have independent input and output bits 0-7, however you should use the output sequencer `DIGLOW` command with caution as it controls the same output bits.

Timing of the digital output

Measurements with templates of 20 points show that the vast majority of spikes are matched within a millisecond (often within a few tens of microseconds) of the end of the spike (less than this with a Power1401). However you cannot rely on it. Even with a fast host, longer delays may occur occasionally, especially when data is written to disk at high sampling rates. At very high data rates, if the 1401 cannot keep up, it may stop matching spikes to templates until the data rate drops.

You can test the delays by feeding the output back into an event channel and correlating the Spike channel with the event channel. If you intend to use these outputs for a timing critical use it is important that you measure the typical time delays for your configuration.

Power1401-3 One bit pulse width

Beware that the `One bit` output from a Power1401-3 is very short (0.1 us). If you want to record this signal with the Power1401-3 by connecting it to an event input (for example to verify the digital output timings) you must use the rear-panel digital inputs, not the more convenient front panel Event inputs. This is because the front panel inputs have deglitching circuits that remove pulses shorter than around 0.25 us. The rear panel inputs are not deglitched in this way so can be used to time the One bit mode output.

Spike shape sorting dialogs

There are three similar dialogs that are used to sort spikes by shape:

On-line template set up

This dialog opens automatically when you open a data file for sampling with one or more WaveMark channels in the sampling configuration. You can control if this dialog appears with the `FileNew() mode%` parameter when you create a file for sampling using the script language.

Off-line template formation

This dialog is opened from a time window by right-clicking on a waveform or WaveMark channel and selecting the New WaveMark command or by using the Analysis menu New WaveMark or New N-Trode commands. From the dialog you can create a new WaveMark channel from one or more source channels.

Edit WaveMark (on-line/off-line)

This dialog can be used on-line to monitor spikes, adjust the trigger levels and control template formation. Off-line it can be used to resort WaveMark channels and to create new WaveMark channels. This dialog is also the starting point for Spike Clustering.

Menus in Spike Shape sorting dialogs

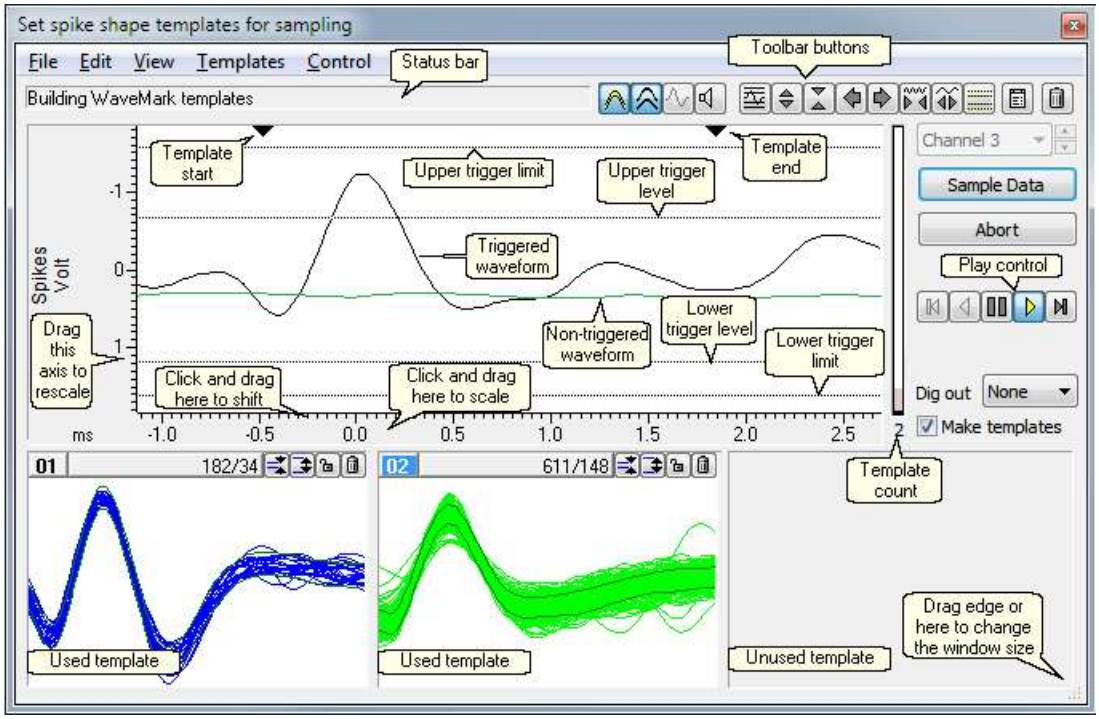
The three Spike shape dialogs (Setup Sampling, New WaveMark and Edit WaveMark) have a very large number of controls and options, most of which are available from buttons and short-cut keys. The menu system in the dialog provides access to commands that are not available on buttons, and also duplicates many of the other controls, for completeness.

Menu	Command	Setup	New	Edit	Comment
File					
	Load and Save	Yes	Yes	Yes	Load and Save templates
	Sample Data	Yes			Accept current dialog settings, sample
	Abort Sampling	Yes			Close dialog and abandon sampling
	Print...		Yes	Yes	Print current set of templates

	New Channel		Yes		Create a new channel from the input
	Close		Yes	Yes	Close the dialog
Edit					
	Clear All Templates	Yes			Delete all templates
	Copy...		Yes	Yes	Copy current templates as image or text
View					
	Overdraw	Yes	Yes	Yes	Overdraw templates in the template area
	Show Template Limits	Yes	Yes	Yes	Show/hide template limits in template area
	Show Non-Matching	Yes	Yes	Yes	Show non-matched spike on all templates
	Overlay Traces	Yes	Yes	Yes	Overdraw multiple traces or side by side
	Optimise	Yes	Yes	Yes	Optimise the y axis of the display
	Show All Y Range	Yes	Yes	Yes	Show full input range of the signal
	Scale Up/Down	Yes	Yes	Yes	Make templates bigger/smaller
	Pan Left/Right	Yes	Yes		Change the number of pre-trigger points
	Enlarge/Reduce View	Yes	Yes		Change number of displayed data points
Templates					
	Make Templates	Yes	Yes	Yes	Create new templates from spike shapes
	Sound	Yes	Yes	Yes	Play a tone when a spike is displayed
	Renumber	Yes	Yes	Yes	Renumber the templates
	Order by Code	Yes	Yes	Yes	Sort displayed templates into code order
	Initialize Counter	Yes	Yes	Yes	Reset the matched template counters to 0
	Template Size	Yes	Yes	Yes	Small, medium or large template size
	Parameters	Yes	Yes	Yes	Open the Template Parameters dialog
	Form Templates			Yes	Form templates by shape or by code
Control					
	Step Back/Forwards	Yes	Yes	Yes	One spike forwards (back where allowed)
	Run Back/Forwards	Yes	Yes	Yes	Run forwards (backwards where allowed)
	Pause	Yes	Yes	Yes	Stop moving through spikes
	Digital Output	Yes			Online digital output on template match
	High level cursors		Yes	Yes	Use outer cursors to exclude large spikes
	Circular Replay		Yes	Yes	Play data in a loop; don't stop at the end
	Set Time Range...		Yes	Yes	Set time range of data to process
	Jump to Start		Yes	Yes	Jump to the start of the time range
	Track Cursor 0		Yes	Yes	Time view scrolls so Cursor 0 is centred
	Always At End			Yes	Online only, display new spikes
Analyse					
	Principal Components			Yes	Open Principal Components dialog
	Cluster on Measurements			Yes	Open Cluster on Measurements dialog
	Cluster on Correlations			Yes	Open Cluster on Correlations dialog
	Cluster on Errors			Yes	Open Cluster on Errors analysis dialog
	Collision Analysis Mode			Yes	Attempt to separate spike collision mode
	Duplicate			Yes	Make duplicate channel for each template
	Set Codes			Yes	Opens the Set Marker Codes dialog
	Marker Filter		Yes	Yes	Display marker filter dialog

On-line template setup

When you create a New file with WaveMark channels, Spike2 opens the template set up window. The window has its own menu and contains four main regions: a status and tool bar at the top, a data display area with the trigger levels, a control area on the right and a region with up to 20 templates at the bottom.



The display area shows data from the channel set by the channel control. Waveforms that meet the trigger criteria are drawn in the WaveMark 00 colour; untriggered waveforms use the time view waveform colour.

The horizontal dotted lines are the trigger (and limit) cursors; click and drag them to change the levels. You can display 2 or 4 trigger levels. With 4, the outer ones act as limits; to trigger a wave must cross the trigger and not the limit. To display level cursors in one direction only, right-click an unwanted cursor and select Move away. To restore cursors that are off the visible area right click in the display area and select Fetch cursors.

The two black triangles at the top of this area mark the start and end of the region used for templates and principal component analysis.

The vertical bar on the right of the data display shows the number of templates that the program is considering (provisional templates) in grey, with black representing confirmed templates. If the number of confirmed and provisional templates gets large, the provisional template colour becomes red. If this occurs, you may need to adjust the values in the Template Settings dialog to be more tolerant of variation in the spikes.

You can change the window size by dragging any window edge. Double click the title bar of the window to maximise (or minimise if already maximised) the window. The window size can be changed from very small (minimised), with no templates, to large (maximised) with all 20 possible templates (though only the first 8 are used on-line). The templates can be small, medium or large. Double click on one of the template rectangles to change the template size or use the View menu.

Channel control

When there is more than one channel holding WaveMark data, you can change the channel either by dropping down the list of channels and selecting one, or by using the small spin box on the right of the channel control to move to the next channel in the list. Each channel has its own set of templates and template parameters.

Sample data

Once you have adjusted the trigger levels and set the templates for each channel, click this button to close the window and sample data.

Abort
This button closes the window and does not sample data.

Dig out
During sampling, the 1401 can flag template matches using bits 0-7 of the digital port. You can select between None (no output), One bit (single pulsed digital bit per template) or Coded (data channel and template coded in the 8 data bits).

Make templates
With this box checked, detected spikes are offered to the template matching system and will create and modify templates. With this box unchecked, spikes are compared against templates but the templates do not change. When you open this window, this box is checked if there are no existing templates, otherwise it is clear. This command is also available in the Templates menu with a keyboard short-cut of `Ctrl+M`.

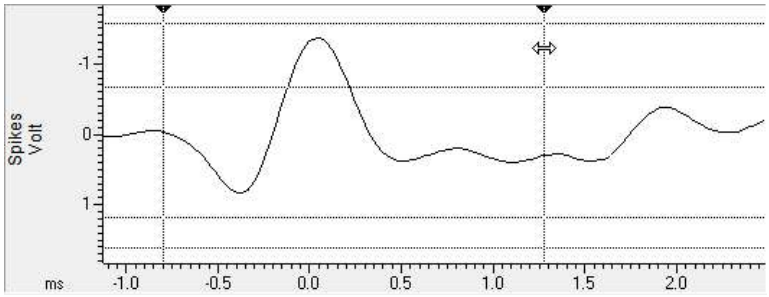
Play controls 
These five buttons control data replay. The centre button pauses replay and the buttons either side of it play the data forwards and backwards. The outer buttons step one spike forwards and backwards. You can only play or step backwards in the Edit WaveMark command. There are keyboard short cuts for these controls. `B` steps backwards one spike, `M` steps forwards one spike, `P` runs forwards, `Q` runs backwards and `V` pauses output. You can also access the Play commands from the Control menu.

Renumber…
This command is available from the Templates menu with a keyboard short-cut of `Ctrl+R`. It prompts you for a starting marker code then renumbers the templates with consecutive codes, starting with the code you set. The templates remain in the original positions and code order. If two or more templates share a marker code they will still share a code after renumbering. For example, templates coded 01, 04, 04, 07 and 03 renumbered to start at 01 would end up with codes 01, 03, 03, 04 and 02.

Order by code
This command is available from the Templates menu with a keyboard short-cut of `Ctrl+B`. It rearranges the templates so that they are in ascending marker code order. For example, templates with marker codes 01, 04, 04, 07 and 03 would be sorted into the order 01, 03, 04, 04 and 07. The sorting order is left to right and top to bottom.

Load and Save…
This command is available from the File menu with a keyboard short-cut of `Ctrl+S`. It opens a new dialog in which you can load templates generated in previous spike sorting sessions and save your templates to resource files.

Selecting the area for a template



The two black triangles at the top of the large display area set the waveform region to use for template formation and for principal component analysis. To adjust this region, click and drag the triangles with the mouse. Vertical cursors appear as guides for the new template position. Old templates are deleted if the size of the template has changed. The vertical cursor positions and the number of data points that will be used in the template appear in the status bar when you activate the vertical cursors.

The best information about the spike class is usually around the peak; the start and end of the spike holds mainly baseline noise. If you use the entire spike shape for template matching, you will be comparing baseline noise for some of the record, which is a waste of time and reduces the sensitivity of the template matching.

If your spike fills a small proportion of the spike trace (as in the example above), you should reduce the number of points for the WaveMark channel with the button or by dragging the x axis. This has several advantages: reduced disk space for storage, faster processing, quicker drawing of data and fewer records with a second spike overlapping the end.

If you have too few points around the interesting region of your spike you will need to increase the sampling rate. Remember that the template routines interpolate between data points, so you only require enough points to define the spike shape.

You will get the best spike discrimination when the area selected shows the greatest variation between different classes of spikes. For example, if all your spike classes look very similar at the end, you will improve discrimination by excluding the end of the data.

Horizontal cursors

There are horizontal cursors in the Spike shape dialogs except in the case of the off-line Edit WaveMark dialog. There is a choice of 2 cursors (setting upper and lower trigger levels) or 4 cursors (setting amplitude limits and trigger levels) controlled by a toolbar button. You can drag these cursors; when you drag, the cursor positions appear in the status bar at the top of the dialog.

Linked cursors

You can link the horizontal cursors to horizontal cursors 1 to 4 in the associated time view. When linked, horizontal cursor 1 is the lower trigger level, 2 is the upper trigger, 3 is the lower limit and 4 is the upper limit. The option is checked when linked. When you set this option, cursors corresponding to the visible cursors in the spike shape dialog are moved to (or created in) the same positions in the source time view channel.

When cursors are linked, each time you release a cursor after dragging with the mouse, the linked cursor will also move. As you can drag time view cursors to any position or channel, only moves to sensible positions on the same channel will have any effect on the spike shape window.

Any horizontal cursors in the time view that did not exist before the link are considered to be owned by the spike shape dialog and will be deleted if the spike shape dialog does not need them. If you change channel in the spike shape dialog while linked, all owned cursors are deleted from the time view, then the dialog behaves as if you had requested a link command for the new channel.

Caveats

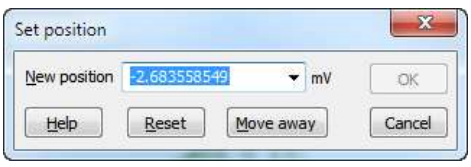
For this feature to be useful, the source channel must have the same DC offset as the spikes you see in the Spike shape window (usually 0). This will be the case for WaveMark source channels (for example when sampling data), but need not be the case with Waveform or RealWave data. Further, the mean level of the spike data must be the same as the DC offset (usually 0). If the source data does not have the same DC offset, the cursors will still work and the values will match, but the source data will be vertically shifted. You can often work around this by applying a DC remove process to the source channel. This is an experimental feature and we do not store the status of the link when the spike shape dialog closed.

Context menu for the cursor area

There is a context menu (right-click) for the data display area to control the horizontal cursors. The first three context menu items only appear if you click on a cursor:

Set cursor position...

This opens the Set position dialog. Click Reset to restore the original position. Move away closes the dialog and moves the cursor to a position where it will have no effect on spike capture. The drop down list contains the original cursor position and also allows you to select horizontal cursor positions in any associated time view(s). OK is enabled if the new position is legal and differs from the original.



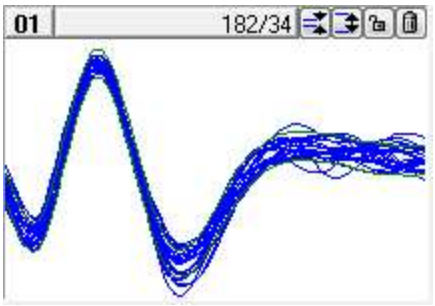
Copy position 2.345 mV
Select this option to copy the current cursor position and units to the clipboard as text.

Move away
Move the cursor to a position where it has no effect on spike triggering. If you have enabled the amplitude limit cursors and you use the Move away command on a trigger level, both the trigger and associated amplitude cursor will move away. If you move away a linked cursor the time view linked cursor will be deleted if it was created by linking, otherwise it will move out of the way.

Fetch cursors
This option appears if you right-click in the WaveMark display and a horizontal cursor is beyond the displayed y axis range. This command moves the cursor back into a visible range. If there are multiple traces, only cursors in the trace on which you click are restored. If you are linked to a time view, fetched cursors will become visible in the time view.

Link Time view cursors
This option links and unlinks the visible horizontal cursors in the spike shape dialog to horizontal cursors 1 to 4 in the time view channel that sources the data. This option is disabled if the Cursor menu is disabled during script operation.

The template area



Templates appear at the bottom of the window. If this area is not visible, drag the bottom edge of the dialog downward until the template area appears. Templates can be displayed in three sizes: small, large or huge; double click a template to cycle round the three sizes. The image shows a medium sized template; small templates have less information at the top. The template code is at the top left of the template area. When a template first appears, it has the lowest unused code. Double click the template code to open a dialog where you can change the code. You are allowed to have multiple templates with the same code.

Event count
The count of spikes that matched the template is in the centre. In large and huge mode, the display also shows the number of spikes added to the template as: matched/added. You can set the count of matched events to 0 with the Template menu Initialise Counters command; the short-cut key is `Ctrl+I`.

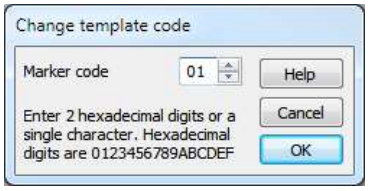
The display area holds the last spike that matched the template, the template itself and optionally all non-matching spikes. You control the drawing style of the matching and non-matching spikes and the template with buttons at the left-hand end of the toolbar. These buttons can appear at the top of each template area:

Change width 
These buttons are hidden in small template windows. They decrease and increase the width of the template. The wider a template, the more spikes can be deemed to fit (unless the constraint that a certain percentage of the points must fall within the template limits is removed).

Lock 
This button locks or unlocks a template. A locked template does not change when new spikes match it. An unlocked template adapts as each new spike is added. Templates can lock automatically if Auto Fix mode is set in the template settings dialog. When building templates, Spike2 cannot merge a provisional template with a locked template. Instead, Spike2 creates a new template with the same code as the one it would have merged with.

Clear
This button erases the template. To erase all templates, click the bin icon in the toolbar.

Edit template code



You can open this dialog by double-clicking a template code at the top-left of a template or one of the four template codes on the right-hand side of the spike sorting dialog.

Double-digit codes are read as hexadecimal marker codes, single characters are read as printing characters. If you have a spike that changes shape too much for one template to represent it, let it generate several templates and then edit the codes to be the same. You can use code 00 to mark spikes or artefacts you are not interested in.

Merging templates

You may decide that two templates are so similar that you would like to merge them into one. To do this, drag one of the templates over the one that you wish to merge it with and release the mouse button. The dragged template vanishes and the template over which you released it changes to take account of the new data.

The mouse pointer is an open hand when it is over a spike or template that you can drag. To drag, click and hold the left mouse button; the pointer changes to a closed hand. When you drag over a drop zone, the closed hand has a plus sign on the back. You can compare templates by dragging, but not releasing the mouse. If you decide not to merge, make sure the mouse pointer is not the closed hand with the plus sign when you release the mouse button.

Manual template creation

When paused, you can use drag and drop to create new templates and to classify spikes. Click the mouse in the data display area and drag the current spike to an existing template to set the marker code, or to an unused template to create a new template.

You can also use the keyboard to do the same thing: The keys `1` to `9` will match the current spike to the first template with the code 01 to 09, or if there is no template with a matching code, a new template will be created with that code. The `0` key can be used when editing spikes to give the current spike the code 00.

Toolbar controls

The toolbar buttons can be clicked with the mouse to control the template forming process. The leftmost group are state buttons; click them to set the state, click them again to remove the state. The remaining buttons act immediately. These commands are also available in the dialog menus and most have short-cut keys. The buttons are:

Overdraw
Normally, the templates display the last spike added. You can choose to show all spikes added to templates by overdrawing. This omits the step of erasing the previous spike before drawing a new one. The short-cut key is `Ctrl+O`.

Template Limits (shape)
You can choose to display the mean template, or the upped and lower limits of the template. The template limits are useful when you display the last spike and need to see how well it fits within the template. The mean template can be useful when you are comparing template shapes. The short-cut key is `Ctrl+L`.

Show non-matching spikes

In some circumstances you need to see every spike displayed over every template. With this button down, all non-matching spikes are drawn on top of the template in the "Not saving to disk" colour. The short-cut key is `Ctrl+N`.

Sound on/off

It can be useful to have an audible indication of each spike as it is added to a template. If you have a sound card, turn this on to hear a tone for each spike. A different tone is played for each template code; the tone is not related to the spike shape. A low-pitched tone is played for a spike that matches no template, and a high-pitched tone for spikes that match templates numbered higher than 20. The short-cut key is `Ctrl+U`.

If you have no sound card the effect is operating system dependent (there may be no sound at all on some systems). With a sound card, there are limits on the maximum spike rate imposed by the operating system and the fact that sounds have to last much longer that the spikes for you to hear the tones!

Scroll time view

If you depress this button, the associated time window scrolls to keep the current spike centred, if this is possible. The short-cut key for this is `Ctrl+K`.

Automatic levels

This uses information collected from recent spikes to set the trigger and limit levels and to scale the data. This gives a good starting point for manual adjustment. The display scaling applies to both the data window and to the templates. The short-cut key is `Home`.

Details: Spike2 tracks the maximum excursion from zero in every displayed spike. If a new spike has a smaller maximum than the current maximum value, the current maximum is multiplied by 0.98. If the new spike has a larger maximum, it sets the new current maximum. The trigger levels are set at 0.7 and -0.7 of the current maximum. If you use this command in the New WaveMark dialog where you can have 4 horizontal cursors, the outer two levels are set to the current maximum and minus the current maximum. The Y scale is set to display from +1.4 times the current maximum to -1.4 times the current maximum.

Scale data

These two buttons increase and decrease the display size of the waveforms and the templates. If you click on a button, the scale changes once. If you hold one down, the scale changes repeatedly until you release the button. You can also scale the data by clicking and dragging the y axis. The short-cut keys are `Up` arrow and `Down` arrow.

Scroll data

These buttons scroll the data in the large window sideways. This changes the pre-trigger time, and is equivalent to changing the Pre-Trigger field of the Channel Parameters dialog for WaveMark data. If possible, the two triangles marking the start and end of the template region also scroll to preserve the template region. You can also click and drag the data x axis ticks to scroll the window or use the `Left` and `Right` arrow keys.

Points

These buttons increase and decrease the number of points saved for each WaveMark. This is the same as the Channel Parameters dialog Points field. You can also use the `PgUp` and `PgDn` keys or click and drag the x axis numbers to scale the view around the 0 ms point. You should save as few points as possible to minimise the file size.

Peak between cursors

Depress this button to display and use the limit cursors. Spikes with peak amplitudes at the trigger point that lie outside the limits are ignored. The short-cut key is `Ctrl+H`. You can use limits when creating new WaveMark

channels from waveform data offline. If you have a Power1401 or a Micro1401-3 or -4, you can also use limits while sampling data. If you sample with a 1401 that cannot use limits, this button is hidden.

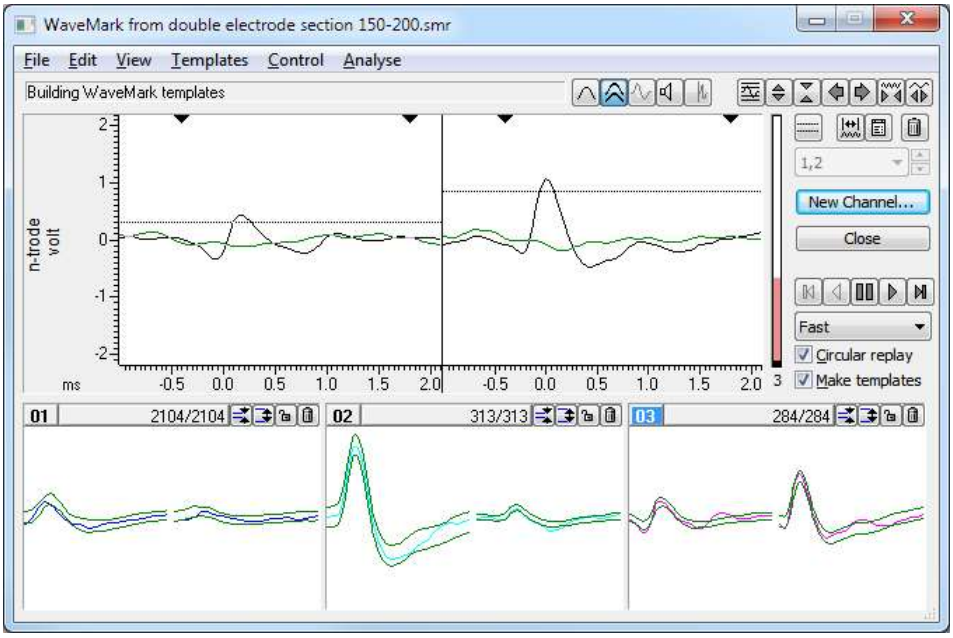
Clear templates

This button clears all templates (and any tentative templates not yet confirmed). Think of throwing the templates in the bin. The short-cut key is `Del`. You can clear individual templates using the clear button on the top bar of each template.

Template parameters

This button opens a dialog that gives you control over template generation. To get the best use of the items in the dialog you will need to understand how templates are constructed and generated,. The short-cut key is `Ctrl+Enter`.

Multiple traces



All the spike shape dialogs can handle WaveMark data with 1, 2 or 4 traces (single electrode, stereotrode and tetrode data). The documentation concentrates on the single trace case; this section explains the differences when you have multiple traces.

The data display is divided up horizontally so that each trace has its own rectangular area. The dialogs that allow you to set trigger and limit levels have independent horizontal cursors for each trace. The x axis and the triangles that mark the start and end of the template region are duplicated for each trace and are slaved together; if you drag the x axis or the template markers in one trace region, all traces change.

The templates are also divided up horizontally, however you can choose to overdraw the traces (perhaps to compare latencies) with the View menu Overlay templates command.

Spike sorting and template matching work in much the same way for multiple traces as for a single trace. The template parameters apply to each trace and all traces must meet the matching criteria for a spike to match a template. Because of this, you may need to relax some of the matching criteria compared to spike sorting with a single trace.

Forming n-trode data from waveform channels

In addition to sampling data with 2 or 4 traces, you can create WaveMark data from 2 or 4 existing waveform channels that have the same sampling rate. To do this select the channels in the time view that you wish to convert by clicking on the channel number, then use the Analysis menu New n-trode command or right click on one of the selected channels and select the New Stereotrode or New Tetrode command. This will open the New WaveMark dialog with the selected channels set as the data source

In the dialog, the channel selector is disabled and displays the list of channels that correspond to the traces. Channels are used in the order that you selected them.

Triggering with multiple traces

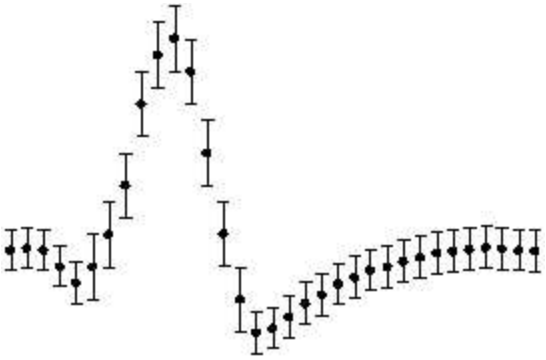
Each trace is considered separately when searching for a trigger condition. If a trigger is confirmed on one trace and another trace finds a trigger condition that is more than 4 data points after the first, it will cause a second spike to be recorded. We would expect the same spike recorded on different (but physically close) electrodes to be pretty much simultaneous, and certainly within 4 samples.

If you want to suppress overlapping triggers from a different trace, check the Disable N-trode independent triggers option (new in version [10.01]) in the Template settings dialog. If you do this, new trigger positions are ignored within the spike data that is saved for the previous trigger. Captured spikes can still overlap, but the maximum overlap is the number of pre-trigger points set for the channel, which is the same as for a spike shape with a single trace.

Template formation

A template is stored as a series of data points. Each point has a width and a minimum width associated with it. The width represents the expected error in a signal that matches the template at that point. Spikes match a template if more than a certain percentage of the points in a spike fall within the template. It is easy to calculate an appropriate template width when you have a population of spikes of the same class. The problem comes when you have a single spike that starts a new template. What width do you give each point?

We set the initial template width to a percentage of the maximum positive or negative amplitude of the spike. The percentage to use depends on the data, 35% is a reasonable first guess. This value also sets the minimum allowed spike width. Experience shows that unless a minimum width is set, the template width tends to reduce as we only add matching spike. We also limit the sum of the differences between a spike and the mean template.



The template width is twice the mean distance between the template and the spikes that created it, but is not allowed to become less than the minimum width. If the variation in spike shape around the mean were normally distributed, you would expect 80% of the data in a matching spike to lie within the template.

When forming templates, the *Minimum percentage of points in template* rule applies, meaning that we do not consider a template if less than the user-specified percentage of data points lie within the template boundaries (defined as the mean template plus or minus the width at each point). This rule can be turned off when not forming templates and is turned off when forming templates if all provisional template space is exhausted.

The template building algorithm works as follows:

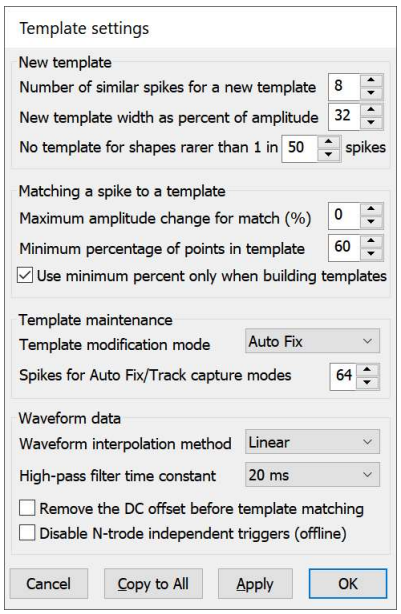
1. The first spike forms the first template with the template width estimated. This is a provisional template. Provisional templates are not displayed.
2. Each new spike is compared against each template in turn to see if sufficient points fall within the template. If there are any *confirmed* templates, these are considered first, and if a match is found, the provisional templates are not considered.
3. If a spike could only belong to one template, it is added to it. Otherwise, the spike is added to the template that is the minimum distance from the spike. Distance is defined as the sum of the differences between the spike and the template.
4. If a spike belongs to no templates, a new provisional template is created. If we have run out of provisional templates, we remove the *Minimum percentage of points in template* rule to give spikes more chance to match one of the provisional templates.
5. Adding a spike to a template changes the template shape and width. Once a pre-set number of spikes have been added to a template, it is checked against existing confirmed templates to prevent generation of two templates for the same data. If no match is found, the template is promoted to confirmed. If a match is

found, the provisional template is merged with the confirmed one unless the confirmed template is locked, in which case a new confirmed template is created *with the same code as the one it matched*. Each time a spike is added to a provisional template, the provisional template *decay count* is reset.

6. Each time a spike is added to a template, all provisional templates have their decay count reduced by 1. If any decay count reaches 0, that template has 1 spike removed from its spike count and the decay count is reset. If the number of spikes in a provisional template reaches 0, the provisional template is deleted.

Step 2 is complex. To compare a spike against a template we shift it up to 2 sample points in either direction to find the best alignment point. Shifts of a fraction of a point are done by interpolation. If you have enabled amplitude scaling, the spike is scaled to make the area under the spike equal to the area under the template (limited by the maximum scaling allowed). Amplitude scaling is only allowed for confirmed templates. As interpolation is expensive, templates also store information to decide if it is worth attempting shifting to fit a spike to a template.

Template settings dialog



The dialog controls the formation of new spike templates and how spike match templates. Each channel has an independent set of parameters. If you use the Channel control in the parent spike shape dialog to change data channel, the template parameters change to match the values for the new channel; changes made since the last Apply or Copy to All will be lost. The buttons at the bottom of the dialog are:

Copy to All
Use this button when you have multiple channels of WaveMark data and you wish to apply the current template parameters to all of the channels. The action is the same as Apply, but it sets the value for all channels, not just for the current channel.

Apply and OK
These do the same thing, but OK closes the dialog and Apply leaves it open. The settings you edit in the dialog make no difference to the template formation process unless you use one of the Copy to All, Apply or OK buttons. You cannot use these buttons if any field in the dialog holds invalid data.

Cancel
This button closes the dialog without applying new changes. Changes made by the Apply or Copy to All buttons are preserved.

The dialog has 4 areas:

New Template
This controls the creation of new templates.

Number of spikes for a new template
This sets the number of spikes at which a provisional template is promoted to a real one. We find that 8 is a reasonable starting value.

New template width as a percentage of amplitude
This is the percentage of the spike amplitude to give the initial (and minimum) template width. As spikes are added to the template, the width changes to represent the variation in amplitude of the spikes in the template. The maximum template width is set to 4 times the minimum width. A value of 30% is a reasonable starting value.

No template for shapes rarer than 1 in n spikes
If this is n, this is roughly the same as saying that you are not interested in spike classes which occur less often than once every n total spikes. If you want to keep all spikes as potential templates you should set this to a large number. We find that 50 is a reasonable starting value. When discussing the template formation algorithm, we

call this the *decay count* as provisional templates can be thought of as decaying if they do not get spikes added to them at reasonable intervals.

Matching a spike to a template

The items in this group are used when comparing a spike with the existing templates to determine if the spike is the same or should start a new provisional shape. Unless amplitude scaling is enabled, to avoid wasting time interpolating spike shapes for data that could never fit a template, if the error between a spike and a template exceeds a value calculated from the template, the spike is rejected before applying the following:

Maximum percent amplitude change for match

Spike2 will scale spikes up or down by up to this percentage to make the area under the spike the same as the area under each target template. This is very useful if you have spikes that maintain shape but change amplitude. Do not set this non-zero on-line unless you need to as it slows down the template matching process. The maximum change permitted is 100%, which allows a spike to be doubled or halved in amplitude. Set this to 0 unless you need it. The value you set depends on the amplitude variability of your spikes; 25% is a reasonable starting value.

Minimum percentage of points in template

The percentage of a spike that must lie within a template for a match. If more than one template passes this test, spikes are matched to the template with the smallest error between the mean template and the (scaled) spike. 60% is a reasonable starting value. With amplitude scaling on, you may want to increase this to 70% or more.

Use minimum percentage only when building templates

The template width is most useful in the set up phase when we are looking carefully at differences between spikes. Once templates are established it can sometimes be better to match to the template with the smallest error and ignore the width. If you check this field, this sets the percentage of points that must lie in the template to 0% unless you are building templates.

Template maintenance

This group of fields determines how the shape of a template changes as more spikes are added to it. On-line, the template shape is fixed in all modes apart from Track. From our experience, Add All and Auto Fix are the most useful modes.

Template modification mode

Add All — All spikes that fit the template are added and modify the template. The effect of each spike becomes smaller as the number of matched spikes gets larger.

Auto Fix — The template is fixed once a set number of spikes have been added. If you have several similar spikes, using Auto Fix after a fairly small number of spikes can stop a template gradually changing shape and becoming the same as another template.

Track — The template shape tracks the spikes. The contribution of each spike to the template decays as more spikes are added. This is only really useful for slow changes in spike shape and always brings with it the danger than all your shapes will merge together. It also slows down on-line spike classification.

Spikes for Auto Fix/Track capture modes

This sets the number of spikes for the previous field in Auto Fix and Track modes. In Track mode, the smaller the number, the more rapidly the template shape changes.

Waveform data

The final group of fields control how the raw waveform data is processed into spikes:

Waveform interpolation method

Spike waveforms are shifted by fractions of a sampling interval to align them with templates using linear, parabolic or cubic spline interpolation. Linear interpolation is the fastest, cubic spline interpolation is the slowest. The on-line 1401 code always uses linear interpolation for speed reasons. For off-line work, speed of computation does not matter.

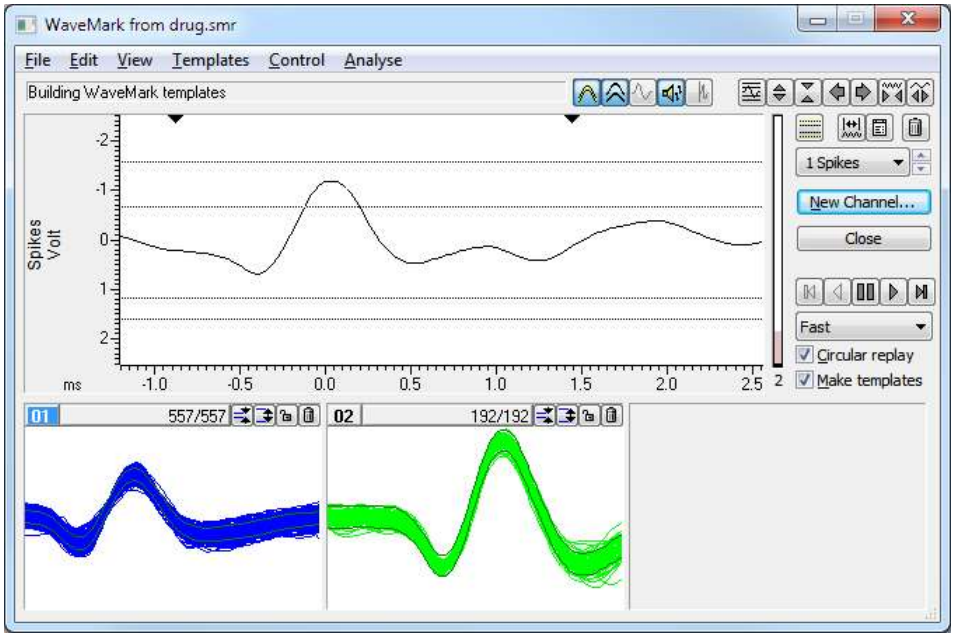
High-pass filter time constant
Use this to remove baseline drift. Set this to a few times the width of the spikes. Do not set the value too low or it will significantly change the spike shapes. If your signal has abrupt baseline changes, you may get better results with the DC offset option.

Remove the DC offset before template matching
Check this field to subtract the mean level from each spike before matching. This effects template formation and matching, not the saved data. Unless your baseline has sudden DC shifts, it is usually better to use the high-pass filter to follow signals that drift. There is a small time penalty for using DC offset removal on-line.

Disable N-trode independent triggers (offline) New at version [10.01]
Normally, when dealing with Stereotrode or Tetrode data, for triggering purposes, all channels are treated independently with the proviso that if two triggers occur within 4 samples of each other, only the first is used. This can lead to highly overlapped traces due to triggering events being detected on separate traces. If you check this box, we ignore new trigger events that occur within the last captured spike shape. Events can still overlap, but only by the pre-trigger time of the spike. This is implemented only for offline sorting when creating new stereotrode or tetrode channels.

Off-line template formation

The Analysis menu New WaveMark... command extracts WaveMark data from waveform channels (and also from existing WaveMark data). If you wish to edit WaveMark data, see the *Off-line template editing* section.



This window is almost identical to the on-line template set up window. There are more menu items and buttons in the control bar, some button labels have changed and there are new controls to set the spike replay rate and you can print and copy template data. You can use the horizontal cursors to set the trigger levels and limit spike amplitude.

The program scans the data in the channel set by the channel control (wrapping round to the start of the data when it reaches the end if Circular replay is checked), and displays triggered and non-triggered events, as if the data were being sampled.

Replay rate
This control sets the maximum number of spikes per second to process when running continuously. With Fast selected, the spikes are processed as fast as possible; you can also set this by pressing the F key (F for Fast). With Real time selected, spikes are replayed no faster than real time; you can set this with R for Real time. You can also select slower rates in Hz; the S for Slow key selects a medium-slow rate.

New Channel...
Once you are satisfied with your templates, click the New Channel... button to open a dialog in which you can select the channel to write to and the new data type.

Time range 
This button opens a dialog in which you set the time range of data to process. You can either type in the start and end times, or select values from the drop down list. When you open the window for the first time, the time range is set to the whole file.

Cursor 0
While this window is open, cursor 0 is added to the time view associated with the data file. This cursor marks the position of the currently displayed trace. When paused, you can drag this cursor to a new position, which will trigger a search for a trigger condition. The special cursor does not appear in Cursor windows. The `Ctrl+J` key combination jumps Cursor 0 to the start of the time range and starts a search for a trigger condition. From version [10.01], if you drag cursor 0 in the associated time view, this triggers a spike search from the trigger position, but we suppress the cursor 0 reposition until you release the mouse button to end the cursor 0 drag. Prior to version [10.01], cursor 0 would flicker between the dragged position and the position of the found trigger, which was visually unpleasant and could cause strange display effects if the Scroll time view option was enabled.

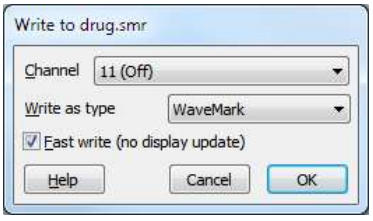
Several other Spike2 processes can drive cursor 0: Measurement processing, Active cursor searches, Offline Waveform output and optionally by multimedia replay. From version [10.01], when any process starts driving cursor 0, other drivers are signalled to stop driving it. Note that changes to the Cursor 0 position will cause updates to the Cursor Regions and Cursor Values dialogs if they are open; changes to these dialogs (especially the Cursor Regions) can take a noticeable time.

Circular replay
This check box enables circular continuous replay. If you clear this box, continuous replay stops when it reaches the end of the time range set for analysis. The short-cut key for this is `Ctrl+Q`.

Scroll time view 
If you depress this button, the associated time window scrolls to keep the current spike centred, if this is possible. The short-cut key for this is `Ctrl+K`.

Marker filter
The Analysis menu Marker filter command (short-cut key `Ctrl+F`) is enabled when the source channel holds WaveMark data. It opens the Marker Filter dialog, or brings it to the front if it is already open. Whenever you change channel to a channel holding WaveMark data with the Marker filter dialog open, the Marker filter dialog automatically changes channel to match.

Create a New spike channel



This dialog opens when you click the New Channel button or select the New Channel command from the Spike shape dialog File menu. You must select the channel to write to; the dialog will offer you the lowest numbered unused disk channel, but you can choose to overwrite an existing channel. The Fast write check box suppresses display updates whilst generating the new channel, to save time. You can also choose the channel type to write the data in (select WaveMark if you are unsure):

Event-, Event+ If you use this format, only the time stamp for each detected event is saved. The new channel has no template matching or waveform information.

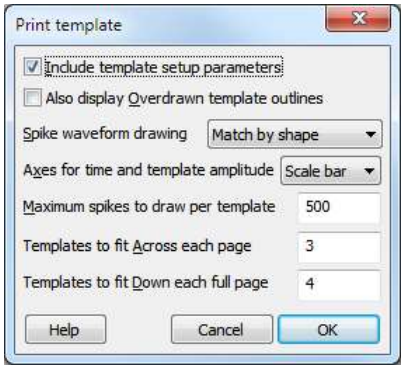
Marker This saves the template matching information and the time stamp, but does not save any waveform information.

WaveMark — You will usually use this format to save a time stamp, a marker code and the waveform for each detected event.

When you choose OK, the selected time range of data is analysed, and any events that cross the trigger levels are written to the new channel in the selected format. Options that change the number of points saved per WaveMark are disabled during analysis.

The title of the new channel is set to `nw-c` where `c` is the original channel number. The channel comment for the new channel is set to indicate the source of the data.

Print templates and Copy



The File menu Print… command (short-cut key `Ctrl+P`) opens the Print template dialog. The output format depends on the printer page size (you can set the print margins in the main application File menu Page setup dialog). You can choose how many templates to draw across and down each page.

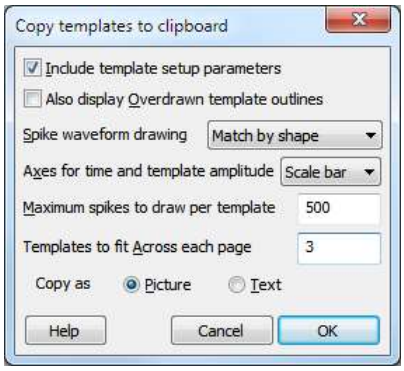
You can include the template parameters, and also an extra picture showing all of the template outlines overdrawn for easy comparison.

The Spike waveform drawing field chooses how spikes are included in the result. Set it to None for no spikes or to the method for selecting which spikes are drawn with a template. You can choose Match by shape or Match by code (only in the Edit WaveMark dialog). Spikes are drawn in the best-fit position to the template. You can limit the number of spikes to draw in each template; some printers cannot cope with huge numbers of lines in the printed output.

The Axes for time and template amplitude field lets you choose between None (no axes drawn), Scale bar (an indication of size) and Full (a full axis with big and small ticks). The y axis range is the same as the y axis range in the Spike shape dialog. The example image below is using a scale bar; this is the default setting.

If you have set a time range to analyse, the overdraw spikes are taken from the time range. If you have not set a time range, the spikes are taken from the start of the file.

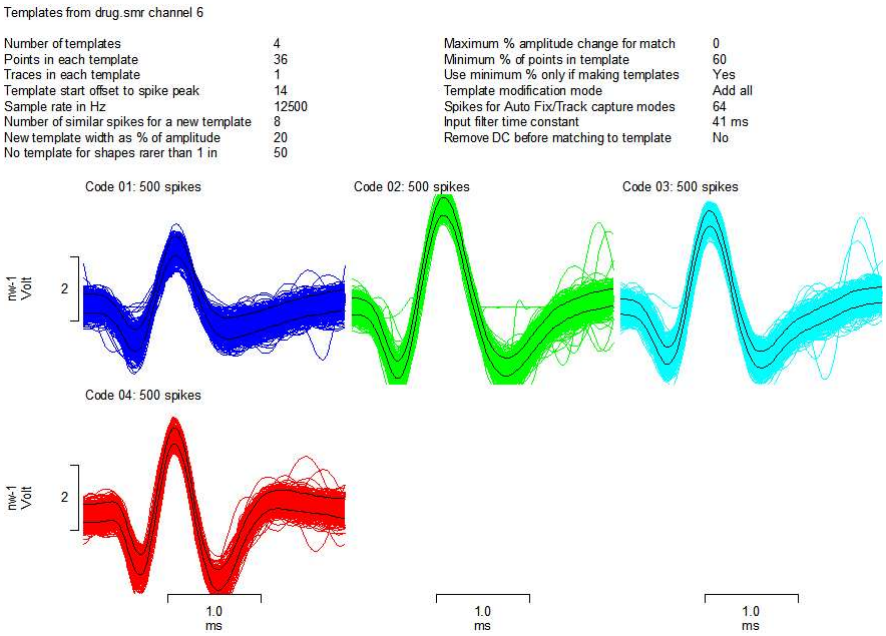
Copy to clipboard



The Edit menu Copy command (short-cut key `Ctrl+C`) opens the Copy templates to clipboard dialog, which is very similar to the Print templates dialog. There are radio buttons to choose between Picture and Text output.

Typical clipboard output

Picture output is in Enhanced Metafile format and can be pasted into documents in most drawing programs and many word processors. Some bitmap-editing programs (for example, the Paint program that comes as part of Windows XP) will accept this format and convert it into a bitmap. If you want to import data into a drawing program you may need to restrict the number of spikes displayed per template as not all drawing programs cannot cope with a large number of vectors in an image.



Text output optionally includes the parameters. The templates are output in vertical columns separated by Tab characters. There is one column or two columns per template, depending on the template shape display mode (mean templates or upper and lower limits). The first row of template output holds column titles, in quotation marks. The output is designed to paste easily into spreadsheet programs. This example is showing the upper and lower template levels (it is displayed in a table to demonstrate the structure of the output).

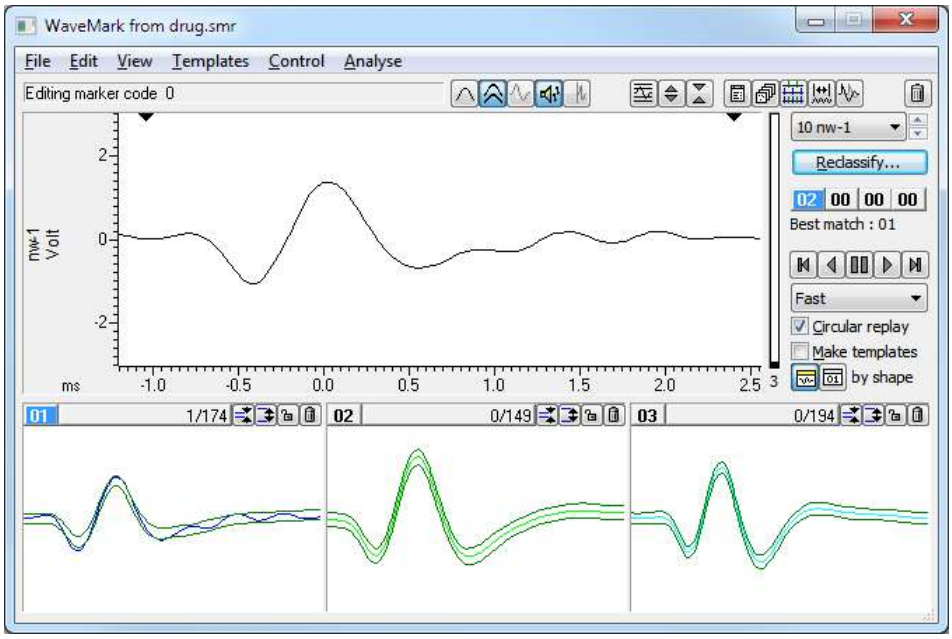
"Number of templates"	4
"Points in each template"	36
"Traces in each template"	1
"Template start offset to spike peak"	14
"Sample rate in Hz"	12500
"Number of similar spikes for a new template"	8
"New template width as % of amplitude"	20
"No template for shapes rarer than 1 in"	50
"Maximum % amplitude change for match"	0
"Minimum % of points in template"	60
"Use mimimum % only when making templates"	"Yes"
"Template modification mode"	"Add all"
"Spikes for Auto Fix/Track capture modes"	64
"Input filter time constant"	"41 ms"
"Remove DC before matching to template"	"No"

"Code 01 upper"	"Code 01 lower"	"Code 02 upper"	"Code 02 lower"	"Code 03 upper"	"Code 03 lower"	"Code 04 upper"	"Code 04 lower"
0.263062	-0.12420	0.189056	-0.15548	0.169983	-0.14434	0.238342	-0.13397
0.266724	-0.12085	0.170135	-0.16952	0.154877	-0.15945	0.236664	-0.13412
0.255432	-0.12878	0.120239	-0.21911	0.114899	-0.20065	0.249329	-0.13092
0.206909	-0.16540	-0.05386	-0.39321	-0.008544	-0.32226	0.2771	-0.110779
0.08255	-0.28121	-0.41549	-0.75485	-0.258636	-0.58090	0.294037	-0.093841
-0.130463	-0.50918	-0.85144	-1.20117	-0.576324	-0.91323	0.147552	-0.237885
-0.359955	-0.77896	-1.09451	-1.45981	-0.804749	-1.1499	-0.25009	-0.673676
-0.461731	-0.90942	-0.94528	-1.30295	-0.761566	-1.08292	-0.69107	-1.12625
-0.326385	-0.73837	-0.38452	-0.72601	-0.323792	-0.63781	-0.76080	-1.15631
0.0418091	-0.27069	0.437317	0.097961	0.41214	0.098419	-0.25512	-0.617981
0.57251	0.313416	1.27914	0.939789	1.20239	0.888672	0.605011	0.264435
1.11221	0.782623	1.91879	1.57883	1.77475	1.46072	1.39893	1.0788
1.41571	1.02417	2.21878	1.866	1.96671	1.65298	1.79184	1.46744
1.37802	0.992279	2.14111	1.79016	1.74911	1.43539	1.63803	1.30844
1.04614	0.726929	1.74225	1.40228	1.21918	0.905457	1.02386	0.713501
0.607147	0.328827	1.15738	0.818024	0.542755	0.229034	0.198975	-0.076904
0.232697	-0.06637	0.500793	0.161438	-0.102997	-0.41671	-0.52581	-0.827942
-0.027160	-0.37445	-0.09979	-0.43914	-0.590973	-0.90805	-0.96420	-1.33621
-0.164642	-0.55648	-0.57418	-0.91354	-0.852814	-1.18149	-1.10382	-1.51031
-0.215607	-0.63247	-0.88302	-1.22665	-0.906372	-1.24695	-0.98861	-1.39847

Off-line template editing

The Analysis menu Edit WaveMark... command opens a dialog in which you reclassify data in a WaveMark channel. This dialog has its own menu system that includes links to Principal Component Analysis and clustering. If the selected channel has a marker filter set, you view and edit only those events that match the filter specification. The window controls are the same as for *Off-line template formation*, except that you cannot

change the number of points in the main window and there are no horizontal cursors (use New WaveMark if you need trigger levels).



Marker codes 

These four codes show the four marker codes of the current spike. Normally only the first code is used for spike classification and the others are 00. If you click on a code, it becomes highlighted and sets the code that is changed by reclassification. If you double-click a code a dialog box opens in which you can edit the code. The Best match code is the code of the template that best matches the current spike or 00 if no template matches. If you use this dialog on-line only the leftmost button is enabled.

Template formation method 

You can build templates by shape or by code. If you build by shape, templates are formed and spikes are sorted based on shapes; prior classification is ignored. If you build by code, the shapes are ignored, and the classification is purely by the code highlighted in the Marker codes. Templates are built automatically when the play controls are set to play forward or backward through the data and when the Make templates box is checked.

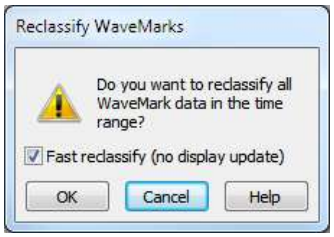
Manual reclassification

When replay is stopped, you can reclassify a spike by dragging the raw data and dropping it on a template or by double clicking a Marker code and editing. You can also reclassify by pressing keys 0 to 9 to give a spike template marker codes 00 though 09. If you use keys 1 through 9 and no template exists with the code, a new template with the code is created based on the current spike.

Automatic reclassification 

Click the Reclassify... button to open the Reclassify WaveMark dialog to reclassify all the spikes in the current time range based on the templates.

Reclassify WaveMarks

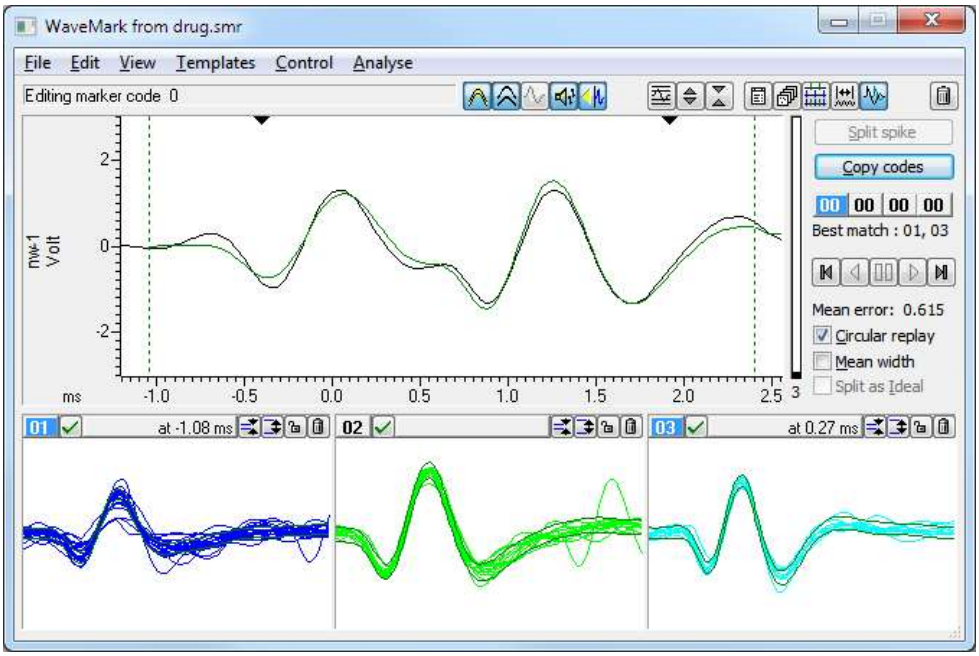


This option reclassifies all the events (or filtered events if a marker filter is set) on the channel in the time range by matching them to the templates. You can set fast reclassification (no display) or a slower mode where each event is drawn in the data display area as it is reclassified. You can stop the reclassification before all spikes have been scanned, but you cannot undo it.

Tip: clear the Make templates check box before reclassifying to stop Spike creating extra templates and modifying existing ones.

Collision Analysis Mode

If a channel contains more than one class of spikes, and the spikes are independent, it is inevitable that there will be collisions. A simplistic analysis shows that if two independent spikes have widths w_1 and w_2 and mean firing rates r_1 and r_2, we would expect $r_1*r_2*(w_1+w_2)$ collisions per second. If both rates are 10 Hz and both widths are 2.5 ms, you would expect 1 collision every 2 seconds. If we can assume that the resulting waveform is the sum of two spike waveforms, and that these waveforms are similar to the established templates, we can search for the template pair whose sum is most similar to the collision.



To enter or leave Collision Analysis Mode, use the Analysis menu command or click on the collision analysis button at the right-hand end of the toolbar. You will notice that some of the controls on the right of the window change, as does the main data display:

Main display

The main display shows the current spike overlaid with the best match combination of templates. The dashed vertical lines show the start and end of the template-forming region. The two black triangles mark the start and end of the *Important area*.

Important area

You can change the important area by clicking and dragging the black triangles. If the area is smaller than the templates (the usual case), it is always included in the best match. If it is more than twice the size of the templates, the best match area will lie within it. In an intermediate case, it marks the area we would like to match and some or all of the best match will lie in the region.

Template display area 

The button to the right of the template code excludes the template from the matching process. You might want to do this if a collision is too close to another spike of the same class. You can exclude all but 1 template. Templates used to make a best match have the template code highlighted and list the offset in milliseconds into the main display at which the template starts.

Best match

This lists the template code or codes that were used to make the best match trace in the data display. The code for the earlier template is listed first. Templates are allowed to overlap the start and end of the displayed spike when making a match, but at least half the template points must be used in the match.

Copy codes

Click this button to copy the *Best match* codes to the current spike. The first code is applied to the currently selected marker code. If there are two codes, the second is applied to the next marker code to the right (wrapping round to the first if necessary).

Mean error

This field displays the mean square error per point. The error is scaled by either the template width at each point, or by the mean template width over all the templates. See the following description of *Mean width*.

Mean width

If you check this box, the errors are scaled by the mean width of all the templates (including excluded templates). This gives all points the same importance when calculating the best template combination. Think of this as "least squares" fitting. If you do not check the box, the template width is used point by point to scale the error. This gives points with a small template width more importance than points with a larger template width. Think of this as "Chi-squared" fitting.

If your templates have a wide variety of widths, you may get better results with this box checked (or at least results that are more what you expect).

Split spike

This button is enabled if the spike source is a memory channel and the current spike is worth separating. Worth separating means that the spike matches two templates, or it matches a single template that doesn't overlap the x axis zero (it matches a template that lies entirely before or after the peak in the raw data).

This replaces the current spike with spikes aligned to the best match templates. How the replacements are calculated depends on the *Split as Ideal* check box. Each matching template generates one output spike. If the best match doesn't include the x axis zero, the original spike (less the best match waveform) is also preserved and *Split as Ideal* is assumed to be checked.

Split as Ideal

If this is checked (or assumed checked), the replacements waveforms are the template shapes. If this is not checked, the replacements share half the difference between the original data and the best match so that the sum of the two spikes recreates the original where the two created spikes overlap. In both cases, any data that is not available from the original data is set to zero.

Caveats

It is important that the templates are centred vertically on zero. If this is not the case, adding templates will create offsets, and the results will not be useful. Likewise, each template should start and end around zero. If they don't, you probably have not selected enough data when forming the templates. If templates start and/or end with a substantial non-zero value, the non-zero ends cause discontinuities in the best match waveform. You must also have sampled fast enough that the change between two consecutive samples is not so large that the matching algorithm fails.

This method will work best in cases where you have a small number of well-defined templates. If you have a large number of indistinct templates, you will likely be able to match just about any shape, but the results will

be meaningless. We suggest that you only use this method where there is an obvious collision and that you treat the results with great caution. Look at the firing patterns of the spike classes and check that the resolved collisions fit the patterns. It can be revealing to exclude a template from a collision to see how close the analysis can get with another pair of spikes.

The matching algorithm

The best match is found by an exhaustive search of all possible pairs of templates at all alignments against the spike data identified by the *Important area*. The searched area may be further restricted by the rule that at least half the points in each template must overlap with spike data points. Templates are combined by adding the mean template shapes and calculating a width. If *Mean width* is checked, the width is the same at all points and is the average width of all templates. If *Mean width* is not checked, non-overlapped areas of the combination copy the original width and overlapped areas use the RMS (root mean square) of the two widths.

For each pair of templates, there are two variables to consider: the offset of the second template with respect to the first, and the offset of the combination into the data. We find the best value of both variables using integral point shifts, then improve the best value by finding the best fractional position of both variables by interpolation.

The match criterion is the mean squared error per compared point. At each point, the error is the difference between the spike and the template combination divided by a width. The width is either the template width at that point, or is the average template width over all templates. The average width includes excluded templates so that errors are comparable when templates are excluded.

Analyse menu commands

The Edit WaveMark dialog Analyse menu contains a range of commands for cluster and Principal Component Analysis, plus commands to manipulate the events that you work with:

Principal Component Analysis

This command (short-cut key `Ctrl+A`) initiates PCA analysis of all the spikes in the current time range using the same waveform area as selected for the templates. This is dealt with in the Clustering section.

Cluster on measurements

This command (short-cut key `Ctrl+Shift+A`) opens the Cluster set up dialog in which you select measurements to take from each spike in the time range for cluster analysis. This is covered in the Clustering section.

Marker filter 

The Marker filter command (short-cut key `Ctrl+F`) opens the marker filter dialog, or brings it to the front if it is already open. Whenever you change channel to a channel holding WaveMark data with the Marker filter dialog open, the Marker filter dialog automatically changes channel to match.

Duplicate 

This command (short-cut key `Ctrl+D`) creates a duplicate data channel in the time view for each template code in the dialog. It is disabled if the current channel is a duplicate channel. It does the following:

1. All duplicates of the current channel are deleted (with no warning) from the time view and from any duplicates of the time view. Any processing that depended on the duplicated channels is cancelled.
2. Spike2 counts how many different template codes you have defined. For example, if you have created four templates with codes 01, 02, 02 and 03, this is counted as 3 different codes.
3. For each template code, Spike2 generates a duplicate channel with the marker filter set to display only spikes that match the template code. The marker filter mask used is the same as the code highlighted in the Marker codes (usually the first mask is used). The channel title of each duplicate is set to `"title-nn"`, where `title` is the original channel title (shortened to 6 characters if too long) and `nn` is the template code in hexadecimal. The new channel is positioned next to the original channel in the time window. It is placed

above the original channel unless the Edit menu preferences have reversed the channel order, in which case it is placed below the original.

The assumption is that you have already, or are about to, classify the spikes in the channel based on the templates in the dialog. If you have duplicated the time view, the new duplicate channels exist in all views, but are only displayed in the view that is linked to the Edit WaveMark dialog. The original channel will remain, unchanged, in all the views.

Set Codes

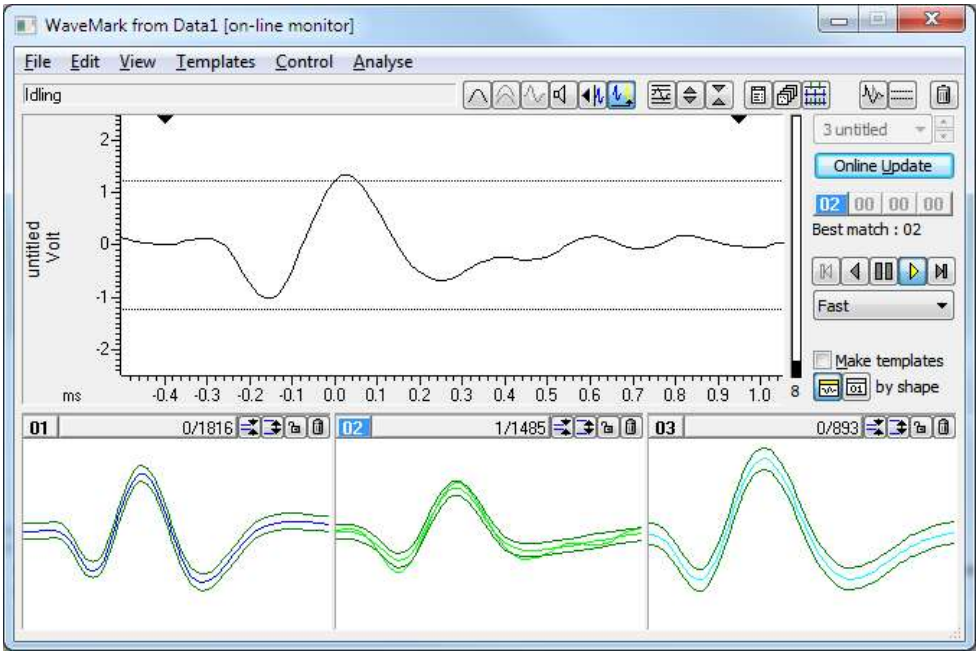
This command (short-cut key `Ctrl+E`) opens the Set Marker Codes dialog. You can use this to give all the displayed spikes in a channel the same marker code. For example, if you want to change all the spikes on channel 2 with codes 04, 05 and 09 to have code 03:

1. Open the Marker Filter dialog and set the filter for channel 2 to display only codes 04, 05 and 09.
2. Click the Set Codes button and set the first code to 03 and click the Set button.

Collision Analysis Mode 

This command (short-cut key `L`) toggles between normal editing mode and collision analysis mode. Normal editing mode attempts to match each spike to a single template. Collision analysis mode attempts to match the spike to a collision of two templates.

On line template monitoring



The Analysis menu Edit WaveMark... command can be used during data sampling if there are any WaveMark channels. The window that opens is very similar to the off-line template editing window; the time range and reclassify buttons are hidden, only the first of the Marker code buttons is enabled, the Reclassify… button is renamed to Online update and there is a new button in the control bar. If there are no spikes to display the view shows the untriggered waveform so you can adjust the horizontal cursors.

 At end mode

When this button is down, all spikes are taken from the end of the file, the horizontal cursors that set the trigger levels appear and controls that cannot be used are hidden. In this state, you monitor the spikes as they are classified by the 1401. If there are no spikes, the data display shows recent waveform data so that you can adjust the trigger levels.

With the button up, the window behaves very similarly to the normal off-line template editing except that the Reclassify command is not available. Once sampling finishes the at end of file button and horizontal cursors vanish, the Reclassify… button appears and the normal off-line editing behaviour is restored.

Online update

During sampling, the CED 1401 interface classifies spikes based on the templates and parameters that were last loaded to it. If you generate more spike templates, change the existing templates, or edit the template parameters this will make no difference to the 1401 unless you click the Online update button. This deletes all templates stored in the 1401 for the current channel and replaces them with the templates displayed in the window. The trigger levels are updated dynamically; you do not need to click the Online update button to change the trigger levels.

On-line and Off-line templates

We save two types of template in the sampling configuration and in the resource files associated with a data file: *on-line* templates and *off-line* templates. Spike2 saves on-line templates when you use the On-line template set up dialog and when you click the Online update button while monitoring spikes during sampling. Off-line templates are saved when you change channel or close the spike shape dialog.

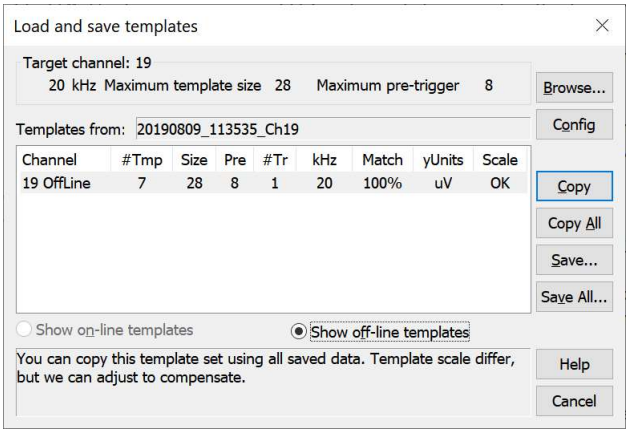
When you change channel, Spike2 loads templates from the saved on-line templates if you are sampling and from the saved off-line templates if you are not sampling. This is so that the templates you see during sampling are as close as possible to the templates that the 1401 uses. Off-line templates are loaded from the resource file associated with the data file. If none are found or there is no resource file, off-line templates are loaded from the sampling configuration.

In summary, on-line templates are the last set of templates copied to the 1401 during sampling. Off-line templates are the last set of templates displayed in a template dialog.

Load and Save templates

You can save templates and load templates into the current channel or into all matching channels with the Spike shape dialog, file menu Load and Save… command. Spike2 saves templates in the sampling configuration and in the resource files associated with data files. Whenever you work on templates in a data file, for example `data1.smrx`, both the associated resource file `data1.s2rx` and the sampling configuration are updated to hold the latest set of templates. You can save and load the sampling configuration in the File menu.

The Spike shape dialog File menu Load and Save... command (keyboard short cut `Ctrl+S`) opens the Load and Save templates dialog:



The box at the top of the dialog shows the sample rate and template size limits of the current target channel. This is the channel that is updated by the Copy command.

The box in the centre of the dialog lists sets of templates from the Sampling Configuration or loaded from a resource or configuration file. When you open the dialog, Spike2 shows you templates in the Sampling Configuration. The columns are:

Channel — Identifies the channel from which the templates originated and if this is an OffLine template (generated during data review) or OnLine online (generated while setting up or sampling).

#Tmp The number of templates stored for this channel. Channels with no templates are not listed. All templates for a channel have the same sampling rate, points and number of pre-trigger points.

Size The number of data points in each template.

Pre The number of data points in each template that lie before the peak or trough used as a trigger. A negative value means that the template starts this many points after the trigger point.

#Tr The number of traces in each template (1, 2 or 4).

kHz The sample rate of the template source channel. This need not match the target channel; Spike2 uses cubic spline interpolation to compensate for sample rate differences. Templates generated before Spike2 version 3.15 did not include a sampling rate and 25 kHz is assumed.

Match Percentage of the time range of each template shape that would be used if this set of templates were copied to the target channel.

yUnits The y axis units of this template. Prior to Spike2 version [10.00], units were not saved, so this field can be blank. If it is blank, Spike2 assumes that the templates have the same units as the target channel. From version [10.01], Spike2 uses the units and template scaling to adjust templates saved from channels with different scales. We expect the y axis units to be one of `V`, `mV`, `uV`, `µV` or `µV` (the last two are the micro symbol and the Greek letter *mu*, which are different, but look very similar). You can replace the `V` with `Vo`, `Vol`, `Volt` or `Volts` and we will still accept it. If we recognise the units, we can convert scaling (e.g. between `mV` and `uV`).

Scale This column indicates how Spike2 will handle scaling differences between the templates listed in this dialog and the temples used by the Target channel. This column will be one of the following:

- No No scaling is possible. This is because the y units of either the target channel or the template were not recognised or because the scaling factor required to convert between the template and the target channel was greater than a factor of 100. If you copy the template the result may not be useful.
- Same No scaling is required as the template has identical scaling to the target channel.
- OK Scales differ and this can be done without clipping the data. The result may lose resolution (but see `Tiny`, below).
- clip Scales differ and this will cause up to 5% of the data points in the template to be clipped.
- CLIP Scales differ and this will cause 5% or more of the data points to be clipped. This may indicate that the template and the target channel are not compatible.
- Tiny The template needs scaling down and the size of the largest result is very small. This may indicate that the template and the target channel are not compatible.

Browse… and Config

Use the Browse… button to list templates in resource (`.s2rx`) or sampling configuration (`.s2cx`) files. The Config button displays templates stored in the sampling configuration.

Show on/off-line templates

Templates copied to the 1401 during sampling (on-line templates) are saved separately from the templates displayed in the template dialogs (off-line templates); you choose which to list with Show on/off-line templates. The Copy All and Save All commands operate on the list selected by this option.

Copy

The Copy button copies the selected set of templates into the target channel and closes the dialog. You can copy a channel as long as Match is at least 10%. This overwrites any existing templates in the target channel. Ideally, when you use Copy, the Scale column of the selected channel will display Same or OK. We do not prevent you using Copy for the other cases, but you should be aware that the results may not be useful.

Copy All

For each channel in the channel control, Spike2 searches the list of templates for a matching channel number. If templates exist for the channel and they match the channel settings, and they have a matching scale or can be scaled with minimal clipping, they are copied from the list and become available for use.

Save
This saves the selected set of templates in the list to a resource file (`.s2r` file extension). You are prompted to supply the name of a new file or to select an existing resource file. If the resource file already exists, any templates for the channel are replaced. If the resource file does not exist, the templates are written to a new file.

Save All
This saves all the templates in the list to a resource file (`.s2r` file extension). You are prompted to supply the name of a new file or to select an existing resource file. If the resource file already exists, any existing templates for the channels in the list are replaced. If the resource file does not exist, the templates are written to a new file.

Cancel
This button closes the dialog.

Template storage

You use templates to classify WaveMark data, but they are not stored in data files. Instead, Spike2 stores templates in the following places:

- In the sampling configuration. You can always load the last set of templates you used on-line or displayed off-line in the current session from here. You must save the configuration to make templates persist after you close Spike2.
- In saved configuration files with the file extension `.s2cx`. For example `Last.s2cx` automatically saves the last configuration used for sampling. However, changes made in the template dialogs after sampling ends are not saved automatically.
- In resource files associated with data files. Each time you use a template dialog, any templates you create or change are saved in a resource file with the same name as the data file, but with the file extension `.s2rx`. During sampling, there is no associated resource file. In this case, when you save the data file, Spike2 creates a resource file and copies the templates in the sampling configuration to it.
- In resource files created from the Load and Save templates dialog.

When you open a new data file for sampling, the template dialogs load up the last set of on-line templates from the sampling configuration. These on-line templates are updated each time you click the Online update button in the Edit WaveMark dialog. The off-line templates are updated to match the templates displayed in the template dialog. When you stop sampling and save the file, the templates are written to both the resource file associated with the data file and to the sampling configuration.

When you work on a data file from disk and open a template dialog, the off-line templates are loaded from the resource file associated with the data file. If none are found, off-line templates are loaded from the sampling configuration. Any changes you make are saved to both the resource file and to the sampling configuration as off-line templates.

By saving templates to both the resource file and to the sampling configuration, Spike2 makes it easy for you to work on a sequence of data files. For example, you might break a sampling session into several data files. On-line templates created for the first data file are automatically used for the second and subsequent files.

Template format and scaling
Templates are saved and manipulated as 16-bit integer waveforms (range -32768 to 32767), which matches how the Spike data is sampled and stored in the WaveMark channel. Templates also contain the channel scale factor to convert from the integer value to user units (typically microvolts or millivolts). From version [10.00], we also store the y axis units of the channel from which the templates were generated with the template.

Prior to Spike2 version [10.01], it was assumed that templates stored in configuration and resource files had the same scaling as the target channel, which was usually the case. However, it was awkward to cope with situations where the same spikes were sampled in two different files with difference channel scales (perhaps due to amplifier settings changing).

From [10.01] onwards, when you Copy a template to the target channel in the Load and Save template dialog, Spike2 will now attempt to scale the template to match the channel scale of the target channel, if the scales differ and if the y axis units of the template and the target channel are in Volts, millivolts or microvolts. Note that this conversion will cause a loss of resolution due to the quantisation of the result of scaling to a 16-bit integer value. If the scaling increases the value, this loss of resolution is not usually significant, but if the values generated exceed the limits of 16-bit integers, the data is limited to the maximum allowed. If the scaling decreases the value, the loss of resolution can be significant and you are warned if the change generates very small templates.

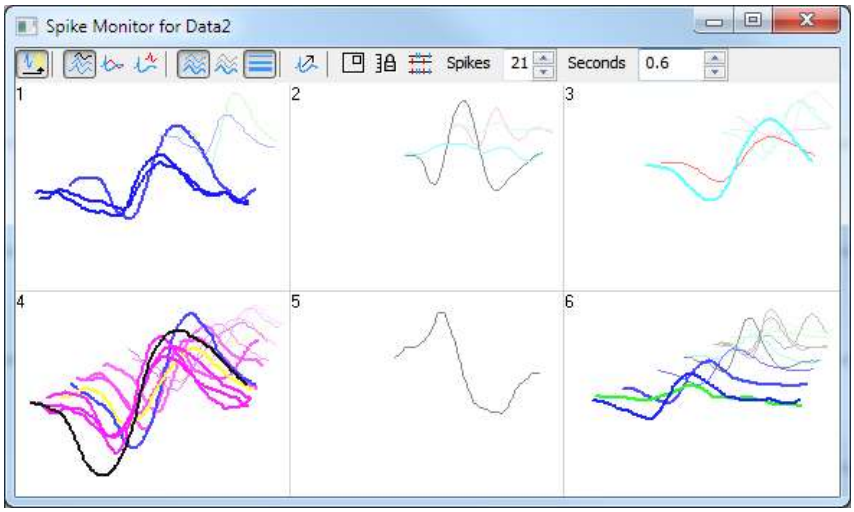
It is likely that spikes recorded using a reasonable fraction of the input range, but allowing some headroom will scale up and down by the typically small adjustments between recording sessions without a problem. You will have problems if you record spikes at very low amplitude, or very close to the limits of the ADC input range.

You can check the range you are using by recording the Spikes source as a waveform channel with the channel Scale set to 6553.6 and the offset to 0. In this case, the y axis units will be Bits (in the range -32768 to 32767). Ideally you would like your big spikes to have a peak-peak amplitude of say 6000 to 20000 bits (this is from 1/10 to 1/3 of the input range). This allows a healthy headroom for scaling up and/or down without losing too much resolution.

If your spike peak-peak amplitudes are down in the 100s of bits, you probably should consider increasing the gain of your input amplifiers. This will both improve the quality of your recordings and also give you more adjustment range for template scales.

Spike Monitor

The Spike Monitor window can be opened with the View menu Spike Monitor command or from the script language with `SMOpen()`. The main use of the window is to give an overview of the current spike activity on all channels during sampling. You can drag and resize the window to position it; the grid of cells organises itself automatically. Each cell displays the channel number at the top left and spikes that lie in a user-defined time range back from the current time. The *WaveMark background* field of the *Application colours* sets the cell background colour.



The window is linked to the Edit WaveMark dialog. Double-click a cell to open the Edit WaveMark dialog. Single-click a cell to select the corresponding channel in the Edit WaveMark dialog. The display is controlled by the toolbar at the top of the window. The toolbar commands are:

At end 

This button is enabled for on-line use and during rerun off-line. With the button down, spike data comes from the end of the file. With the button up, the data comes from the cursor 0 position in the associated time view. If you start to rerun a data file, this window automatically switches to *At end* mode, and cancels this mode when the rerun ends.

Display mode 

There are three display modes: *3D*, *2D* and *2D separate*. Spikes are positioned in the cell using two rectangles: a front rectangle that is anchored to the bottom of the cell and a back rectangle that is anchored to the top of the cell. You can display the rectangles and change their positions with the Rectangle command, described below. Normally the front rectangle is larger than the back rectangle. The display modes are:

3D
Each spike is drawn in a rectangle that is positioned between the front and back rectangles depending on how old the spike is relative to the current time. New spikes appear in the front rectangle and then move to the back rectangle as time passes.

2D
All spikes are drawn in the front rectangle; the back rectangle is not used.

2D separate
The last spike is drawn in the front rectangle, the rest are drawn in the back rectangle.

Colour fade 

With this button down, the spikes colours fade into the background colour with time. A new spike is drawn in its normal colour. A spike that is half the user-defined time range old would draw in a colour that is half way between the normal colour and the background colour.

Colour front only 

With this button down, only the newest spike is drawn in its normal colour. All other spikes appear in a grey that contrasts with the background enough to be visible. This is particularly useful in the 2D drawing mode.

Vary line thickness 

Normally, all spikes are drawn with the same line thickness as data in a time view as set in the Edit menu Preferences. With this button down, the lines become thinner with time, with new spikes being drawn with a thick line. The line thickness for a new spike depends on the height of the cell. Use this option with caution; a thick line takes longer to draw than a thin one.

Timed (smooth) updates 

In *At end* mode, cells update when new spikes enter or leave the time range, which can lead to a jerky display. With this button down, cells update continuously, which can look much better, particularly in the 3D display mode. However, this option uses more system time especially if *Vary line thickness* is enabled. The continuous update rate is decreased when the Spike Monitor window is not the active window to make more time available to other windows.

Show rectangles 

Normally, the front and back rectangles that are used to position spikes are hidden. With this button down, the rectangles are drawn and can be moved by clicking and dragging with the mouse. The spikes are still displayed, but are drawn in grey so that the rectangles are clearly visible. To move a rectangle sideways, click in it and drag it. To size a rectangle, click in it to display two drag handles and then click on a drag handle to resize.

Lock y axes 

With this button up, the spikes in each cell self-scale. Each cell tracks the maximum amplitude it has seen. When a new spike that exceeds the current maximum amplitude is seen, it updates the maximum amplitude. This new maximum is held for two seconds, then it decays with time. With this button down, the current y axis scale is held.

Duplicate channels 
With this button up, duplicate channels are not displayed and any marker filters set for channels are ignored. With this button down, duplicate channels are displayed and only the spikes that meet the marker filter settings for each channel are displayed.

Spikes
You can set the maximum number of spikes to display in each cell, in the range 1 to 40. The time taken to display the data depends on the number of spikes, especially if you enable *Vary line thickness*. You should set this to the minimum number that is useful.

Seconds
This field sets the time back from the current time range over which to display spikes. It is normal to set this to a small number of seconds, but you may need to set a very short period if you are trying to visualise burst characteristics in the 3D display mode.

Getting started with spike shapes and templates

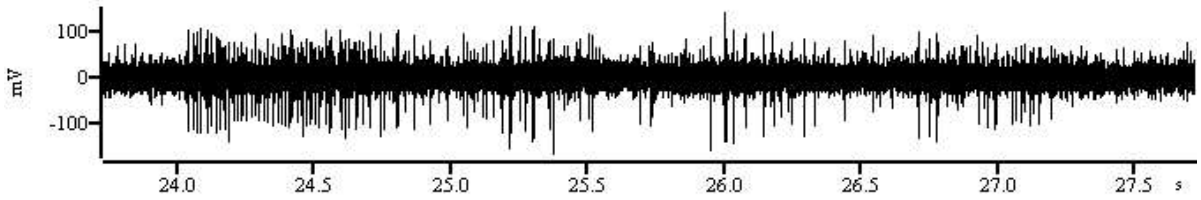
Sorting spikes is as much an art as a science; there is no substitute for the skill and experience of the investigator. Spike2 provides you with a toolbox of routines that will help you to discriminate various shapes, but there is no guarantee that two spikes are from the same source just because they have the same shape (or because any other measured parameter or set of parameters are the same).

Quality of the original signal

Probably the most important contributor to successful spike sorting is the quality of the original data. If you can improve the signal to noise ratio, or adjust the electrode position to increase the amplitude of your target spikes relative to the background, the time spent will be rewarded by more certain classifications in less time.

The waveform sample rate used is also very important. For most spikes lasting 1 to 2 ms, a sample rate of 20 to 25 kHz is about right (some workers like higher rates than this). It is important that the data is frequency band limited to half the sample rate before it is sampled. For example, when sampling at 25 kHz there must be no frequency components above 12.5 kHz. If such components are present, they are aliased to lower frequencies and appear as noise that is very difficult to remove.

To check the quality of your signal it is a good idea to sample a section of data as a waveform first. Here is an example waveform sampled at 25 kHz from a tape recording.

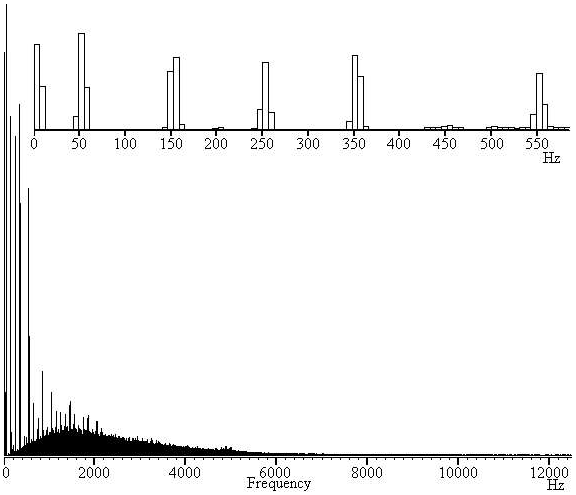


Sampling configuration for this data

Open the sampling configuration Channels tab and click Reset. Click New Channel and select a Waveform channel. Set Channel to 1 and 1401 Port to 0. Set the Ideal adc sampling rate to `25000`, and set Units `mV` = Input in Volts x `1000` + `0` and click OK. Now click the Resolution tab and set Optimise to Full. Finally click the Mode tab and select Continuous. You can now sample.

The example waveform amplitude is around ±100 mV, which is a bit small. The 1401 family usually expect a full-scale input range of ±5 Volts, and our spikes are using only 2% of the available range. The size of each digitisation step is about 0.15 mV for the Power1401 or Micro1401. Although the system will work fine with signals of this amplitude, if your rig has more gain available, it is much better to make the signals around ±1000 mV in amplitude. This gives you headroom for larger spikes and a good signal resolution.

Check the input for unexpected frequencies with the Analysis menu: New Result View: Power Spectrum command with a block size of 4096. Here we found narrow peaks at low frequencies and almost nothing above 6 kHz. Apart from a peak at 0 Hz caused by a small DC offset, the remaining peaks are all at odd multiples of 50 Hz. This data was recorded in England where the mains frequency is 50 Hz. Anything that reduces mains interference (better shielding, removal of earth loops, cleaning up of connections) will make spike sorting easier.



Sharp power spectrum peaks often indicate signal contamination. If the peaks change frequency when you change the sample rate, they are caused by aliasing of frequencies above half the sampling rate and they may be removable with an external low-pass filter.

Setting up the WaveMark channel

Assuming that we have adjusted the rig to get the best signal to noise ratio and spike amplitude, we must now set up a suitable sampling configuration to capture our spikes. We could sample the data as a waveform, but unless you really need all the data in between your spikes, this approach is wasteful of disk space. For example, a single waveform channel sampled at 25 kHz consumes 50 kB of disk space per second, or 3 MB per minute, or 180 MB per hour.

An alternative is to sample WaveMark data. In this case, each time the input signal crosses a positive or negative threshold we search for a peak or trough in the data and save the time and a small fragment of waveform. If we save 32 data points per spike, each spike costs us 80 bytes of data. As long as you set the threshold levels so that you get less than 625 spikes per second, this method uses less storage space. At a mean spike rate of 20 spikes per second, you would consume 96 kB per minute, or 5.8 MB per hour.

There is one further advantage to WaveMark data. As the data is captured, Spike2 can match it to templates and code each spike accordingly. This gives you further on-line analysis capability as you can look at the responses of individual spike types on-line. The on-line sorting is not the final word as you can revue and resort the data off-line.

Suggested configuration for WaveMark data

Open the sampling configuration Channels tab and click the Reset button. Then click New Channel and select a WaveMark channel. Channel should be `1` and 1401 Port should be `0`. Set the Maximum sustained event rate to `30`, and set Units `mV` = Input in Volts x `1000` + `0`. Set Points to `32` and Pre-trigger to `10` and WaveMark sample rate to `25000` and click OK. Now click the Resolution tab and set Optimise to Full. Finally click the Mode table and select Continuous. You can now sample.


Spike2 knows that you have a WaveMark channel in the sampling configuration, so when you click the Sample now! Button in the toolbar, it opens up a window for you to set trigger levels and adjust the number of sample points and to build templates.

When you open a window for the first time you are likely to have a more or less flat line across the centre of the display area and the trigger levels will not be in useful positions.

There are a lot of buttons and controls in this window. If you move the mouse pointer over one and leave it for a second or so, a line of text will “pop-up” with a short explanation.

Home key or 
This button is one of the most useful when you are setting up for spike shape capture. Each time you click the button, Spike2 will make a guess at reasonable trigger levels (and limit levels for the New WaveMark command with limits enabled) and adjust the display gain so that you can see the data. Once a trigger level is crossed a second trigger cannot occur until the waveform falls below half the trigger level.

It is likely that you will only wish to trigger in one direction, so move the cursor that you do not want to use to the edge of the window. To change a level, move the mouse pointer over the level, hold down the mouse button and drag to the new position.

It is possible to use both trigger levels, but we strongly suggest that you do not to avoid the sideways shift you will get if a spike triggers on the opposite side from the one you intended. If your spikes are biphasic, that is they have a rising peak and a falling peak, it is best to trigger in the direction that has the clearer peak. If your spikes are triphasic, as in the example above, it is best to trigger in the direction that has only one peak to avoid problems caused by false triggers on the second peak. If both directions are equally clear, choose the direction with the narrower peak (usually the first one).

Up and Down keys or 

Once you have got more or less the desired display gain you can use these two buttons to change the display scaling manually. You may also want to reduce the display gain so you can drag one of the trigger levels off the screen to reduce visual clutter. As an alternative to using this button, move the mouse pointer over the y axis and click and drag to scale the data.

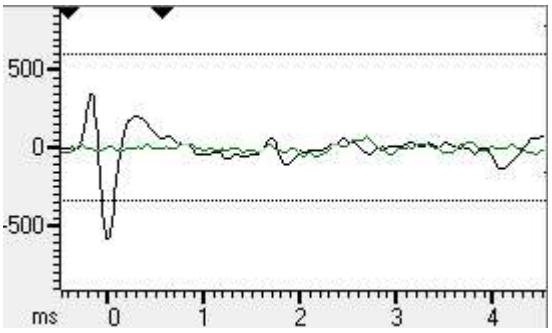
Left and Right keys or 

These buttons change the number of pre-trigger points. The trigger point is the first signal peak or trough after it crosses the trigger level, and is labelled 0 on the time axis. You can also click and drag the tick marks of the x axis to adjust the pre-trigger. You cannot change the number of pre-trigger points after you start to sample.

PgUp and PgDn keys 

These buttons increase and decrease the data points that are displayed and written to disk for each spike. The number of pre-trigger points is preserved if this is possible, so you should set the pre-trigger points first. You can scale the x axis around the zero point by clicking and dragging the x axis numbers.

You need to capture sufficient points to enable accurate spike identification. You also want to capture as few points as possible to minimise the data storage. In this example we have captured about 5 ms of data, which is at least 3 ms more than we need. As well as leading to huge data files, sampling too many points also increases the chance of the data for one spike containing data for a second or even a third spike. Further, Spike2 will not trigger for the next spike until it has finished sampling the previous one. You cannot change the number of points once you start sampling.



 The two black triangles mark the start and end of the region of the spike that is passed to the template matching system. Click on a triangle and drag to change the region. When you drag the markers, vertical cursors drop down to identify the selected area.

 The play control buttons from left to right are: step back, run back, pause, run forwards and step forwards. Step and run backwards are disabled during set up. There are keyboard short cuts for these buttons: `b` or `B` steps backwards, `N` runs backwards, `v` or `V` pauses, `n` runs forwards and `m` or `M` steps forward.

Forming templates

By now you should be able to set up the system to capture spike shapes. The next step is to form templates from these spikes. The first step is to check that the template parameters are set to sensible values.

Ctrl+Enter or 

Click this button to open the template parameters. To get started, set these values:

Suggested template parameters

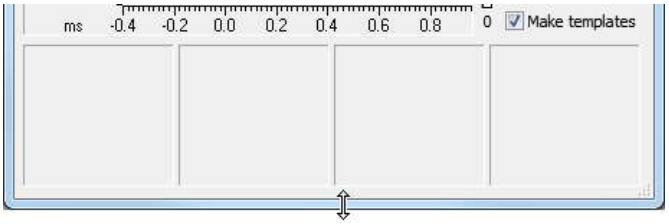
In the New template section, set Number of similar spikes for a new template to 8, New template width as a percentage of amplitude to 30, No template for spikes rarer than 1 in to 50. In the Matching a spike to a template section, set Maximum percent amplitude change for a match to 0, Minimum percentage of points in template to 60 and check the box for Use minimum percent only when building templates. In the Template maintenance section, set Template modification mode to Add All. In the Waveform data section, set the Waveform interpolation method to Linear, High-pass filter time constant to 20 ms and leave the Remove DC offset before template match unchecked. Finally click OK.

As you get more familiar with template matching you will probably change these values. However, this set is a useful starting position. We have turned off the amplitude scaling as this causes a large increase in on-line computation and so should not be used unless you really need it. Amplitude scaling comes into its own when your spikes change amplitude while preserving shape, for example during a burst.

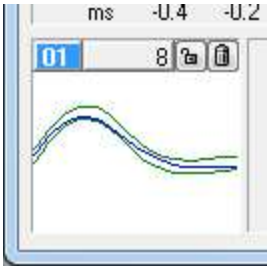
Del key or 

Spike templates are displayed below the data display area. If you cannot see the template area drag the bottom edge of the window downwards with the mouse. The template area may already contain templates. If it does, you can clear them all by clicking the bin button at the top right of the window. You can remove individual templates by clicking the small bin button in the template window.

Expanding the window to show the template area



The template windows have three sizes: small (in the picture), medium and large. You change from small to large by double clicking in the template area. Spike2 remembers the last window size you used for small, medium and large template displays. You can also expand the window sideways by dragging the right-hand edge of the window, or you can grab one of the corners to expand in both directions. As a short-cut, you can maximise and minimise the size of the window by double-clicking in the title bar.



To build templates, make sure that the Make templates box is checked, the play control is set to run forwards and that spikes are crossing the trigger level you set in the data display area. If all is well, when 8 similar spikes have been detected (8 is the number you set in the template parameters) a new template should appear in the template area.

In small mode, the four rectangles at the top of the template hold, from left to right, the code associated with the template (01), the number of spikes in the template (8), a button that indicates the locked state of the template and a button that can be used to delete the template. The vertical size of the displayed template is set by the vertical scale of the data display area.

As spikes are processed, the vertical bar on the right of the data display area indicates the number of templates that are being considered in grey and the number of confirmed templates in black. The first 20 confirmed templates appear in the template area.

The horizontal size of the template is set by the two black triangles in the data display area. For good spike sorting, it is important to select the region of the spike that shows the most variation between different classes of spikes. Do not include baseline areas before or after the spike. Spikes are matched to templates based on the errors between the mean template and each spike. Spikes are excluded from templates based on individual points falling outside the template boundaries.

You can control the appearance of the templates with these three latching buttons. If you click them once they become active, click them again and they become inactive. The leftmost button controls overdrawing of spikes in the template. With the button down, all spikes that match the template are over-drawn in the template. With the button up, only the last matching spike is drawn.

The centre button chooses between a display of the template boundary with the button down, and a display of the mean template with the button up. With the right-hand button up, spikes are displayed only in the template that they match. With the button down, spikes are also displayed (in grey) in all non-matching templates.

Spike code and template colour

Each template has a code in the range 1 to 255. When the templates are used for spike sorting, the code is used to label matching spikes. Code 0 is not used by templates, but spikes can be given a zero code, meaning that they are not coded.

The code is represented by two hexadecimal digits; 1 is represented by 01, 2 by 02 and so on up to 9 which is 09. In hexadecimal (base 16 numbers), the numbers ten to fifteen are represented by the letters A to F, so ten decimal is 0A, eleven is 0B through to fifteen which is 0F. Sixteen decimal is 10 in hexadecimal, seventeen is 11 and so on.

We also allow the use of single character codes in place of the hexadecimal codes 20 to 7E. The keyboard marker channel uses the same coding system and the ASCII codes for the Roman printing characters lie in the range hexadecimal 20 to 7E. Although Spike2 can manage up to a hundred templates internally, it displays only the first twenty, so codes above 14 (decimal 20) are rarely used and single character codes are rarely a problem.

The code given to a template also determines the template drawing colour. In the View menu Change Colours…dialog you can assign colours for template codes 01 to 08 (or to a much higher code, if you choose), see the dialog for details. It is a good idea to set colours that contrast strongly with the background colour you have set for the time view as this colour also sets the background colour for the template windows.

New templates take the lowest available code, but you can change the template code by double clicking on it. This opens a dialog box in which you can edit the code. There is no restriction on the code you give, other than it must be in the range 01 to FF. In particular, you can have more than one template with the same code.

There are useful commands in the Templates menu. Renumber changes template codes to remove gaps caused by deleting or editing codes. Order by Code sorts the templates into ascending code order.

Template operations

Automatic template creation

With the play control set to run and the Make templates box checked, Spike2 creates templates automatically by looking for similar spikes and following the rules set in the template parameters dialog. Each new spike is compared first with the existing confirmed templates (displayed in the template area). If it doesn't match any of them it is compared with the provisional templates. If it matches nothing, it forms a new provisional template.

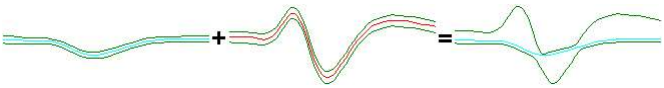
When a spike matches a template it modifies the mean template shape and width unless the template is locked. If the spike matches more than one template it is added to the one with the smallest error between the spike and the mean template shape.

When a provisional template gets a new spike and this causes the template spike count to reach the Number of similar spikes for a new template set in the template parameters, it is ready to become a confirmed template. If it matches a confirmed templates it is merged with the confirmed template. Otherwise, a new confirmed template is created.

To avoid the system being swamped with provisional templates, there is a mechanism that makes them decay away. This is set by the No template for spikes rarer than 1 in field of the template parameters dialog. If this number of spikes are tested and a provisional template does not get one new spike added to it, the template spike count is reduced by one. If the spike count becomes 0, the provisional template is deleted.

Manual template creation and merging

Instead of letting the system create templates, you can build them yourself. Use the play control to step forwards one spike at a time until you find one that you want to turn into a template. Click the mouse in the middle of the display area and drag the spike to an unused window in the template area and release. If you drag a spike shape to an existing template, the spike is added to it.



You can combine two templates by dragging one of the templates and dropping it on the other. Spike2 will align the dragged template with the target and then merge the two shapes by combining the template limit.

This example shows the effect of merging two very dissimilar templates. You can see that the extremes of the template boundaries of the original templates are preserved in the result. Normally, you would merge two similar templates. You can test for template similarity by dragging a template over another without releasing.

Number of templates

You can create (and Spike2 will remember) up to 20 templates per channel. However, if you are preparing for on-line sampling, only the first 8 templates in the template area will be passed to the 1401 for use on-line. You can, of course, resort the data off-line using all 20 templates, and it is possible to sort spikes into many more classes than this by using the Analysis menu Marker Filter and Edit WaveMark commands.

Locking templates

Each time a spike matches a template, and the Make templates box is checked, the spike is added into the template unless the template is locked. The lock button in each template can be used to lock and unlock templates manually. You can also make the templates lock automatically after a set number of spikes have been accumulated by setting the Template modification mode in the template parameters dialog to AutoFix.

If you don't lock the templates, they change as more spikes are added. This means that the shape you end up with could be quite different from that with which you started. For example, if you create a template manually from a specific spike, you may want it to keep the exact shape so you would lock it.

Amplitude variation

If you find that the system keeps generating multiple templates for the same spike because the spike amplitude changes, there are several things to try. The obvious solution is to set the Maximum percent amplitude change for a match to a non-zero value. For example, if you set this to 30%, each spike can be increased or decreased in size by up to 30% before attempting to match it to the templates. Spike2 does this by computing the area between the spike and the zero baseline and changing the spike amplitude so that this area is the same as the area between the template and the baseline.

Beware that this is computationally expensive, which does not matter for off-line sorting, but it can be important on-line, especially if you don't have a Power1401. The more channels of spikes and the more templates you are matching against, the larger the time penalties you will impose. If you find that turning this option on leads to spikes not being classified when the spike rate is high, then you may need to consider other options.

Another method to cope with amplitude variation is to make the templates wider. You can do this by merging templates, or by increasing the New template width as a percentage of amplitude in the template parameters. However, wider templates leads to less discrimination between spike shapes.

Closely allied to making the templates wider is to reduce the Minimum percentage of points in template field. However, this also leads to less discrimination between spike shapes.

You might also consider checking the Use minimum percent only when building templates box. If you do this, when you run on-line and when reclassifying spikes, spikes are matched to the template that has the smallest error between the shape of the spike and the shape of the template, the template limits are ignored. This option is most useful when you know that a whole bunch of spikes are either type A or B but the data is very noisy, so you chose a good spike of each class to make two templates, then you check the box and sort on the basis of minimum error.

If you do not need to separate many shapes on-line, you can accept that it takes 2 or even 3 templates to cover the full range of amplitudes and give the 2 or 3 templates the same code (double click the code and edit it).

Using sound

If your computer has a sound card, you can play a sound for each spike. Spikes that match a template play a sound with a pitch that rises as the code for the template increases. Spikes that don't match will play a lower pitch. How successful this is depends on the capabilities of your sound system! You may find that listening to the raw signal wired to the input of an audio amplifier is a better solution.

Sampling data

Although we have not covered every possible control you could have used, you should now have enough information to set up your system with some templates and sample data. Click the Sample Data button and the dialog box will close and a new data file window opens. Closing the template set up dialog, fixes the number of points that will be captured for each spike and the number of pre-trigger points. If you followed our sampling configuration settings, the new data window will contain a channel of WaveMark data and the Keyboard marker channel.

Monitoring spikes

You can adjust the trigger levels and even change the templates during sampling with the Analysis menu Edit WaveMark command. If you use this a window that is very similar to the template set up window opens.

The display area shows the last spike that crossed a trigger level, and when there are no spikes, it shows the latest data. You can adjust the trigger levels by dragging them, and *the level change is passed to the 1401 immediately*. If you make any other change, this has no effect on the 1401 unless you click Online update.

The 1401 holds a set of templates that it uses to classify spikes. There is a separate set held by Spike2 that start out as identical to the ones in the 1401; you can change them by adding spikes, deleting templates or creating new ones. Click Online update to copy the latest Spike2 templates to the 1401.

The leftmost of these 4 buttons shows the marker code assigned by the 1401; the other three buttons are not used on-line and are disabled. The Best match field displays the marker code of the template in the template area that best matches the spike in the display area. These will usually be the same. They can differ if you have edited the templates, or if Make templates is checked so that the template area is changing.


If you enable sound, the tones you hear depend on how the spikes match templates in the template area; the tones do not depend on the classification done by the 1401. This is to allow you to modify templates in preparation for Online update.


This is a latching button that is normally down for on-line use. Click the button to change the state between down and up. With this button down, the displayed spikes are always taken from the end of the data file, that is from the spikes that have just been sampled. If the spike rate is too high to show all the spikes, spikes are skipped.

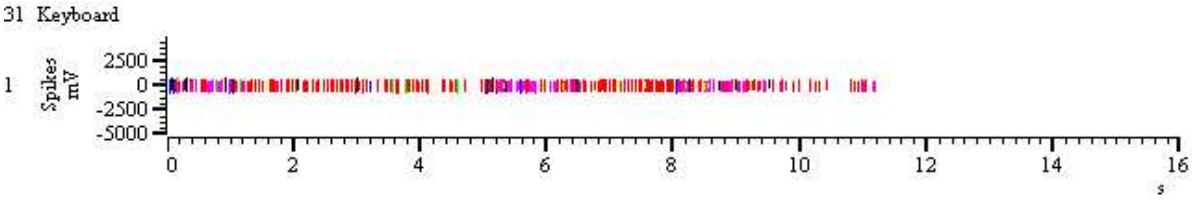
With the button up, a new cursor appears in the time view to mark the point from which spikes are taken, all the play control buttons become enabled and the windows behaves almost identically to the off-line Edit WaveMark window which is described later. The marker code buttons still only allow the first marker code to be used; you can use all four codes when off-line.

Monitoring multiple channels

If you have multiple channels of WaveMark data, you can monitor them all by opening the Spike Monitor window (use the View menu Spike Monitor command). This view allows you to check all the WaveMark channels at a glance. This window also has links to the Edit WaveMark dialog. If you click on a channel in the Spike Monitor window, this selects that channel in the Edit WaveMark dialog, ready for adjustment.

Drawing modes for spikes

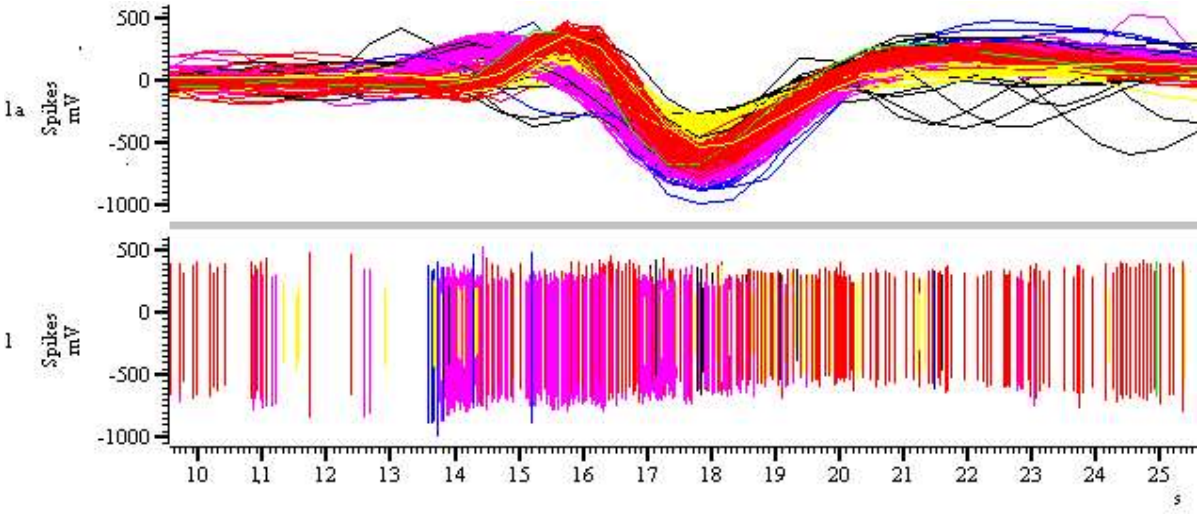
If you use the View menu Channel Draw Mode… command and select the WaveMark channel, you will find that you have a very wide choice of display modes. To start with, display the channel as Waveform. Arrange the x axis to show 10 to 20 seconds worth of data and click the Start button in the sampling control bar or use the Start Sampling command in the Sample menu. You should see something like the following:



If no spikes are appear, use the Analysis menu Marker Filter… command and select each of the four layer in turn and click All, then click OK. We will discuss this command in more detail later.

The spikes are colour coded for the template that they match, or are drawn in black if they don't match any template. These spikes are rather small, so the first thing to do is to double click the y axis and optimise the display. You can display the codes for the spikes as well as the waveforms by changing the drawing mode to WaveMark, but this is not very useful when you have a lot of spikes in the window as the codes will overwrite each other. Drawing in WaveMark mode on-line is much slower than drawing as Waveform, so we do not recommend this mode for on-line use.

Another useful mode is Overdraw WM (overdraw WaveMark). This gives an immediate overview of the spike sorting. To see this mode in action, use the Analysis menu Duplicate Channels command to duplicate the WaveMark channel. Then use the View menu Channel Draw Mode… command to set the drawing mode for the duplicated channel to Overdraw WM. You may need to adjust the size of the data window to generate a useful image. In the next example we have hidden the keyboard channel.

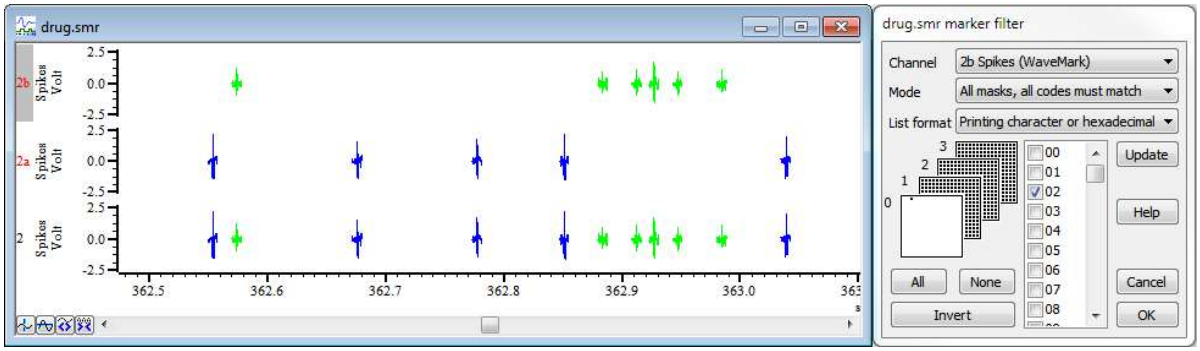


In Overdraw WM mode, the full width of the window is used for each spike. There is a grey bar between the overdraw area and the rest of the window to show that the x axis does not apply. When the window scrolls, new spikes are added to the overdraw area, but old spikes are not removed as this would force the entire window to redraw, which could take a long time. If you want to force the overdraw area to redraw, click or drag the thumb in the scroll bar at the bottom of the window.

You can locate a particular spike in Overdraw WM mode. Move the mouse pointer so that it is over the desired waveform at a place that is clear of all other waveforms, right-click and select Find with cursor 0 from the pop-up context menu. This moves cursor 0 to the position of the event. If the Edit WaveMark dialog is open, it will also move to the position as cursor 0 controls where the dialog collects data.

Using the Marker Filter

Although it is nice to see all the spikes you have collected, for most purposes, you are interested in the behaviour of one unit, or the relationship between one unit and another. If your templating and spike sorting has gone well, you will have isolated each unit as spikes with a particular code (or possibly more than one code). You now need a way to treat each unit as a separate channel of data.



The hard way

In the example shown, channel 2 has two classes of spike, sorted into codes 1 and 2. You could generate channels 2a and 2b by duplicating channel 2 twice (right click on the channel and select Duplicate from the context menu). Then right click on channel 2a and select the Marker Filter command, clear the check box next to 01 and click the Invert button. This clears all check boxes except the one next to 01. Then click Update and channel 2a will display only spikes with code 01. Next, select channel 2b in the channel list and repeated the procedure for code 02.

The easy way 

However, there is a much easier way to do this. The Edit WaveMark dialog has a button that generates a duplicate channel with the marker filter set to display each template code. Alternatively, you can use the Edit WaveMark dialog Analysis menu Duplicate command (short-cut `Ctrl+D`).

The result of this is that we have two new channels, each with spikes from a single unit. If your sorting resulted in one unit having more that one code, then you should select all the codes for that unit in the Marker Filter dialog; there is no need to sort spikes so that one template matches every single spike from the unit.

If you decide that spikes with several different codes should all have the same code, you can use the new Set Marker Codes command in the Analysis menu, or right click on a channel and select this command from the context menu. Set the marker filter for the channel to display the codes that you wish to amalgamate then use the Set Marker Codes dialog to change all the spikes with these codes to the same code.

Creating on line templates with cluster analysis

Cluster analysis is normally used off-line. This is because extracting principal components and automatic clustering can take a noticeable time if you are working with a large number of events; also the clustering analyses are not available from the on-line template set up dialog. However, you can use clustering to create on-line templates.

Off-line clustering

Sample as normal and set the spike trigger levels in the on-line template set up dialog. There is no need to create templates at this stage. Start sampling and capture enough data to be able to form templates, and then stop sampling. Open the Edit WaveMark dialog and from the Analyse menu use the clustering method of your choice to classify the captured data.

When you are satisfied with the clusters, use the cluster dialog File menu Apply command to classify the original data. This will also clear all templates for the channel from the Edit WaveMark dialog and build new templates based on your classification. Now close the Edit WaveMark dialog. This will save the templates in the sampling configuration as off-line templates.

Now sample again, and when the on-line template set up dialog opens, use the File menu Load and Save command (short-cut Ctrl+S) and select the off-line template for your channel. This will load the templates that you created off-line.

On-line clustering

We do allow you to use the Edit WaveMark dialog on-line to monitor spikes and you can also use the clustering commands on-line. However, the clustering functions will run more slowly than when working off-line. If you have large data files and huge numbers of events or a high data throughput to disk, this may not be a viable procedure as the system may not be responsive enough to be useful.

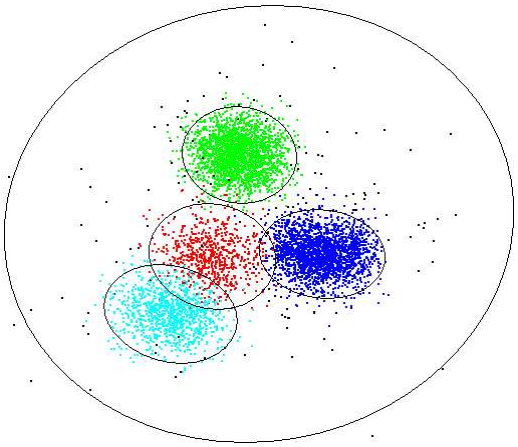
If you are using timed or triggered sampling, data sections that are not marked for writing to disk are volatile; data that the clustering code expects to find may have vanished by the time you try to apply changes.

When you apply the results of clustering on-line, the newly created templates are copied to the 1401 as if you had used the Online update button.

17: Clustering

Clustering

Introduction



Clustering is the generic term for methods that group similar objects into classes. In our case, the objects are spikes that consist of 1, 2 or 4 waveform traces of typically 20 to 40 data points each. The number of objects to cluster will often be large; clustering tens of thousands of spikes is not at all uncommon, so the algorithms used must be fast enough to manage large data sets in a reasonable time. Spikes also form a temporal sequence and we have to take into account that the clustering properties may change with time.

If you have `n` data values that define each object, you can cluster in `n` dimensions. However, if `n` is greater than 3 it is very difficult to visualise the clusters. For spikes, `n` is typically 30 or more, so we need ways to extract 2 or 3 independent measurements from our data that maximise the differences we care about between the classes and minimise the differences due to noise in the data. Spike2 provides four methods:

Principal component analysis	Principal Component Analysis (PCA) is a mathematical procedure that automatically extracts the features from your data that contribute the most to the differences between the waveforms that make up your spikes. This is usually the method to try first as it is less susceptible to noise than feature measurements.
Feature measurements	Feature measurements are user-defined values extracted from each spike, such as amplitudes, latencies, areas and slopes. Each trace is parametrized based on positions of peaks and zero-crossings. You then choose two or three measurements that emphasise the differences between the spike classes to cluster the data.
Template correlation	You select two or three spike templates, then the x, y and optionally z parameters of the clustering are the waveform correlations of each spike with the selected templates. The correlations are amplitude independent, so this can be useful with spikes that maintain shape but change amplitude.
Template errors	You select two or three spike templates, then the x, y and optionally z parameters of the clustering are the sums of the squares of the errors between each spike and the three selected templates.

Extracting clustering values

All methods generate a table of values with one row per spike. Each table row holds the spike time, a class code and the x, y and z values derived from the spikes. All clustering operations operate on the data in this table, not on the original spikes in the data file; you can choose to apply classification changes to the original data at any time. We call the items in the cluster window *events* to distinguish them from the original spike data.

The clustering dialog contains tools that make it easy to visualise the clusters in two and three dimensions and tools that allow you to classify the clusters either manually or with various degrees of automation. How well automatic clustering works will depend on how well separated the clusters are.

Using cluster analyses

The starting point for cluster analysis is the Edit WaveMark dialog. The clustering commands are in the dialog Analysis menu. The analyses operate on data in the time range set in the Edit WaveMark dialog and honour marker filters set for the data channel. Changing the Edit WaveMark channel closes any associated clustering windows.

The clustering dialog is linked to the Edit WaveMark dialog. The current event in this dialog flashes in the cluster window. If you click on an event in the cluster window, the Edit WaveMark dialog will jump to the spike that generated the event; when you apply the results of the cluster analysis, the Edit WaveMark dialog will re-evaluate its templates to match the new classifications.

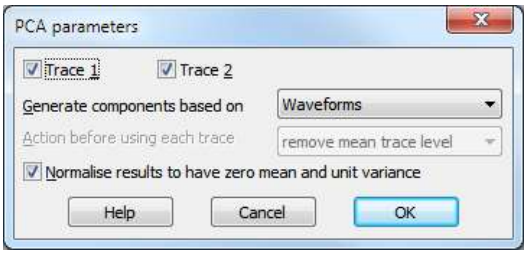
Reasonable limits

Although we try very hard to make drawing the clustering images as fast as possible, once you have more than a few hundred thousand spikes it takes an appreciable time to recalculate the display, so the animation of the clustering becomes jerky. Further, the PCA analysis takes a time proportional to $m*n^2$ where m is the number of spikes and n is the number of data points that represent the spike. You may find that you want to limit the value of $m*n^2$ to around 1 billion to avoid very long waits for the PCA analysis to complete.

Principal Component Analysis

To use Principal Component analysis, open the Edit WaveMark dialog (make sure you are not in collision analysis mode) and drag the black triangles at the top of the data display area to select the region of each spike to process. Ideally you should exclude baseline regions from the spikes to reduce noise and improve the cluster separation. The time range in the data file to process is set by the Set Time Range command in the dialog Control menu.

The Principal Components command in the Edit WaveMark dialog Analysis menu opens the PCA parameters dialog. If your spikes have a single trace, everything except the Normalise measurements check box is disabled. With more than 1 trace (stereotrode and tetrode data), all dialog items can be used. The items are:



Trace n

When you have more than one trace you can choose which to use for analysis. If you clear all the check boxes, Trace 1 is selected automatically. Some of the other options in the dialog require at least 2 selected traces; reducing the number of selected traces may change other dialog fields.

Generate components based on

This field allows you to choose the data that the principal components are generated from when you have more than one data trace. You can choose from:

Waveforms	The data source is the waveform region of each spike set in the Edit WaveMark dialog.
Amplitude at time 0	The data source is the peak amplitudes of the selected traces.
Mean amplitude and ratios	The data source is the mean peak amplitudes (ignoring sign) of the selected traces and the ratios of the peak amplitudes to the mean peak amplitude. The peak amplitude is found by searching each trace for the peak (or trough) that is nearest to the trigger point. The trigger point is at 0.0 milliseconds in the Edit WaveMark data display window. Traces are pre-processed as set by the Action before using each trace field before locating the peak.

If there is not enough data to generate three output values, a z value is not generated and you will not be able to rotate the data around the x or y axis. This will happen if you choose a mode other than Waveforms and have 2 selected traces. It can also happen in Waveforms mode if the selected spike region contains less than three data points.

Action before using each trace

This field allows you to decide what pre-processing should be applied to each trace before it is used. In Waveforms mode the mean level of the selected region of each trace is always subtracted. In other modes you can choose from:

No trace pre-processing	This trace is used exactly as read from the data file.
Remove mean trace level	The mean level of the trace is subtracted before using the data. You can use this to remove baseline drift.

Subtract best-fit line — A straight line is fitted to the data of each trace and subtracted from the trace before the data is used. If your spikes suffer from sloping baselines you can use this method to remove them. This also removes DC offsets from your data.

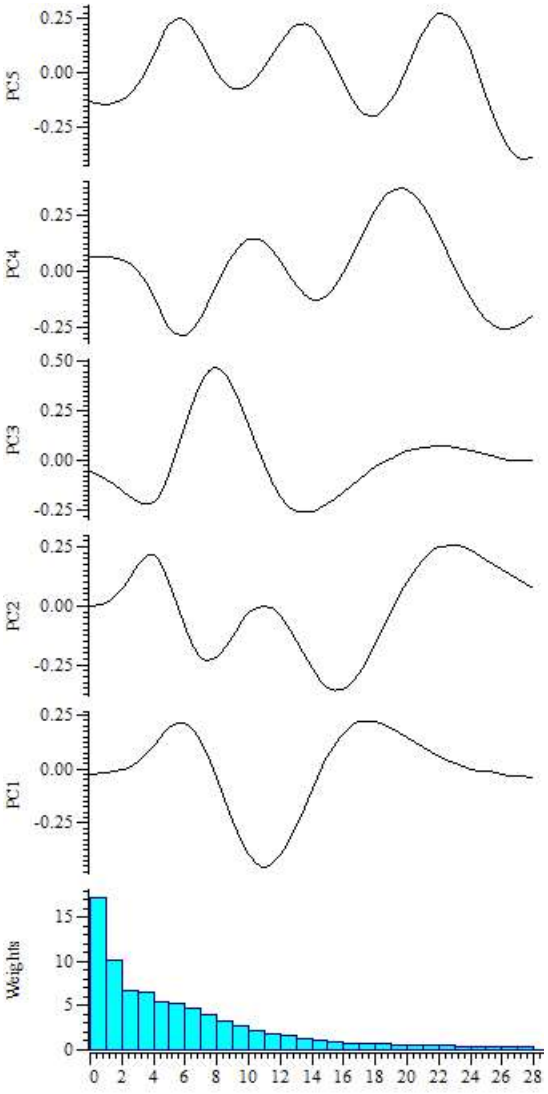
Normalise measurements

If you check this box, the x, y and z components generated by the principal component analysis are shifted and scaled so that they all have a mean of 0 and a variance of 1. This can make the data easier to manipulate in the clustering dialog. If you do not check this box, the displayed axes in the cluster plot originate at (0, 0, 0); events close to this point are small spikes, and are more likely to be due to noise.

Performing the analysis

Click the OK button to analyse the data. This can take a long time if you have a many tens of thousands of spikes and many data points per spike. If the analysis takes longer than one second, a progress dialog appears with a Cancel button that allows you to abandon the analysis and return to the Edit WaveMark dialog. If a marker filter is set for the channel, only spikes that match the filter are processed.

What is Principal Component Analysis



Principal component analysis reduces a large set of non-independent data into a set of orthogonal components that is ordered in terms of the significance of each component to the whole dataset. Orthogonal means that if you multiply and sum corresponding elements of two components, the result is zero. In terms of the Spike2 script language, `ArrDot(Ci, Cj)` where `Ci` and `Cj` are two components is 0 for `i<>j` and 1 for `i=j`. For most sets of spikes, only the first few components are useful, the rest correspond to noise. For clustering, the x, y and z values are the proportions of three user-selected principal components in each spike (usually the first three).

There is no need to be familiar with the mathematics behind principal components to make effective use of it. However, some knowledge of how it works may help you to understand its limitations and let you decide it is appropriate for your data. Principal Component Analysis is based on Singular Value Decomposition (SVD), which can be summed up by the matrix equation:

X = **U** . **W** . **V'**

Where **X** and **U** are matrices with m rows by n columns, **W** is an n by n diagonal matrix and **V'** is the transpose of an n by n square orthogonal matrix. When applied to spike waveforms, m is the number of spikes, n is the number of data points in each spike. The rows of the matrix **X** are the spike waveforms with their mean value removed. The matrix **V** holds the principal components, the diagonal matrix **W** holds the variance of the original data that each component accounts for and each row of **U** holds the contribution of each component to the spike in the corresponding row in **X**.

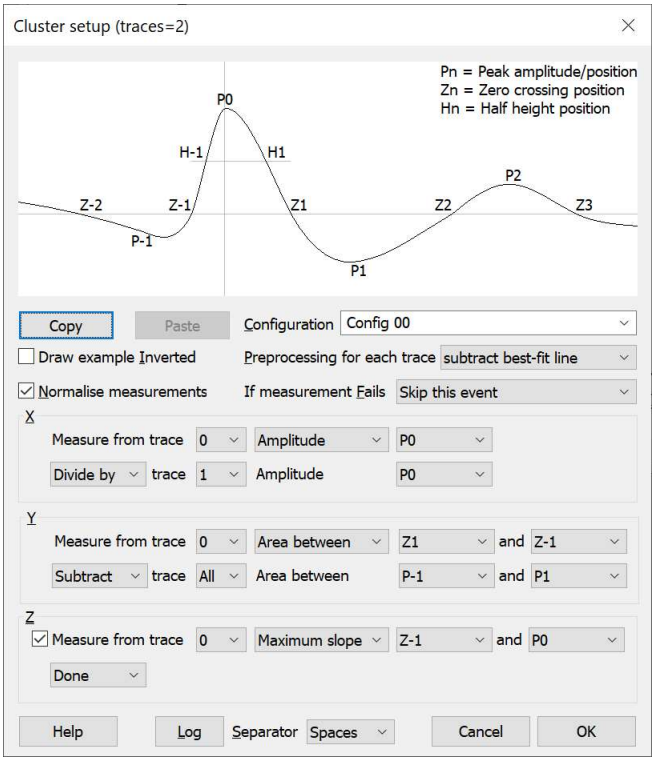
The principal components are ordered so that the first principal component contributes the most variance in the original data, the second represents the most remaining variance after removing the first component, and so on. If there are n points in the original spike waveform, you will get up to n principal components.

The time taken to compute **U**, **W** and **V** from **X** is proportional to mn^2; this means that doubling the number of spikes doubles the time, doubling the number of data points per spike quadruples the time. This is why you should not include baseline data for each spike; not only does it add noise to the analysis, it also takes longer.

If you want to see what the components look like, the File menu Principal Components option of the clustering dialog creates a result window holding the principal components and how much each of them contributes to the original data set.

Cluster on measurements

When you select Cluster on Measurements from the Edit WaveMark dialog, a new dialog opens in which you can choose either 2 or 3 measurements to take from the spike data for clustering. The entire spike trace is used for measurements. The time range in the data file to process is set by the Set Time Range command in the Edit WaveMark dialog Control menu.



The dialog has 4 main regions: a schematic of a spike trace to use as a reference when selecting measurements, general controls, definitions of the x, y and z measurements, and the buttons to close the dialog. The dialog controls are:

Configuration

There are 10 measurement configurations, selected with the Configuration combo box. Configurations for 1, 2 and 4 traces are stored separately. Initially, the configurations are labelled as Config 00 through Config 09, but you can edit these names to describe the type of measurement. You can copy a configuration to a different position in the table. Click the Copy button on the configuration you want to copy, select a different configuration and click Paste to overwrite it. The Copy button also copies a human-readable text version of the measurement configuration to the clipboard using the format set by the Separator field.

Draw example inverted

The example waveform at the top of the diagram can be drawn either way up. This control lets you choose the direction that best suits your data.

Preprocessing for each trace

If your data has a large DC offset or has a sloping baseline, measurements may not generate the expected results. You can choose to process each trace in your data before the measurements are made. You can choose from:

No trace pre-processing
This trace is used exactly as read from the data file.

Remove mean trace level
The mean level of the trace is subtracted before using it.

Subtract best-fit line
A straight line is fitted to each trace and subtracted from it before the data is used.

If measurement fails

It is not always possible to make a measurement. For example, a peak might not exist in the data. You can choose between:

Use standard work-around
This sets unmeasurable amplitudes to 0 and unmeasurable times to a time just past the end of the spike when searching forwards or to a time just before the start of the spike when searching backwards.

Skip this event
The spike is not included in the measurements or clustering. If you use the Apply command in the cluster dialog, all skipped spikes are given code 00.

Normalise measurements

If you check this box, the measurements for x, y and z are shifted and scaled so that they all have a mean of 0.0 and a variance of 1. This can make the data much easier to work with in the cluster dialog, especially if you rotate the data. Of course, the measurements are then in arbitrary units, so you may not wish to do this if you want to export the values as text from the cluster dialog.

X, Y and Z measurements

These three areas are identical except that you can enable and disable Z measurements with the Measure from trace check box.

We model each trace by searching outwards from time 0 for the peak P0. We expect this peak to exist because the spike data was aligned on a positive or negative peak when it was captured. Next we search outwards in both directions from P0 to find the zero crossings Z-2, Z-1, Z1, Z2 and Z3. Then we find the peaks P-1, P1 and P2. Finally we locate the half peak height positions H-1 and H1. To make the searches as accurate as possible, we fit a cubic spline through the trace data points and search for peaks and level crossings using the fitted data. In addition to these measurements, the following were added at version 7.02:

PkMax	The position and amplitude of the highest (most positive) peak.
TrMin	The position and amplitude of the minimum (most negative) trough.
Max	The position and amplitude of the most positive point in the trace.
Min	The position and amplitude of the most negative point in the trace.

The top line of each area sets a basic measurement. The first field on the second line can be set to one of Done to use this value or Subtract or Divide by to make a second measurement and subtract this from the first or divide the first measurement by the second. You can choose from the following measurements:

Amplitude	You can select any of the peaks (P-1 to P2) as the measurement. The units of measurement are the y axis units of the channel.
Time at	The measurement is the time of any feature (Peak or level crossing). The units are milliseconds relative to the peak trigger point (the 0 time in the Edit WaveMark dialog).
Area between	The measurement is the area between any two features, from the curve to the y axis zero. The units of the area are y axis units times milliseconds. All areas are treated as positive.

Maximum slope	The measurement is the value of the steepest slope of the fitted cubic spline between any two features in y axis units per millisecond.
Best fit slope	The measurement is the slope of the least-squares best-fit line between any two selected features in y axis units per millisecond.

If you are working with multiple trace data you can choose which trace to use as a source of your measurement. If any part of the measurement calculation uses a failed feature measurement, the measurement is marked as failed. You can also choose All traces, in which case the measurement is the average measurement value on all of the traces.

Log and Separator

The Log button copies a human-readable version of the measurement configuration to the Log view (see Copy to copy to the clipboard). The Separator field allows you some control over the format:

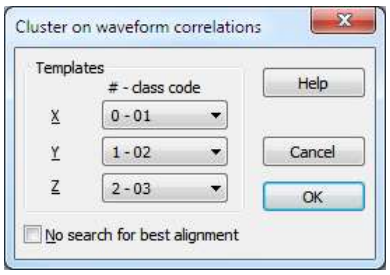
Spaces	Separate fields with spaces, suitable for printing with a monospaced font or to paste into a notebook.
Commas	Separate fields with commas, which can read into some spreadsheets.
Tabs	Separate fields with Tab characters, for instance to read into the Grid view.

Running the analysis

Click on the OK button to save any changed configuration settings and generate the measurements. The analysis process takes a time proportional to the number of spikes. Once the analysis is done the Cluster dialog will appear. The Cancel and OK buttons change to Close and Apply if you open this dialog with the Reanalyse command from the clustering window.

Cluster on template correlations

To use this option, click on the Cluster on Correlations command in the Analysis menu of the Edit WaveMark dialog. There must be at least 2 templates defined in the dialog for this command to be enabled. The command opens a dialog in which you can choose two or three templates to correlate each spike with to generate the X, Y and optionally Z values for clustering. You can choose None as an option for the Z value.



No search for best alignment

If you check this box, the section of the spike shape correlated against each template will be the section marked for template formation in the main display in the Edit WaveMark dialog. If you leave the box unchecked, Spike2 will search around the section marked for template formation for the best correlation. This search extends for up to two sample points in either direction and uses cubic spline interpolation of the spike waveform to estimate values between sample points.

You will normally leave the box unchecked unless you have chosen a template region that does not include the spike alignment point (usually the peak or trough), and the time delay of similar shapes is the distinguishing criterion. If you check the box, the analysis is much quicker, but the result is usually significantly worse.

How the correlation is calculated

The correlation between a segment of the spike shape starting at position `s` and a template of `n` points is calculated as follows:

1. Make a spike segment of `n` points (use cubic spline interpolation if `s` is not integral).
2. Normalise the spike segment by shifting and scaling so that the mean value is zero and the sum of squares is unity.
3. Normalise the template to have zero mean and unity sum of squares.
4. The correlation is the sum of the point by point product of the two waveforms.

The result of this is a value between 1.0 (if the spike and template have the same shape) and -1 (if they have the same shape but one is inverted with respect to the other). If the two shapes are significantly different, the result will be nearer to 0 than to 1 or -1.

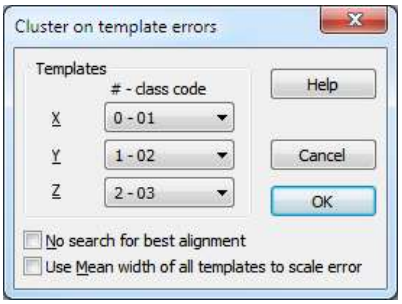
When to use correlations

The correlation has the property that it is *independent of the amplitude of the spike*; the correlation is purely a function of shape. As spike amplitude is often the single most important feature when sorting, you should consider carefully whether this is worth losing. On the other hand, if you have a spike of more or less constant shape, but variable amplitude, this may be exactly what you want.

We anticipate that this method will be useful in sorting similar spikes with varying amplitudes into two or three categories, possibly after using other sorting methods to reduce the problem down to the difficult cases.

The results produced in three dimensions tend to be a thin layer of events with patches at a radius of 1 from the origin for each of the templates, which may not be ideal for the automatic clustering. You may find that you have to use manual event selection.

Cluster on template errors



To use this option, click on the Cluster on Errors command in the Analysis menu of the Edit WaveMark dialog. There must be at least 2 templates defined in the dialog for this command to be enabled. The command opens a dialog in which you can choose two or three templates to compare each spike with to generate the X, Y and optionally Z values for clustering. You can choose None as an option for the Z value.

No search for best alignment

If you check this box, the section of the spike shape compared with each template will be the section marked for template formation in the main display in the Edit WaveMark dialog. If you leave the box unchecked, Spike2 will search around the section marked for template formation for the smallest error. This search extends for up to two sample points in either direction and uses cubic spline interpolation of the spike waveform to estimate values between sample points.

You will normally leave the box unchecked unless you have chosen a template region that does not include the spike alignment point (usually the peak or trough), and the time delay of similar shapes is the distinguishing criterion. If you check the box, the analysis is much quicker, but the result is usually significantly worse.

Use Mean width of all templates to scale error

With this unchecked, the error at each comparison point is scaled by the template width at that point. If you check the box, the error is scaled by the mean width of all templates (with multiple traces, the error for a trace is scaled by the mean template error for that trace in all templates). If you check this box, the error can be thought of as the least squares error. If you don't check it, the error is more like a chi-squared error based on the template width. Use whichever gives the better separation.

How the error is calculated

The error between a segment of the spike shape starting at position `s` and a template of `n` points is calculated as follows:

1. Make a spike segment of `n` points (use cubic spline interpolation if `s` is not integral).
2. Adjust the spike segment to have a mean value of zero.
3. Adjust the template to have zero mean.
4. Form the sum of the squares of the point-by-point difference of the two waveforms divided by either the point-by-point width or by the mean width.
5. Divide the result by the number of points to form the average error per point.

The result of this is a value from 0 upwards. You would expect values for a matching template to be less than 1.

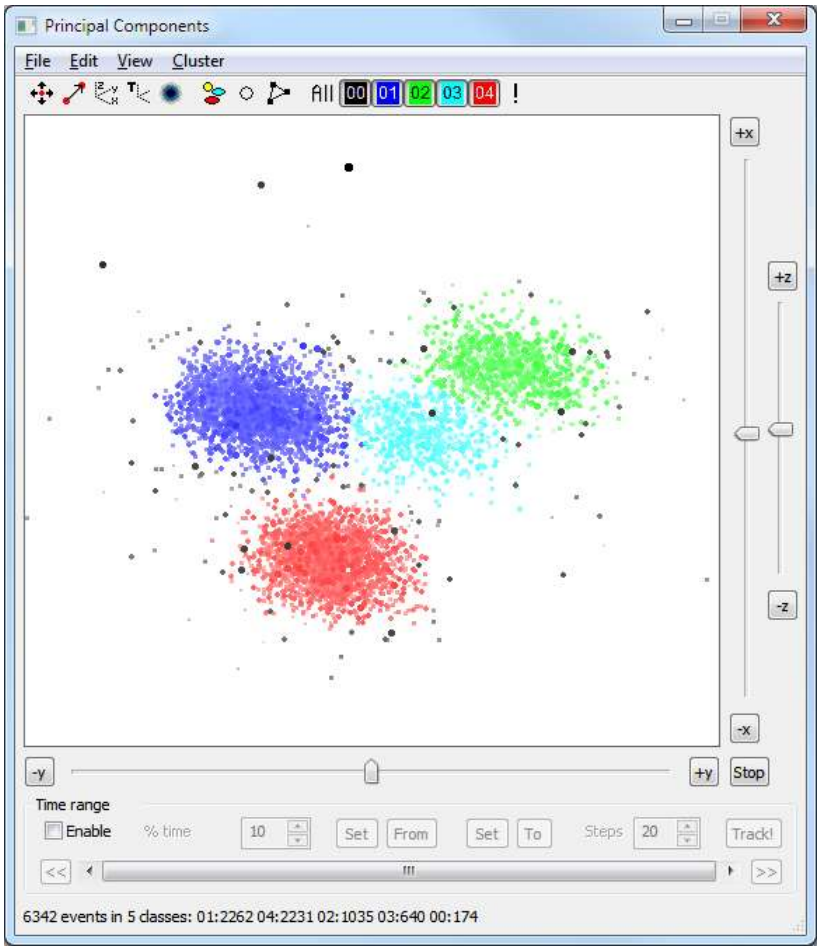
When to use errors

We anticipate that this method will be useful in sorting similar spikes into two or three categories, possibly after using other sorting methods to reduce the problem down to the difficult cases. Spikes that resemble each of the templates should lie in the range 0 to 1 on the axis that corresponds to the template.

If you check the *Mean width* box, this method is similar to the method used for template matching in the Edit WaveMark dialog and you can use it to get a visual indication of the reliability of the separation achieved by the template method. If this produces non-overlapped clusters, it is likely that the template method is reliable.

The Clustering dialog

You reach this dialog by using one of the clustering commands in the Analysis menu of the Edit WaveMark dialog. The dialog behaves almost identically for all analysis methods; the dialog title indicates the source of the data. The dialog contains a menu, a toolbar that selects what to display, the cluster window, sliders for rotation around the x, y and z axes, the Time range area and a status line at the bottom. You can hide the rotation slider controls and the Time range area.



Clustering dialog (click features for more information)

You can resize the dialog by clicking on any edge or corner. The minimum size depends on the displayed items; if you hide the Time range area and the rotation controls you can make the dialog very small indeed!

The analysis you selected built a table of events. Each event contributes one dot to the display. For each event, the table holds the associated spike time, the x, y and z values from the analysis and the class code. The initial class codes in the table are copied from the spikes. Each spike has 4 marker codes; we use the marker code selected in the Edit WaveMark dialog. Class assignments made in this dialog only change the table. The File menu Apply changes command copies changes to the data file. There are two methods you can use to display the events, Density Plot and Class code colours.

Density plot
Select this display mode with the View menu Density Plot command. The display shows the density of events in each pixel. The View menu Density Colour Map command lets you change the colour scale used to indicate the event density. The View menu Density Settings command gives you control over the creation of the density map.

Class code colours
In this display mode, the colour of each dot is set by the class code (matching the WaveMark colours used in the time window) and the background colour for the cluster region is set in the main Spike2 colour palette.

Current event
The event corresponding with the spike displayed in the Edit WaveMark dialog blinks on and off if it is in the visible area. This event blinks, even if it is not part of the currently selected set of event; this is for information only. The event blinks between the class colour of the event and the background colour. You can change the current event by clicking on a different event; this will change the current event here and in the Edit WaveMark dialog, and will scroll the associated time view if it is tracking the Edit WaveMark dialog. If you have Minimum Intervals enabled, clicking near an arrow indicating a short interval will set the event that starts the interval as the current event.

Class ellipsoids
We calculate statistics for each class of events and based on the assumption that the event x, y and z values in each cluster are normally distributed about the cluster centre, we calculate ellipsoids of constant probability around the cluster centres. You can set the size of the ellipsoids in terms of the Mahalanobis distance (this is multi-dimensional version of standard deviation) with the View menu Ellipse radius command. We display the projection of the ellipsoid onto the cluster window.

User shapes
We also maintain a two-dimensional user ellipse and a user-defined shape that are used to surround events and give them a user-defined class. The user ellipse is independent of the three dimensional class ellipsoids.

Status line
The status line shows the results of operations, but more usually it reports on the state of the clustering. If no class ellipses, or user ellipse or user-defined shape is selected, the report is on the number of visible event classes, for example:

6341 events in 5 classes: 01:2263 03:2231 02:1041 04:634 00:172

In this case, 6341 events are visible and they are in 5 classes. The classes are listed from the most populated to the least populated in the format class:count. The report stops after 9 classes. If Short events intervals are being displayed, the report is of the form class:count(short) where short is either the count, or the percentage of events in each class that were at intervals less than the minimum:

6341 events in 5 classes (1 short): 01:2263(1) 03:2231(0) 02:1041(0) 04:634(0) 00:172(0)
6341 events in 5 classes (0.016% short): 01:2263(0.044%) 03:2231(0%) 02:1041(0%) 04:634(0%) 00:172(0%)

You can select count or percentage in the Minimum interval dialog.

If you select one or more class ellipses or the user ellipse or the user-defined shape, then the summary changes:

Encloses 2300 events in 3 classes (1 short): 01:2241, 00:51, 03:8

Now all the selected events that are selected are grouped together, regardless of class and the total number/percentage of short intervals is displayed.

Toolbar controls

There is a Toolbar across the top of the display. Each button has an associated tool tip that is displayed if the mouse pointer lingers for more than half a second over a button. The tool tip contains a reminder of the function of the button. The buttons are arranged in three groups:The group on the right controls the display of events in the cluster window.

Code buttons 

There is one button for each class code that is present in data. Each button displays an event colour and class code. The colour scheme is the same as for WaveMark data in a time view. Events are displayed for buttons that are down, events are hidden when buttons are up. Only visible events are modified by the commands in this dialog. The All button is a short-cut to display all the event codes. Right-click All to hide all codes. The ! button inverts the displayed class codes; all visible event classes are hidden, all hidden event classes are made visible.

State buttons 

The group on the left provide short-cuts for various menu commands. You will find the details of these commands in the View menu description, below. From left to right:

Autoscale	Scale the image to keep everything visible. You can zoom in and out using the mouse wheel; `Ctrl`+mouse wheel zooms faster.
Link close events	Link events closer that a minimum time period with an arrow. Right click this button to open the Minimum Interval dialog.
Show axes	Display x, y and z axes in the display area.
Use Time as Z	Use event times as the z axis for display purposes
Density plot	Display a density plot. Right-click to open the settings dialog.

Ellipse buttons 

The central group controls the display of shapes that either describe the extent of a class of events or that are used to mark events as belonging to a class From left to right:

Class ellipses	Display ellipses for each visible class of events. Right-click the button to open a dialog that controls the size of the ellipse.
User ellipse	Display a user-defined ellipse. This is used to select events for coding and to restrict the events used in the interval histogram.
User shape	Display a user-defined shape formed from straight lines. This is used in the same way as the User ellipse. To create a User shape, `Alt`+click in the display area, then click for each new node and double-click to close the shape.

Time range control

You can hide and show the Time range control area with the View menu Show Time Range command or with the `Ctrl+T` keyboard short-cut. This area restricts the number of displayed events by setting a time range that they must lie within. The % time range field sets the time range as a percentage of the time range spanned by all the events that were analysed. The thumb of the scroll bar is set to a size to match the time range.

When you check the Enable check box, the events displayed in the cluster window and used to build cluster statistics (and hence the class ellipses) are restricted in time. You can drag the thumb of the scroll bar to change the time range. If you click the >> or << buttons the scroll bar steps through the data in the indicated direction, a second click stops the stepping. You can use this to see if the clusters change position with time.

When you change the time range, everything that depends on the displayed events changes to track the displayed time range. If you have an active interval histogram and you have selected automatic updates in the INTH settings dialog, the histogram will update to track the current time range.

If the number of displayed items of any class falls below the minimum required to calculate the class ellipses (4 in three dimensions, 3 in 2 dimensions), the class ellipse will not move until sufficient items are in the selected time range.

Tracking clusters

If you have clusters that move with time you can use the Track! button to assign class codes automatically over a time range. Proceed as follows:

1. Move the thumb to the start of the time range to track and make your initial cluster assignments, then click the Set next to the From button to mark the start position.
2. Move the thumb to the end of the time range (this can be before or after the From position time) and click the Set next to the To button to mark the end position.
3. Set the Steps field to the number of intermediate steps to use in progressing from the From position to the To position.
4. Click Track! to class events based on the clusters at the From position. The K Means dialog will open and you can choose the method to use for normalising the data. Once this is done, click OK to start tracking.

The From and To buttons jump the scroll bar to the start and end of the time range. The tracking algorithm works as follows:

1. The clusters at the From position are assumed to be correct. The statistics of these clusters are calculated. All the events past the start position to the end of the time range are marked as code FF – which stands for unknown.
2. The time range is moved on by: (To time – From time) / Steps and we run the K Means algorithm on the new data and recalculate the statistics. The classes of events more than 3 times the Mahalanobis distance from the cluster centres are not changed.
3. Repeat step 2 until we reach the To time. Any events that are still marked as unknown are set to code 00.

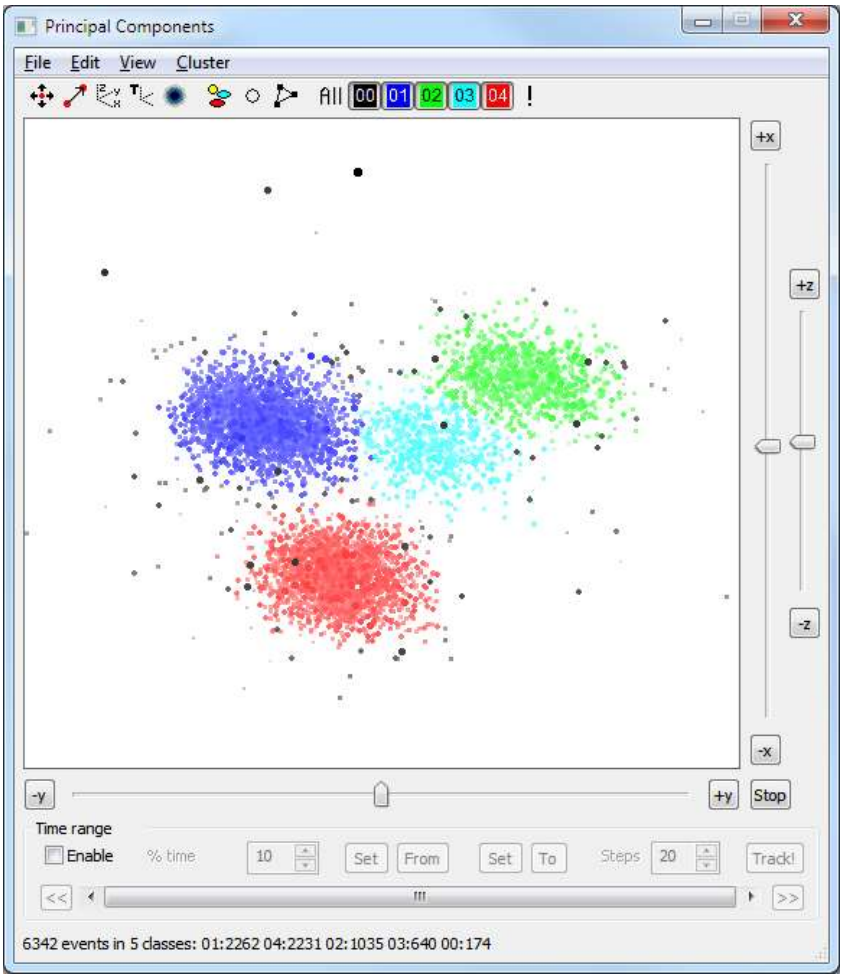
The larger the number of steps, the better the algorithm will cope with rapid cluster movement, but the longer the process will take. Because the algorithm is based on K Means, all the problems that K Means suffers from also apply. If you have periods where a class of events vanishes, or if new events appear in the middle of a time range, you may need to break down the tracking into multiple time ranges.

If there are clusters that do not move, you can greatly simplify tracking by coding these first (you may need to do this with the Set Codes command) and then hiding them.

Identifying events

The mouse pointer changes to a cross hair when it is over the cluster window. To find out which event produces a particular dot in the window, centre the cross hair on the dot and click the mouse button and release. Spike2 will locate the nearest event to the centre of the cross hair and display it in the Edit WaveMark dialog. Cursor 0 will also move to the spike position in the associated time view. This does not work if you are on-line and the *always take spikes from the end of the file* button is down in the Edit WaveMark dialog.

Scaling and shifting the cluster window



Clustering dialog (click features for more information)

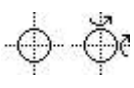
You can zoom in to an area in the cluster window by clicking and dragging with the mouse. While the mouse is down, the mouse pointer changes to a magnifying glass with a plus sign in the centre and a dotted rectangle shows the selected area. When you release the mouse, this area expands to fill the window. If you hold down the `Ctrl` key before you click and drag, the magnifying glass has a minus sign in the centre and when you release the mouse, the original area of the window is scaled to fit in the dotted rectangle.

You can also zoom in and out using the mouse wheel (if present on your mouse). Use `Ctrl`+mouse wheel to zoom faster.

You can drag the cluster window by holding down both the `Shift` and `Ctrl` keys before you click and drag the display. The mouse pointer is an open hand before you click and a closed hand when you drag.

Rotating the cluster window

There are three sliders with associated buttons that rotate the cluster window around the x, y and z axes. Initially, x runs from left to right, y runs from bottom to top and z is out of the screen towards you. The small buttons at the ends of the sliders (labelled +x, -x, +y, -y, +z and –z) set a continuous rotation. Each time you click one it changes the rotation rate. The Stop button cancels continuous rotations. If the sliders are not visible you can use the `Ctrl+S` short-cut key to show them.



You can also rotate the cluster window using the mouse. If you hold down the `Shift` key and move the mouse over the cluster window, the mouse pointer changes to a cross-hair with a circle. Now imagine that the events are in a large glass sphere and that you are looking at it through a square window. If you click and drag, with the `Shift` key down, this "grabs" the surface of the sphere and moves it in the direction that you are dragging.

When you rotate the cluster window, any displayed class ellipses also rotate. If the user ellipse is present it does not change.

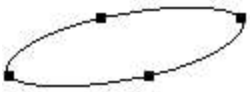
No z information

In some cases, the data resulting from the analysis will not have any z information. In this case, rotation with the x and y sliders and by clicking and dragging is disabled. You are still allowed to use the z slider to rotate the data around the z axis.

Jitter the cluster window

Instead of rotating the view, you can apply a small circular jitter to the display with the Jitter command in the View menu. This allows you to rotate the display to best view the clusters, then apply a small jitter to help you visualise the clusters in three dimensions.

Working with ellipses



You can select, drag and modify the shape of displayed ellipses. Click on or within an ellipse to select it; four resizing handles appear to show that it is selected. To select multiple ellipses, hold down the `Ctrl` key before you click. If you click in a region that is enclosed by more than one ellipse, the ellipse with the smallest area is selected.

To move a selected ellipse, click inside the ellipse and drag. To resize an ellipse, click on one of the handles and drag. The opposite handle is fixed while you drag. If you hold down `Ctrl` and drag, the ellipse resizes around its centre. If the display window scaling is different in the x and y directions you will find that the ellipses distort if you rotate them by dragging a handle. You can force the scales to be the same in both directions with the View menu Equal scales command.

When you select one or more ellipses, the status bar displays a list of the classes and the number of events in each class that lie within the selected areas. The class list is sorted with the largest class first. If you have short intervals enabled, the count of short intervals in all the selected events is displayed.

Ellipse types

There are two types of ellipse: the class ellipses that are centred on each class and the user-defined ellipse that you can activate from the toolbar.

Although you can change the positions and sizes of the class ellipses, such changes have no effect on the classes unless you use one of the methods discussed below to change the class codes. To restore the positions of the class ellipses, click the toolbar SD button twice: once to hide ellipses, and a second time to restore them to their original positions. Alternatively, force the display to redraw by zooming or shifting it.

With x, y and z axes, class ellipses can only be calculated if the class contains at least 4 events and these events are not co-planar. If you are working with only x and y axes, you need at least 3 events and they must not lie on a line.

Changing class code with ellipses

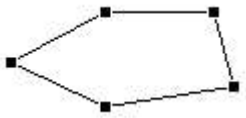
You can change the class codes of the events inside or outside selected ellipses with the Cluster menu Set Codes command. As a short-cut, if you press the keys `0` through `9` with one or more ellipses selected, all events within the selected ellipses change class to match the key: `0` for class 00 up to `9` for class 09. The `F` key sets code FF as a way to temporarily position undecided items. You can 'paint' codes by dragging an ellipse (or a user-defined shape) while holding down one of the keys.

You can also set the codes of all visible events with the `Ctrl+Shift+0` through `Ctrl+Shift+9` key combinations and `Ctrl+Shift+F` to set code FF. Beware that Microsoft used `Ctrl+Shift+0` for IME language control in Windows 7. You can get instructions to defeat this here.

Scaling the user-defined ellipse

If you select the user-defined ellipse and hold down the Shift key, rolling the mouse wheel will scale the size of the ellipse around its current position. This also works for the user-defined shape.

Working with the user defined shape



If your clusters are not ellipsoidal, you may find it easier to select items with the user-defined shape. To start a shape, hold down `Alt` and click in the cluster window. Then move the mouse and click to add additional corners. To close the shape, click at or near the start point, or double click to add a point and then close the shape.

The user-defined shape behaves in similarly to the user ellipse. You can select and move it around the cluster window. You can select individual handles and drag them to adjust the positions of the corners.

Changing class code with the user-defined shape

If you press the keys `0` through `9` with the user-defined shape selected, all events within the shape change class to match the key: `0` for class 00 up to `9` for class 09. The `F` key sets code FF as a way to temporarily position undecided items. You can 'paint' codes by dragging the user-defined shape (or an ellipse) while holding down one of the keys.

You can also set the codes of all visible events with the `Ctrl+Shift+0` through `Ctrl+Shift+9` key combinations and `Ctrl+Shift+F` to set code FF. Beware that Microsoft used `Ctrl+Shift+0` for IME language control in Windows 7. You can get instructions to defeat this here.

Scaling the user-defined shape

If you select the user-defined shape and hold down the `Shift` key, rolling the mouse wheel will scale the size of the shape around its current position. This also works for the user-defined ellipse.

Online Clustering

When you open the clustering dialog while sampling data, you can choose to have new events added into the display as they occur (online update) or to run exactly as if you were analysing a file offline. You can enable and disable online update mode, set how far back in time to search for events, and limit the number of events to display with the File menu Online Update dialog.

If the Edit WaveMark dialog is in At End mode when the cluster dialog opens, the clustering starts in online update mode, and processes the most recent range of data as specified in the Online Update dialog. Otherwise, the range of data set for the Edit WaveMark dialog is processed. This initial analysis sets the parameters for online analysis. Changes in the Edit WaveMark dialog have no effect on online updates; you must use the Reanalyse command to pick up such changes.

Principal component analysis

Principal component analysis is complex and time-consuming. It is not practical to re-analyse all the data for each added event. Online updates use the matrices created by the initial analysis or by the last Reanalyse command to map new spikes onto the screen. The added spikes do not change the principal components.

Measurements

This method works identically online and offline.

Cluster on correlation and Cluster on errors

These methods take a snapshot of the templates in the Edit WaveMark dialog when the cluster windows are created and each time you Reanalyse the spikes. The online update mode uses these snapshots, not the templates in the Edit WaveMark dialog.

Menu commands

The clustering dialog has its own menu. All references to menu items in this section of the documentation are to the dialog menu, not the main Spike2 menu. Most menu items have short-cut keys that allow immediate use; you can also right-click in the dialog to open a context menu that duplicates many of the menu commands.

File menu	Apply changes, restore the original class codes from the data file, re-analyse the data, create interval histograms of the data and display the principal component waveforms.
Edit menu	Undo class changes, copy an image, copy data as text and edit interval histogram settings.
View menu	Everything to do with the appearance of the display and control over rotation, automatic scaling and ellipse and dot sizes.
Cluster menu	Commands for automated data clustering and setting class codes.

File menu

The file menu contains commands to apply changes, restore code, re-analyse the data, generate an interval histogram, display the principal components and select the principal components to use when clustering.

Apply changes

This copies the current state of the classes in the clustering window into the data file. Any events that were skipped during measurement analysis are treated as having a class code of 00. For this to work as expected you must not have changed the marker filter for the data channel since extracting the clustering values. If you do not use this command, all class changes you make in this dialog are lost when the dialog closes.

In addition to changing the events in the data file, the associated spike shape dialog will recalculate its templates based on the class assignments of the events in the marker filter and current time range set in the dialog. If you are working on-line, the newly created templates are copied to the 1401 for immediate use.

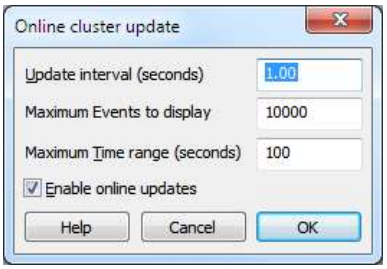
Restore codes

This command sets all the class codes to match the data file. You can use this to restore values after an unsuccessful clustering attempt. Another use is to pass class information between the measurement and PCA clustering dialogs if you have both open at the same time; use Apply changes in one and Restore codes in the other.

Reanalyse

This command opens the Principal Components, Cluster on measurements, Cluster on Correlations or Cluster on Errors set up dialog so that you can repeat the analysis.

Online Update



This File menu command is enabled if you are clustering a channel that is being sampled. There are four fields you can set:

Update interval
This sets the minimum gap between updates of the cluster window in seconds. A value of 0 updates the window as often as possible.

Maximum events to display
This limits the number of events to display in the window. At high event rates, if you set a large number and a short update interval, Spike2 may become slow to respond as it is spending a lot of time recalculating the display. You can delete all the spikes in the display with the Edit menu Delete online spikes command.

Maximum Time range
This field limits how far back from the current time to search for events. The combination of this and the Maximum events to display field limits the number of displayed events.

Enable online updates
You can enable and disable online updates here. If you disable them, the Reanalyse command will process the time range set for the Edit WaveMark dialog, or the entire file if the Edit WaveMark dialog is in At End mode.

Interval histogram

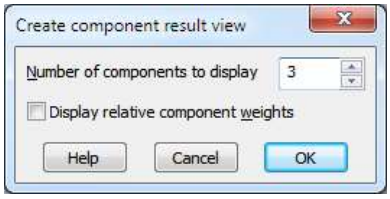
This command creates an interval histogram of the events in the clustering window. If an interval histogram of the events already exists, the previous values in the result view are replaced by a new analysis. Each class code present in the clustering window generates a separate channel in the histogram. The histogram bins are coloured to match the class colours except for code 00, which is filled with white rather than black. You can set the histogram width and bin width with the Edit menu INTH Settings command.

The first time you open this dialog in a Spike2 session, it opens in a default position. Subsequently, it opens in the last position it was moved to.

Minimum interval cursor
A vertical cursor is added to the interval histogram at the position of the minimum interval set for the clustering. If you drag the cursor in the histogram, this will change the minimum interval, and if the cluster display is set to link events closer than the minimum interval, the display will update on each change.

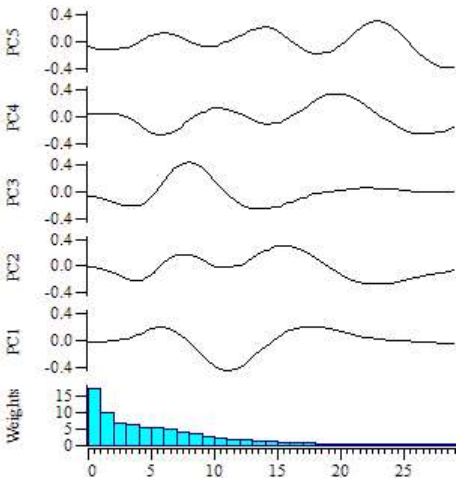
Restrict events with a user shape
You can restrict the events that are displayed in the INTH by enabling the user ellipse of the user shape and selecting it. This can be used to check that the events within a shape are not too close to each other.

Display Principal Components



This command is enabled for Principal Component Analysis. It creates (or updates if you have already created it) a result view holding the principal component waveforms and optionally the relative weights of each component. You can choose the number of components to display. The display order is in terms of their significance, based on their contribution to the variance of the original data. You will usually find that only the first 3 or four components make a useful contribution to clustering.

If you choose to display the weights, the x axis of the result view is the component number for the weights which corresponds with data points for the principal component waveforms. If you do not display the weights, the x axis is set to milliseconds with zero time representing the trigger point when the data was captured. If the spike data has multiple traces, each component shows all the traces, in order.

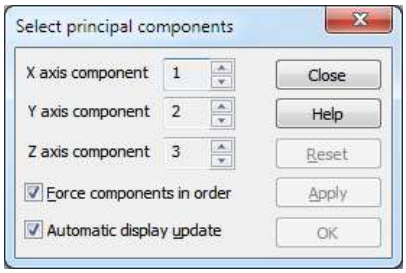


The tail end of the weights usually appears as a smooth curve trending to zero. The principal components that correspond to these weights are generally just noise.

The above assumes a waveform-based analysis. Analyses of amplitudes or amplitude ratios will not make any sense when drawn as waveforms.

Select Principal Components

This dialog sets the principal components to display. Reset selects components 1 to 3; these are usually the best ones to choose.



Force components in order

Check this box to make sure that the X component has the lowest number and the Z component has the highest.

Automatic display update

Check this to update the display on every change; this can be slow if you have a very large number of events. The Apply button can be used if you disable automatic updates.

Close

This closes the dialog. It does not save any changes you have made to the spike classes.

Edit menu

This contains the following commands:

Undo

This undoes the last change made to the class codes; only one change is saved.

Copy

This command copies the cluster window to the clipboard as a bitmap. The bitmap image is always created with z buffering enabled, so events that are further away from the viewer are hidden by events that are closer. This may differ from the screen image where the z buffer is only used if selected because it slows down drawing.

Delete online spikes

This command and the short cut key `Del` are enabled when you are working online. It deletes all the spikes from the clustering window.

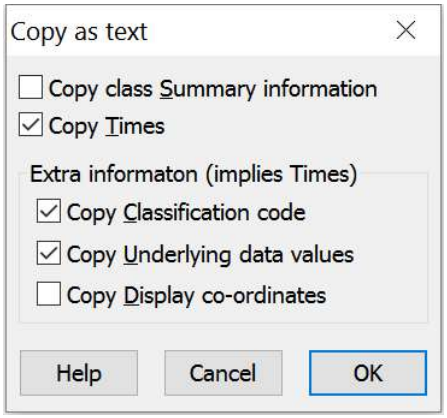
Copy As Text

Copy the cluster summary information and optionally all the raw cluster data to the clipboard as text.

Copy Cluster Values
Copy information about the cluster statistics to the clipboard as text.

INTH Settings
Change the settings for the interval histograms of each of the cluster classes.

Copy As Text



This menu command opens a dialog from which you can copy clustering data to the clipboard as text in a spreadsheet-compatible format. The output is in columns separated by tabs, with text enclosed in double quotes. There is one line of output for every visible event in the cluster window. The text output has the form:

```
6342 events in 3 classes: 01:3300 02:3002 00:40
"Time"  "Code" "X"          "Y"         "Z"
0.00016  "02"  -0.0107435  0.00288545 -0.0126408
0.03768  "03"  -0.0151009  0.00494189  0.00245431
0.17096  "02"  -0.00890594 0.00563056 -0.0174058
```

The time of each event is always output in the first column. You can also choose to copy the following, which are described in the order that they are added to the output line:

Summary information	This is the same as the line of text displayed at the bottom of the cluster window.
Copy Times	Check this box to list the times of all the events. The times of the events are automatically added if you select any of the Extra information items.
Classification code	Check this box to output the class code for each event, with the column title `"Code"`.
Underlying data	These are the x, y and z values that resulted from the principal component analysis or the measurements. The columns titles are `"X"`, `"Y"` and `"Z"`. If there are no z values, the z column is omitted.
Display co-ordinates	These are the underlying data values rotated into display co-ordinates. If there is no rotation these are the same as the data values. The x and y values set the event positions in the cluster window. The z value is used for the z buffer. The titles are `"View X"`, `View Y"` and `"View Z"`. If there are no z values in the original data, the `"View Z"` column is omitted.

Copy cluster values

This copies the clustering information for all visible cluster classes to the clipboard as text. The columns of text are separated by tab characters and the titles are surrounded by double quotes, which should make the data easy to import into other programs or to paste into a Spike2 Grid view.

```
"Cluste"N"   "X"     "SigmaX"Y"     "SigmaY'"Z"      "SigmaZ"XY"      "XZ"     "YZ"    "IsoDist
00     33    -2.0352 0.97429 0.133977 0.72856 0.286374 0.97417 -0.227977 -0.128655 0.192386 -1
01     2332  -1.00491 0.44436 0.401314 0.26529 0.211819 0.79801 -0.004795 0.012640 3 0.020282 47.7508
```

```
02     1036  1.49054 0.412861.39924 0.289621-0.323211.17921 -0.004020-0.038439 0.10258639.5792
03     645   0.733802 0.386060.448787 0.279842-0.289231.00992 -0.007358-0.02595  0.04998813.2346
04     2218  0.177234 0.36108 -1.208010.279931 0.008110 1.03576 -0.002905-0.004453 0.08771756.2281
```

The displayed data relates to the last clustering operation. The columns hold:

Cluster
This is the cluster code to identify the data for the row of data. Code 00 will usually be the events that do not fall into a useful cluster.

N
The number of items in the cluster.

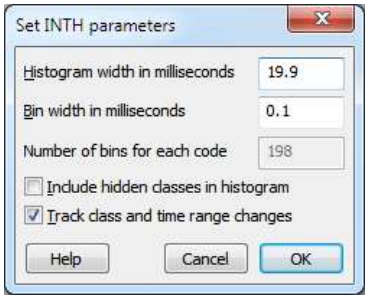
X, Y, Z
The position of the centre of the cluster in the clustering space. If there is no third dimension, only X and Y columns are generated.

SigmaX, SigmaY, SigmaZ, XY, XZ, YZ
These values the covariances of the points that make up each cluster. The Sigma values are the mean sums of squares of distances in the `X`, `Y` and `Z` directions (variances) from the centre of the clusters. The `XY`, `XZ` and `YZ` values are the mean of the cross products of the distances from the centres (covariances). If the clustering axes were independent, the cross terms would be zero.

IsoDist
This is the isolation distance. If a cluster has N members, this is the Mahalanobis distance from the centre of each cluster to the N^{th} nearest item that is not a member of the cluster. See: *Quantitative measures of cluster quality for use in extracellular recordings, N. Schmitzer-Torbert et al. / Neuroscience 131 (2005) 1–11* for the underlying mathematics. The larger this value, the more separated are the clusters. The square root of this value is analogous to the linear distance to the N^{th} nearest non-member item. The value is displayed as -1 if this is cluster 00 (unclassified) or if there are fewer than N non-member spikes. In the example above, cluster 03 is the least isolated, cluster 04 the most isolated.

LRatio
This measure is also described as L_{ratio} in *Quantitative measures of cluster quality for use in extracellular recordings, N. Schmitzer-Torbert et al. / Neuroscience 131 (2005) 1–11* which you should consult for a detailed explanation. Essentially, this is the number of non-member spikes you would expect to get in each cluster due to cluster overlaps per cluster spike assuming normally distributed clusters. If you multiply this value by the number of spikes in the cluster, you get what the paper refers to as the L value, which you could think of as the number of extra spikes you might have expected to get in the cluster from the other clusters due to chance if the data were normally distributed. With the data above, the L values for clusters 01, 02, 03 and 04 are: 8, 10, 24 and 0.4. This again indicates that cluster 03 is the least well-separated and cluster 04 the best-separated.

INTH Settings



This menu command opens a dialog that sets the parameters for creating an interval histogram based on the events in the cluster window. The histogram has multiple channels, one for each event class. When you close the dialog with the OK button, the interval histogram is created or updated if it already exists. The File menu Interval histogram command also uses the values set in this dialog.

You can set the desired histogram width and the width of each bin, both in milliseconds. The number of bins in the histogram is also displayed for your information, but this field cannot be edited.

If you set the Include hidden classes in histogram check box, all event classes are processed and displayed. If the check box is clear, only visible classes are included and the displayed classes will change when you show or hide classes in the cluster window.

If you check the Track class and time range changes check box, any operation that changes the visible events or the event classes causes the interval histogram to recalculate.

View menu

This menu (plus suitable entries depending on your current selection) can also be activated as a context menu by right-clicking the mouse in the cluster window.

Autoscale, Equal Scales, Rescale

The events to be draw each have an x, y (and possibly z) position. When you rotate the display, these positions are multiplied by a rotation matrix to generate an x', y' and z' display co-ordinate. If your data had very different ranges of x, y and z (for example when using measurements and not normalising the result), as you rotate the display, the width and height of the result will change dramatically. To counter this, we can also apply a display scaling that looks at the range of values in x' and y' and scales the result so that they fill the available space.

Autoscale

When you rotate the cluster window you may find that the data tends move out of the viewing area. Turn this option on to rescale the view to fit all visible events, axes and class ellipses in the window. The view rescales automatically if you change the displayed classes, the displayed time range or the window rotation. If the Equal scales option is set, the larger of the two display axes sets the scaling for both axes.

The state of Autoscale is indicated by a button in the toolbar.

If your interest is purely in the clusters and not at all in the actual value used for clustering, you can often remove the need for scaling by checking the normalise box in the set up dialog for the analysis you used to create your cluster data. This forces the x, y and z values to have a zero mean and a unity variance.

Equal Scales

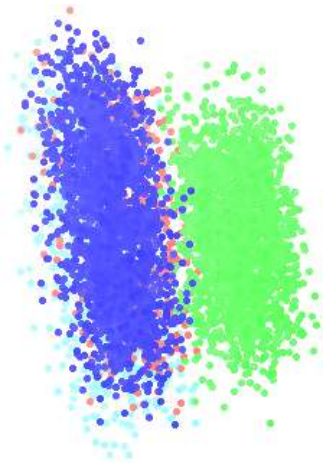
If you enable this option, the cluster window is scaled so that the width and height of the window have the same numeric size. This makes ellipses easier to manage, as they will preserve their shape when rotated. The more different the scaling of the x and y display axes, the more the ellipses will distort when rotated. Put another way, whichever of x' or y' has the greater numeric range sets the scaling for both the x and y directions.

Rescale

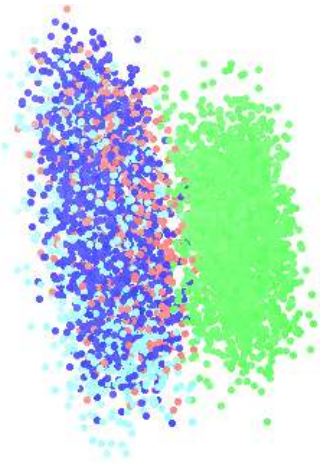
This command scales the display window so that all the displayed event data, axes and any class ellipses are visible. If the Equal scales option is set, the larger of the two display axes sets the scaling for both axes. If axes are enabled, the axis origin is also displayed. You would only need to use this command if Autoscale were disabled.

Z Buffer

With this option on, events nearer to the viewer draw on top of events that are further away. With the Z buffer off, events are drawn in time order. Turning this option on increases the illusion of three dimensions, but at the cost of drawing more slowly. If you have a very large number of events you may be able to improve the drawing speed if this option is off. The keyboard short-cut for this option is `Ctrl+B`.



Z Buffer On



Z Buffer Off

The use of the Z Buffer is recommended if you use the Colour Fade with Z or variable dot size options. You should only need to turn it off if you have a very large number of points to display and the display response has become too sluggish to be useful.

When you copy the image to the clipboard, this option is always turned on. The previous state is restored once the bitmap has been copied.

Dot size

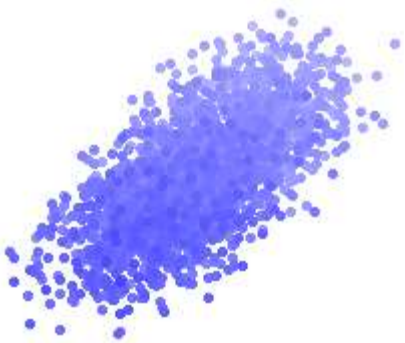
The events in the cluster window can be drawn at six sizes, numbered 0 to 6 (was 0 to 4 before [10.15]); the larger the size, the larger the dot and the longer it takes to draw the events, especially if the Z Buffer is enabled. You can also choose Varies, which draws each event at a size that depends on the notional distance of the event from the viewer.

If you are using circular dots, this makes no difference until you reach size 3. Smaller sizes do not produce circles. To make drawing fast, circles are squares with the corners removed.

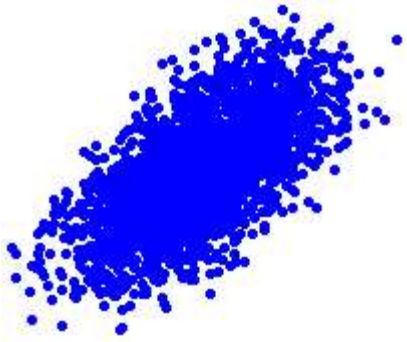
The Varies method can be effectively combined with Colour Fade and the Z Buffer to enhance the impression of three dimensions.

Colour Fade with Z

Each displayed event has a notional x, y and z position with respect to the viewer. The x and y positions are used to set the screen position. If you enable this option, the z position is used to fade the event colour to the background colour as the event becomes further away from the viewer. The scaling is based on the maximum distance between the nearest event and the furthest (in the z co-ordinate) at the last time that the display was scaled. If Autoscale is enabled, this will always be set for the current data. If Autoscale is not enabled, this is the scale set by the last Rescale or Equal Scales operation. The keyboard short-cut is `Ctrl+F`.



Colour Fade On



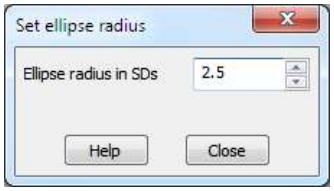
Colour Fade Off

There is a small time penalty for using this option, which might become significant if you have a very large number of data points. This option works best with the Z Buffer enabled.

Circular Dots

You can choose to display square dots or circular dots. Circular dots become non-square at size 3, and are really only successful at size 4 (well you try making a circle out of solid colour pixels at sizes of 3x3 pixels or less!) Circles are slower to draw than squares, so if you have a huge number of points (tens of thousands) you may find that drawing as squares or at size 0 is faster. The short-cut for this command is `Ctrl+L`.

Ellipse radius



This command opens a dialog in which you can set the size of the class ellipses that are drawn around each class cluster. The size is set in terms of the Mahalanobis distance, which is a multi-dimensional analogue of the standard deviation. If the data were normally distributed around the mean position, you would expect 68% of the events to be within the Mahalanobis distance, 95% of the events to be within twice this distance and 99% to lie within three times this distance.

You can set the distance to a value in the range 1.0 to 4.0 times the distance. The default is 2.5 times, which seems to be about right. If the class ellipses are visible, any change made in the dialog has immediate effect.

This value also sets the limit for the distance from the centre of an ellipse for an event to belong to the event class that is used when clustering.

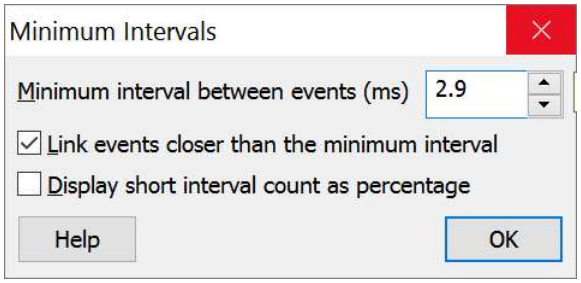
Minimum Interval

You can open this dialog from the Clustering dialog View menu Minimum Interval... command, or more conveniently by right clicking on the Link close events arrow in the Clustering dialog toolbar.

The minimum interval tool lets you identify pairs of events that are too close together to be in the same class. You can use this tool in two ways:

1. With no ellipses or user-defined shape selected events in the same class that are too close together are linked by arrows.
2. With ellipse(s) or user-defined shapes selected, all events that lie within the selections are considered and events that are too close to be the same class are linked by arrows.

You can navigate to the first event of each linked pair by clicking near an arrow. If you have selected an ellipse or user-defined shape (as left-click prepares for a drag) you must right-click and select Find Short Interval unless you are close to an arrow when we assume you intend to hit the arrow.



Minimum interval between events (ms)
This field sets the minimum interval, in milliseconds, to apply when the Link events closer than the minimum interval is checked.

Link events closer than the minimum interval

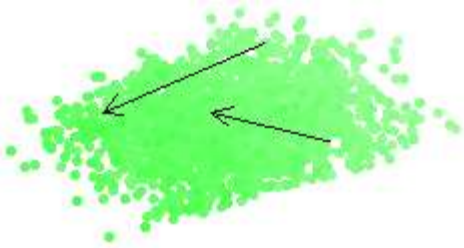
If you check this box, pairs of events that are closer than the minimum interval field are linked by a line with an arrow pointing from the earlier event to the later event. You can left-click on the events to move to them in the Edit WaveMark dialog. If the Scroll time view option in the Edit WaveMark is set, the associated time view will scroll to this event. The arrows are drawn on top of the data, even if the belong to events that are buried inside a cluster. With this box checked, the status information at the bottom of the clustering window is extended to display information about short intervals.

Display short interval count as percentage

Check this box to display the short interval count as a percentage of the applicable events, leave it unchecked to display it as a count.

Find Short Interval

If there are a lot of events, even though the arrow that links them is drawn on top of everything else, it can be difficult to find the event without zooming in. There is an easy way. Click within a few pixels of an arrow or right-click within 100 pixels and select the Find Short Interval command in the pop-up context menu. This will then find the nearest line to the click point and then make the event that starts that interval the current event. This event is displayed in the associated Edit WaveMark window, and if that dialog is set to scroll the time window, the time view will also scroll to show the event.



Short interval arrows

Link to INTH window

If you display the INTH window, a vertical cursor appears at the minimum interval you set in this dialog. Changes made in this dialog set the cursor position. You can also drag the cursor in the INTH window to change the minimum interval.

Use Time as Z

You can choose to display the clustering data using Time as the Z axis. This is not useful when clustering data, but can help in visualising changes with time. To do this, the times of all the data points are mapped into the range -1 for the oldest event to +1 for the newest. Applying and removing this option can take a noticeable time if you have a very large number of events. This option is not allowed if you are in online update mode, or if you have had an online update.

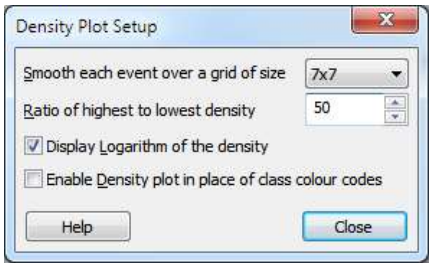
If you select this mode and the axes are visible, the Z label on the axis changes to T.

The short-cut for this is `T` (or `Shift+Ctrl+T` which may be removed in future).

Density Plot

There are two basic display modes: Density Plot and Class Colours. This command chooses between the two modes. In the Class Colours mode, each event is drawn as a dot in the colour of the event class. In Density Plot mode, each pixel of the display is drawn in a colour that shows then number of events at that point in the display.

Density Settings



This command leads to the Density Plot Set up dialog, which controls how the event density is converted into a density display.

Smooth each event over a grid of size
To convert the events from individual points to a smoothly varying density plot, each event is spread over a square grid of pixels that is centred on the event and the result of this is summed over all the events. You can choose from a 1x1 grid (no smoothing) up to a 7x7 grid.

Ratio of highest to lowest density
The highest density found is allocated to one end of the density colour scale. This figure sets lowest relative density to the highest density that is visible as a contrast with the background. Smaller numbers will make individual outlier events merge into the background.

Display Logarithm of the density
If all your clusters have similar densities, then it makes sense to map densities to your colour map linearly. However, if you have clusters of widely differing densities, it can make more sense to display the logarithm of the density.

Enable Density plot in place of class colour codes
This is exactly the same as the Density Plot command and allows you to swap between a display of class colours and density from within the dialog.

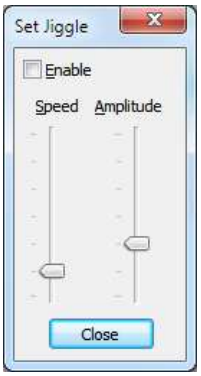
Density Colour Map

This command allows you to choose the colour scale that is used to indicate density. This dialog is exactly the same as for the sonogram colour scale as described for the View menu.

View along axis

This command (enabled with three dimensional data only) display a pop-up menu with three options: X, Y and Z. Choose one of these options to rotate the cluster view to point one of the three data axes upwards and cancel any display rotation. It does not cancel any display jiggle. The keyboard short cuts for these commands are `X`, `Y` and `Z`.

Jiggle Display and Jiggle Settings



With three-dimensional data, it is sometimes easier to visualise the clusters if the display is "jiggled" around the current view angle. If you enable the display jiggle, this is equivalent to using the mouse to rotate the display in a circle around the centre of the displayed cluster area. You can enable and disable this rotation with the Display Jiggle command or with the `Ctrl+J` keyboard short cut.

The Jiggle Settings dialog gives you additional control over the jiggle. The Amplitude slider sets the radius of the notional circle and the Speed slider sets how fast to rotate around the centre of the circle. The Enable check box starts and stops the display jiggle. The keyboard short cut to open the dialog is `Ctrl+Shift+J`. The image to the right has been adjusted for height; the dialog is much taller than shown here.

Clear rotation

This command restores the cluster window to its original, non-rotated state and removes any continuous rotation set by the buttons around the rotation slider controls. It also cancels any display jiggle.

Show Sliders

You can choose to show or hide the slider controls and associated buttons that rotate the clustering window around the x, y and z axes. The dialog size will grow, if required, when you turn on the sliders.

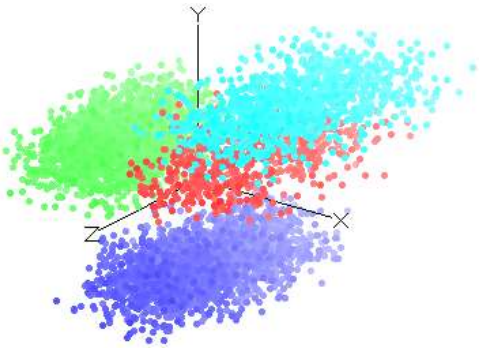
Show Time Range

The Time range control area can be shown and hidden as required. The dialog will change size when you show and hide this area. This area allows you to display events that fall within a restricted time range and track how the clusters change with time.

Show Axes

You can display axes in the cluster window. They are labelled X, Y and Z and are drawn along the principal axes of the measurements. The keyboard command to show or hide the axes is `Ctrl+X`.

If the Z buffer is enabled, the axes are draw with true depth, that is events that are further away will be behind the axis lines and events that are nearer will be in front, as in the image. The X, Y and Z used to label the axes are always drawn on top of the events. If the Z buffer is not enabled, the axes are always drawn on top of the data. In Density Plots, the axes are always under the data.



Axes with the Z Buffer active

Cluster menu

This menu contains commands related to manual and automatic assignment of class codes to event clusters.

Set codes

This command is enabled when there are selected shapes in the cluster window. It opens a dialog in which you can set the codes of all events that lie inside and/or outside selected shapes. When you open the dialog the inside code is set to the most common code of the selected events unless this code is 00, in which case the lowest unused code is set. The outside events code remembers the last code you set.

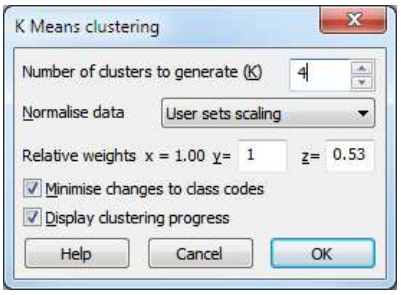
The check boxes control which set of events has its codes changed. The edit boxes on the right set a code to apply as two hexadecimal digits (0-9, a-f or A-F). If you set a single character in the field, a 0 will be inserted in front of it. Leaving the field blank is the same as clearing the check box and no class codes will change.

Click OK to change the displayed events. This has no effect on the original spikes in the data file. You must use the File menu Apply command to change their class codes. You can undo this command with `Ctrl+Z`.

Clear all visible codes

This command is here as a quick way to set the class codes of all visible events to 00. It has a short-cut key combination `Ctrl+Shift+0`. In fact, you can set all visible events to any code from `00` to `09` using `Ctrl+Shift+0` through `Ctrl+Shift+9`. You can undo the effect of these commands with `Ctrl+Z`. Beware that Microsoft have grabbed `Ctrl+Shift+0` in Vista and later for IME use, see here for how to defeat this.

K Means from existing and K Means



These commands open a dialog in which you can assign events to classes automatically. The K Means from existing command uses the existing cluster centres as the seeds for the K Means algorithm. The K Means command runs the algorithm 10 times (or until you stop it) with random seeds and chooses the result that has the best ratio of cluster separation to cluster size. You can find details of the algorithm at the end of the chapter.

The K Means algorithm is a well-known and fast clustering method that is effective when the number of clusters is already known, the clusters are spherical and are of a similar size. Put another way, if the clusters are not spherical (or cannot be made spherical by scaling) or are of very different sizes, KMeans will not give useful results.

Number of clusters to generate

This field sets the number of clusters to generate. It is preset to the number of visible, non-zero class codes in the data set. We allow you to set from 1 to 20 clusters. This field is disabled if you have selected the K Means from existing command.

Normalise data

This field sets the strategy to use to make the clusters spherical. You can choose from:

None	No scaling: using the measured values directly. This is equivalent to the User sets scaling option with all weights set to 1.
Use existing clusters	The relative scaling of the x, y and z data is deduced from the shape of the current class clusters.
Use visual scaling	We assume that you have arranged the display so that the clusters appear circular. The data is clustered based on the visual appearance of the data in the window.
User sets scaling	Extra fields appear in which you can set the relative weights to give to the y and z values relative to the x values. For principal component analysis, these fields are initially set to values based on the relative weights of the components. For measurement-based clustering, the relative weights are initially set to 1.

Minimise changes to class codes

If you check this box, the event class codes are chosen to maximise the number of events that keep the original code. If you do not check this box, class codes are assigned so that the lowest free codes are used. This is always checked for K Means from existing.

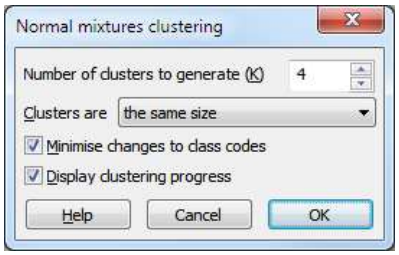
Display clustering progress

Check this option to follow the clustering in detail. The K Means command updates the display after each iteration regardless of this setting. This option slows analysis.

Running the command

Click OK to run the command. This may take some time if there is a large number of events and K is large. When the analysis is complete the command calculates the ratio of the average separation of the clusters to the average cluster size. This is displayed in the status line as the J3 value.

Normal Mixtures from Existing and Normal Mixtures



These commands open the Normal Mixtures dialog in which you can assign events to classes automatically. The Normal Mixtures from existing command uses the existing clusters as the seeds for the Normal Mixtures algorithm. The Normal Mixtures command runs the algorithm 10 times (or until you stop it) with random seeds and chooses the result that has the maximum likelihood of fitting the data.

The Normal Mixtures algorithm assumes that the probability density of data points around each cluster centre follows a multivariate normal distribution. This is a more general approach than the K Means algorithm, as it does not require spherical clusters and can often give results that seem more intuitively correct. However, it is a more complex algorithm than K Means and takes noticeably longer to run.

Number of clusters to generate

This field sets the number of clusters to generate. It is preset to the number of visible, non-zero class codes in the data set. We allow you to set from 1 to 20 clusters. This field is disabled if you have selected the Normal Mixtures from existing command.

Clusters are

You can use this field to place restrictions on the clusters. The choices range from least to most restrictive. You can choose from:

Independent	The clusters are independent of each other and have different sizes, shapes and orientations. This is the most general setting and takes the longest to run. It is usually most successful when run with existing clusters as the starting point. Using random starting points can generate unexpected results.
the same size	The clusters are assumed to be the same size and shape. This is often true, especially with principal component data, so this is the default value.
the same size and axially aligned	The clusters are assumed to be the same size and shape, and the principal axes of the cluster shapes are aligned to the x, y and z axes of the data.
the same size and spherical	The clusters are assumed to be the same size and shape and are spherical. This is the most constrained setting and is the most similar to KMeans clustering.

Minimise changes to class codes

If you check this box, the event class codes are chosen to maximise the number of events that keep the original code. If you do not check this box, class codes are assigned so that the lowest free codes are used and ordered so that classes with more events have lower codes. This is always checked for the Normal Mixtures from existing command.

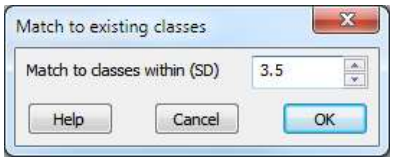
Display clustering progress

Check the box to follow the clustering in detail. The Normal Mixtures command updates the display after each iteration regardless of this setting. This option slows analysis.

Running the command

Click OK to run the command. This may take some time if there are many events and the number of clusters is large. During analysis, a progress bar gives you the option to stop analysis early. When the analysis is complete the command displays the log likelihood value per point for the solution. See the discussion of the Normal Mixtures algorithm for more information.

Match to classes



This command opens a dialog from which you can use the current cluster statistics to assign visible events to the nearest visible cluster and mark outliers as code 00. The typical use of this command is to clean up manual cluster assignments in situations where the K Means from Exsting or Normal Mixtures from Existing commands would make too drastic a change.

The basis idea is that each visible event is assigned to the cluster that it is most likely to belong to. The shape of each cluster is taken into account, so a cluster need not be spherical or aligned with the x, y or z axes. Probability is measured in terms of the Mahalanobis distance, which is the multi-dimensional equivalent of standard deviation.

The Match to classes within field sets the maximum separation in terms of the Mahalanobis distance between a cluster and an event, for the event to be considered as a possible cluster member. Events that do not belong to any cluster are given code 00. The matching does not pay any attention to the number of items in each class (unlike the Normal Mixtures from Existing command). The matching does take into account the fact that the class ellipsoids may not be aligned with the x, y and z axes (unlike the K Means from Existing command).

Getting started with clustering

We would suggest that you start by trying out principal component analysis, as this avoids you making subjective decisions about which features of your data are more important than others. It will also tend to produce clusters that are suitable for use with automatic clustering methods.

In the Edit WaveMark dialog choose the Analysis menu Principal Components command. This will open the cluster dialog and display the data, ready to cluster. It is a good idea to check that visible clusters are stable over time by using the Time range control area. There are three basic ways to cluster the data:

Automatic	If you have reasonably well-defined clusters that are, or can be made more or less spherical by scaling, and that contain similar numbers of events, then the K Means command will be able to separate them for you. This will often be the case with principal component values. If K Means does not make sensible choices, you should try the Normal Mixtures algorithm. This is slower, but can give better results.
Manual	If the data is chaotic, the only sensible course may be to manually position ellipses or user-defined shapes and set codes. You can position an ellipse on a cluster centre by right clicking at the centre and selecting a small, medium or large ellipse from the context menu.
Semi-automatic	If the automatic methods almost do what you want, you may be able to help them by marking the centres of clusters manually, then using one of the …from existing commands to iterate from your centres.

Whichever method you use, you may also wish only to mark events that you feel "certain" about. You can set events that are too far from the cluster centres to code 00 with the Match to classes command.

Remember that changes made in the clustering dialog have no effect on the original data until you use the File menu Apply command.

K Means algorithm

The K Means algorithm has many variants, but the basic idea is based on the following iteration sequence.

1. Given the current set of K cluster centres; assign each event to the nearest centre.
2. Recalculate the K cluster centres based on the new assignments.
3. If the centres changed in the iteration, go back to 1, else done.

The algorithm always converges, but there is no guarantee that it converges to the optimum solution. The result depends on the original cluster centres. The K Means for existing command takes the current set of cluster centres as the starting point. The K Means command assigns events to clusters randomly before starting the algorithm and repeats the entire procedure 10 times. It chooses the solution that maximises the variance of the distance between all clustered points and the centre of gravity of all the points divided by the variance of the distance between each point and the centre of its cluster. The better the clusters are separated, the higher the value. This measure is known as J3 in some literature.

Once we have iterated to a solution the next task is to remove outlier points. We generate the statistics for each cluster and based on the assumption that the data follows a multivariate normal distribution, we set all points that lie more than 3 times the Mahalanobis distance away from the centre of the their assigned cluster to have class code 00. We then run the K Means algorithm again, staring with the current centres. This usually converges in one or two iterations because removing outliers should make little difference to the distributions.

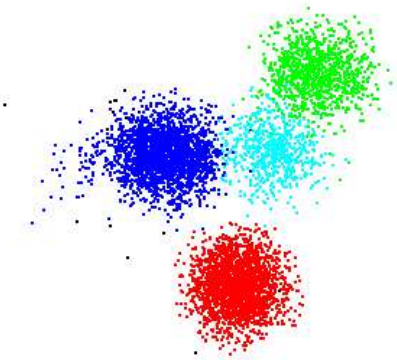
Limitations of KMeans

The KMeans algorithm assumes that the clusters are spherical and similar in size. The clustering is based on the distance between a each point and the centre of each cluster. It is often the case, for example with clustering based on measurements, that the scaling of the axes are very different. You can compensate for this to some extent by normalising the data before clustering (shifting and scaling so that the data on each axis spans a similar range about zero). However, this will probably not make the clusters spherical.

Assuming that the clusters are more or less the same size and shape, they can be made spherical by stretching or shrinking the y and z axes relative to the x axis. This is done by applying weighting factors to the y and z axis data (the x axis data is assumed to have a weighting factor of 1.0). When data comes from the Principal component analysis, we guess the weights based on the results of the analysis. When data comes from measurements, the weights all start as 1.0 and you can adjust them by hand. You can also choose to weight the data based on how it looks on the display, or apply weights calculated so as to make the existing clusters as spherical as possible, or to apply no weighting.

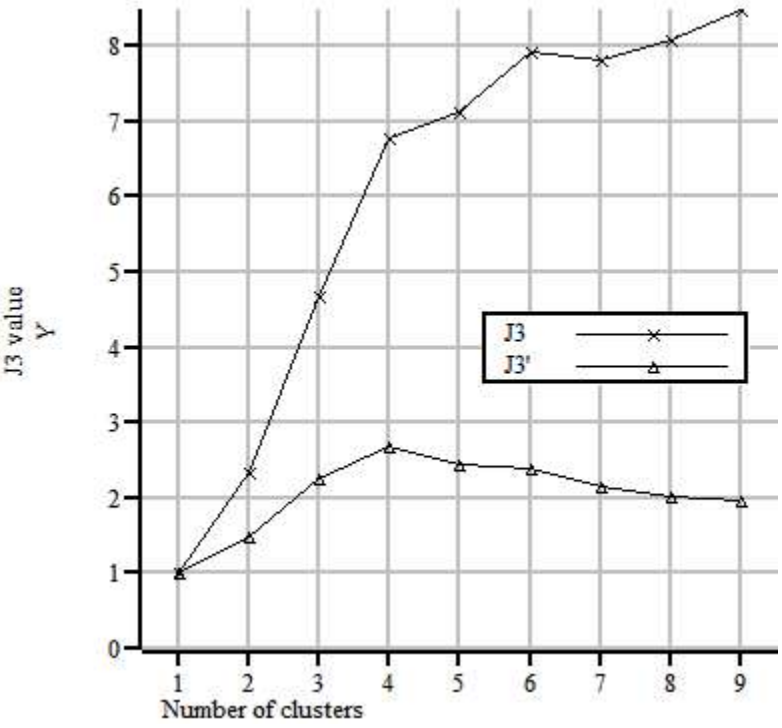
The KMeans algorithm also assumes that the x, y (and z) values are independent. If they are not, your data may look like ellipsoids that are not aligned with the x, y or z axis. If your data looks like this, you should probably be using the Normal Mixtures algorithm for clustering.

J3 clustering measure



The J3 measure is displayed at the bottom of the clustering window after running the KMeans algorithm. It measures how compact the clusters are compared to the overall spread of the data. In this example there are 4 fairly clear clusters (although, at a stretch you could merge the two in the upper right corner together to make three clusters). The J3 values when analysing the data with different numbers of clusters are given in the graph. The data was normalised using the existing cluster sizes, so we repeated the KMeans process until the J3 value stopped changing.

The J3 value is given by J2/J1 where J2 is the sum of the squares of the distances of all clustered points from the centroid of all the points and J1 is the sum of the squares of the distances of each point from the centroid of the cluster it belongs to. Dividing the data into more and more clusters reduces J1, so J3 increases with cluster count



In this case, the J3 value increases sharply from 1 to 4 clusters, then settles down at a fairly constant slope. Ideally, we would like a measure of clustering efficiency that would be a maximum at the most likely number of clusters.

We can estimate the value of J3 when data is divided up into more clusters than actually exist with the following simplistic argument. Consider the case where there are N points spread at a constant density through a m dimensional space. If this is divided up into k clusters of N/k points of similar (but arbitrary) shape, the radius (linear size) of each cluster will be proportional to $(N/k)^{1/m}$ and the variance of the distances from the centre of each cluster (J1) will be proportional to $(N/k)^{2/m}$. By the same argument, the J2 value will be proportional to $N^{2/m}$ with the same constant of proportionality, so J3 is given by $k^{2/m}$. What we are saying is that in a case where there is no way to divide data up into clusters (because it is spread through the space at a constant density), we know what the graph of J3 looks like.

In addition to J3, we also give the value J3', being J3 divided by $k^{2/m}$. This scales J3 by the value you would expect if all the points were uniformly distributed. In our case, we have m=2 (x and y axes) or m=3 (x, y and z axes). However, if one dimension makes no, or only a small contribution to the sorting, the effective number of dimensions is reduced so the J3' value can only be taken as a guide.

You can see that with this data, the J3' value peaks at 4 clusters, indicating that this is likely to be a reasonable interpretation of the data.

Normal Mixtures algorithm

The Normal Mixtures algorithm assumes that the density of events around a cluster centre follows a multivariate normal distribution and that the number of clusters (K) is known. Clusters may be independent, or have the same shape, or have the same shape and be axially aligned or be spherical. We calculate the probability that an event belongs to a particular cluster based on the distances from each cluster, the shape of each cluster and the number of events in the cluster.

The algorithm is adapted from the Normal Mixtures algorithm described in *Clustering algorithms* by P.A. Hartigan, published in 1975 by Wiley, ISBN 0-471-35645-X. This process iterates towards a local maximum in the probability density of the set of events given the clusters. The value we return is the log of the probability density divided by the number of events (see below). In outline, the algorithm proceeds as follows:

1. Make an initial assignment of the probabilities of each event belonging to each cluster. This can be based on the current classes, or can be done by randomly assigning events to clusters.
2. Using the probabilities of cluster membership, calculate the centres, shapes and number of events in each cluster.
3. Work out the probabilities of each event belonging to each cluster based on the new cluster centres and shapes.
4. Work out the most likely class code for each event. If any event is more than 3 times the Mahalanobis distance from the centre of the most likely class, set class 00.
5. If the most likely class assignment has not changed since the previous iteration or the iteration count is too high then stop, otherwise go back to step 2.

As set out, the algorithm converges slowly, so there are additional steps (not described here) required to accelerate the process. There is always the possibility that one or more classes will not be represented when iterating has finished. In this case, we take the class with the largest number of members and split this in two, then repeat this process until we have the desired number of classes, then we restart the iteration process.

There is no guarantee that any solution is the best as the outcome depends on the initial assignment of probabilities. If you choose to run the algorithm with no use of existing classifications, we repeat the process 10 times with randomised initial conditions and we choose the result that has the maximum probability density.

The mathematical details

We want to partition our M events into the k most likely clusters. Given a partition, for each of the k clusters, we can work out a probability density. For the j^{th} cluster we can write this as $N_j * W_j$ where N_j is the normalised multivariate Normal distribution and W_j is the number of events in that cluster.

The normalised multivariate Normal distribution for a point at position $\boldsymbol{x}$ relative to the centre of the distribution is given (in matrix notation) by:

$$N(\boldsymbol{x}) = exp(-½\, \boldsymbol{x}'\, \mathbf{Cinv}\, \boldsymbol{x})/((2\pi)^n\, det(\mathbf{C}))^{½}$$

where:

n	Is the number of dimensions of the distribution (in our case this is 2 or 3).
$\boldsymbol{x}$	Is a vector length n holding the position relative to the centre of the Normal distribution.
C	Is a covariance matrix of size n by n. We construct the covariance from the positions of all the points in the cluster relative to the centre of the cluster.
Cinv	Is the inverse of the covariance matrix.
$det(\mathbf{C})$	Is the determinant of the covariance matrix.

We have M data points that we wish to cluster. We define the value G_i for the i^{th} of our M points as:

$$G_i = \sum_j N_j(i) * W_j$$

where the sum for j is over the k clusters and $N_j(i)$ is the normalised multivariate Normal distribution value at data point i relative to the centre of cluster j. This value G_i is the probability density for the i^{th} point. We want to maximise the probability density for all points, which comes to maximising:

$$\prod_i G_i$$

where the product is over all M points. This number is likely to be inconveniently large, so we take its logarithm:

$$\ln(\prod_i G_i) = \sum_i \ln(G_i)$$

The value that is reported after the normal mixtures algorithm runs is this value divided by M, being the log probability density per point.

Mahalanobis distance

When describing clustering and the ellipses drawn to illustrate class boundaries we have referred to the Mahalanobis distance (named after P.C. Mahalanobis who described this measure in the 1930s). Given multivariate data values for which the values in each variable are normally distributed around a mean, this measure allows us to define boundaries of constant probability around the multi-dimensional centre of the distribution.

In one dimension, the normal distribution leads to a probability density $P(x)$ that is proportional to:

$$P(x) \ \alpha \ \exp(-\tfrac{1}{2}x^2/\nu)$$

where x is the distance from the mean and ν is the variance (σ^2). In two and three dimensions, this becomes:

$$P(x, y) \ \alpha \ \exp(-\tfrac{1}{2}(x^2/\nu_{xx} + 2xy/\nu_{xy} + y^2/\nu_{yy}))$$
$$P(x, y, z) \ \alpha \ \exp(-\tfrac{1}{2}(x^2/\nu_{xx} + 2xy/\nu_{xy} + y^2/\nu_{yy} + 2yz/\nu_{yz} + z^2/\nu_{zz} + 2zx/\nu_{zx}))$$

where x, y and z are the distances from the means and ν_{ab} is the joint variance in the two components a and b. These equations generalise to any number of dimensions and are often written using matrix notation:

$$P(\boldsymbol{x}) \ \alpha \ \exp(-\tfrac{1}{2}\, \boldsymbol{x}'\, \mathbf{Cinv}\, \boldsymbol{x})$$

where $\boldsymbol{x}$ is the vector of distances from the mean, $\boldsymbol{x}'$ is the transpose of this, and **Cinv** is the inverse of the covariance matrix.

The boundaries of constant probability for one, two and three dimensions satisfy the equations:

$$x^2/\sigma^2 = r^2$$
$$x^2/\nu_{xx} + 2xy/\nu_{xy} + y^2/\nu_{yy} = r^2$$
$$x^2/\nu_{xx} + 2xy/\nu_{xy} + y^2/\nu_{yy} + 2yz/\nu_{yz} + z^2/\nu_{zz} + 2zx/\nu_{zx} = r^2$$

where r is a constant. By choosing suitable values of r we can set boundaries within which we would expect to find a certain proportion of the data. In one dimension, the boundary is a distance from the centre, in two dimensions it is an ellipse and in three dimensions this is an ellipsoid. When r is 1.0, we refer to the distance of constant probability density as the Mahalanobis distance. In the one-dimensional case, this is equivalent to the standard deviation.

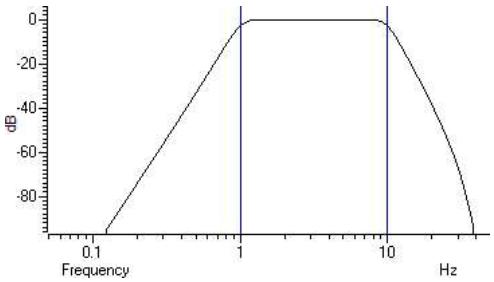
18: Digital filtering

Digital filtering

FIR and IIR filters

Filtering is used to remove unwanted frequency components from waveforms and can also be used to differentiate a signal. In Spike2 we provide you with two basic types of filter: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR). Both types of filter have their advantages and disadvantages.

IIR filters



These are similar to analogue filters; we design them by mapping analogue Butterworth, Bessel, Chebyshev filters and resonators into their digital forms. They have advantages:

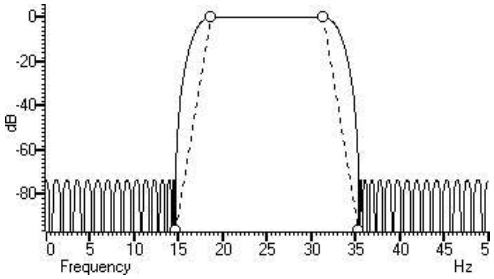
- They generate much steeper edges and narrower notches than FIR filters for the same computational effort.
- They are causal; they do not use future data to calculate the output, so there is no pre-ringing due to transients.

They also have disadvantages:

- IIR filters are prone to stability problems particularly as the filter order increases or when a filter feature becomes very narrow compared to the sample rate.
- IIR filters impose a group delay on the data that varies with frequency and increases with filter order. This means that they do not preserve the shape of a waveform, in particular, the positions of peaks and troughs will change.

The output of an IIR filter may take a long time to settle down from the discontinuity at the start (transition from no data to the supplied data).

FIR filters



We describe FIR filters in terms of frequency bands: *pass bands*, *stop bands* and *transition gaps*. You define a filter by the arrangement of bands and the corner frequencies of each band. FIR filters have these advantages:

- They are unconditionally stable as they do not feedback the output to the input.
- There is no phase delay through the filter, so peaks and troughs do not move when data is filtered (this is called *linear phase* in the literature).

They also have disadvantages:

- They are poor at generating very narrow notches or narrow band pass filters.
- The narrowest frequency band or band gap is limited by the number of coefficients (we allow up to 2047).
- FIR filters are not causal; they use future as well as past data to generate each output point. A transient in the input causes effects in the output before the transient.

If your FIR filter has `n` coefficients, the first and last `n`/2 output points are estimates due to the discontinuity at the start and end of the data.

So which to choose?

If you want a differentiating filter, you have no choice as we have not implemented IIR differentiators. Unless one of the FIR disadvantages is a problem, you will likely have fewer unexpected effects with an FIR filter.

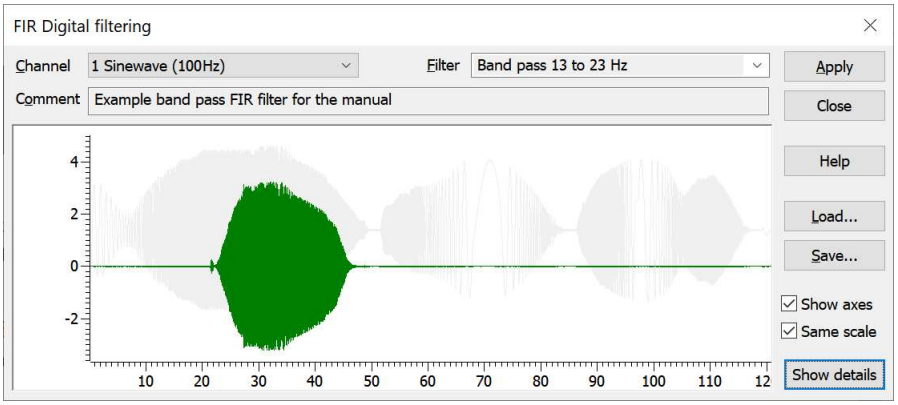
However, there are circumstances in which only an IIR filter will do. If you need a high Q notch filter or resonator, then use an IIR filter. If you are interested in small changes just before a large discontinuity, only the IIR filter will help you. However, make sure that you understand the disadvantages of IIR filters before you depend on their output.

Digital filter dialog

The digital filter dialog opens in two situations:

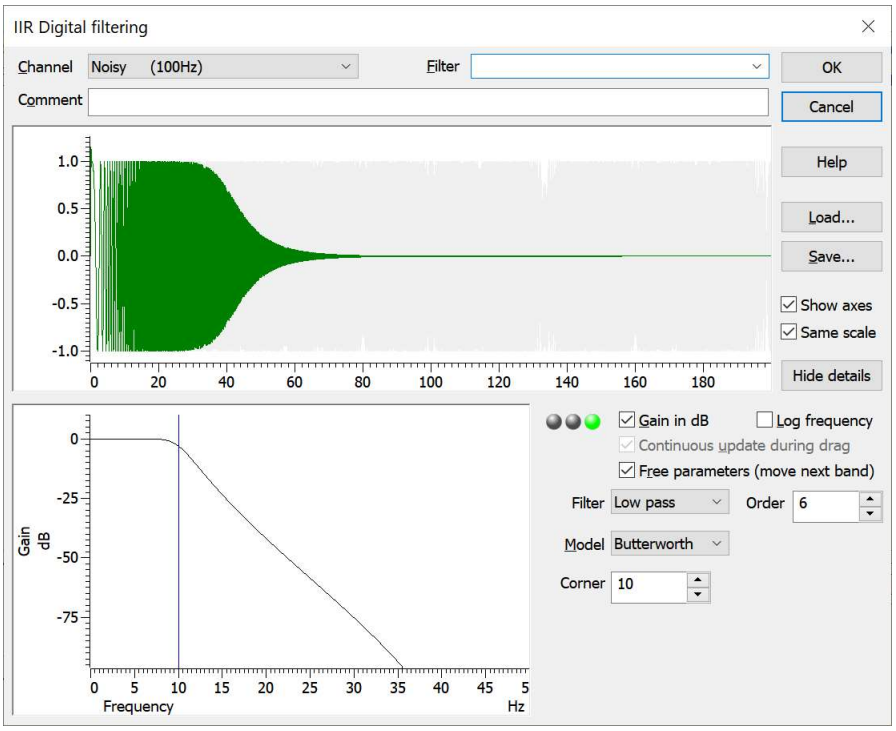
1. When you have a data file open that contains waveform channels you can use the Analysis menu Digital filters command or right click on a suitable channel and select the either the Finite Impulse Response (FIR) filters or Infinite Impulse Response (IIR) filters option.

   Apart from the dialog title, the initial dialog display is the same for IIR and FIR filters and displays data from the selected channel. The dialog opens with the details panel closed:



2. When you are in the Sampling Configuration channels dialog and are working with a Derived channel or are adding real time processing to a Waveform channel and either Add an IIR filter or double-click an IIR filter in the list of processes.

   In this case, the IIR variant of the dialog opens with the details panel open. In this situation, there is no existing data, so an example waveform is generated of a sinusoid that increases in frequency from 0 Hz to half the waveform sampling rate to show the effect of the filter. There is no Apply button; you are designing a filter to process incoming sampled data:



Filter dialog controls common to FIR and IIR filters

You can apply one of twelve stored digital filters to a waveform channel, or you can create your own digital filter. You can also load and save additional sets of filters from the dialog.

The dialog shows the original waveform in grey, and a filtered version in the colour you have set for waveform data in a time view. Whenever you change the filter, the display updates to show the effect of the change.

Show axes
You can choose to Show axes for the original data and for the filtered version. The axis for the original data is always on the left. If a separate axis is required for the filtered data, it is drawn in the filtered data colour on the right.

Same scale
Initially, the filtered data is drawn at the same scale as the original data. However, sometimes this is inconvenient, for example when high-pass filtering a signal with a significant DC offset or when the result of filtering is very small compared to the original. If you clear the Same scale check box, the filtered data is scaled to fit in the window independently of the scaling of the original waveform.

Channel
The Channel field allows you to select a Waveform or a RealWave channel to filter.

Filter
The Filter field of the dialog box selects the filter to apply. There are up to 12 saved filters to choose from. You cannot edit this field when the details panel is hidden; click the Show details button to modify the filter and set the filter title.

Comment
The Comment field is for any purpose you wish; there is one comment per filter. You cannot edit this field when the details panel is hidden; click the Show details button to edit the comment.

The Filter field of the dialog box selects the filter to apply and the Channel field sets the waveform channel to filter.

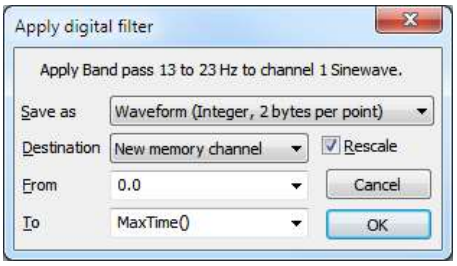
The Close button shuts the dialog and will ask if you want to save any changed filter and the Help button opens the on-line Help at the digital filtering topic.

Close
Click the Close button to shut the filtering dialog. If you have made a change to any of the filters, or loaded a new filter set, you are asked if you want to save the current set of filters as your standard filter set.

Load and Save
You can choose to save the current set of filters to a `.cfbx` (filter bank) file, or to load a new set of filters from a `.cfbx` or `.cfb` file. If you are working with FIR filters, this only saves or loads FIR filters. If you are working with IIR filters, this only saves or loads IIR filters. Loading a new filter set does not change your standard filter set, however, you will be asked if you want to change your standard set when you close the dialog.

Show details
The Show details button increases the dialog size to display a new area in which you can design and edit filters. Click this button again to hide the new dialog area. When you display the filter details, the Filter and Comment fields become editable. If you change a filter or create a new filter or load a new set of filters from a file, you will be prompted to save the filter bank when you close the digital filter dialog. The details are different for FIR and IIR filters.

Apply



The Apply button opens a new dialog in which you set where to write the result of the filter operation on the data channel. In the IIR filter dialog, this button is disabled if the filter is not valid. The channel comment of the new channel holds the source channel number and a description of the filtering operation.

Save As

You can save the results of filtering as 16-bit integer data (Waveform) using a scale and offset to convert to user units, or as 32-bit floating point data (RealWave). RealWave output gives you a more accurate result at the cost of using twice as much storage space.

Destination

You can write the result to a new memory channel, to an unused channel on disk, or to a memory channel that holds waveform or RealWave data with the same sampling rate as the channel you are filtering.

From and To

The From and To fields set the time range to process.

Rescale

If you check Rescale, the output data scale and offset are set to give the best possible representation of the waveform as 16-bit integers. If you save the output as a Waveform, this doubles the time required for filtering. If you Rescale when adding data to an existing memory channel, the scale and offset take into account both the added data and any remaining original data. If Rescale is unchecked, the scale and offset values from the source channel are copied. You would normally wish to rescale output to a RealWave channel as this improves accuracy if the channel is ever read as integer data.

OK

Click OK to start filtering. As applying a filter can be a lengthy process, a progress dialog appears with a Cancel button during the filtering operation.

Filter bank

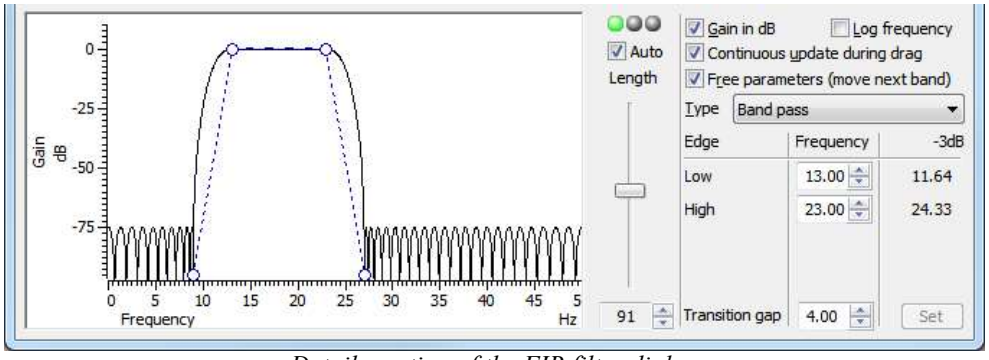
A digital filter definition is complex and it would be tedious to specify all the properties of a filter each time you wanted to apply one to data. To avoid this, Spike2 contains a filter bank of 12 FIR and 12 IIR filter definitions. This filter bank is saved to the XML file `filtbank.cfbx` when you close Spike2 and reloaded when you open it. If this file is not found, `filtbank.cfb` (the old format version) is read. If no filter bank file is found, Spike2 creates a new one with standard settings.

This default filter bank file is in the system folder for Spike2 user and application data, the same folder as returned by `FilePath$(-3)`. This is usually of the form: `C:\Users\username\AppData\Local\CED\Spike11\` where `username` is your user name. This means that each user of the system has their own default filter bank. You can load and save the filter bank files to other locations from the digital filter dialog. The file name used is `filtbank.cfbx`.

When you use the digital filter dialog, you specify which filter you want by the filter name. Script users identify the filter by its type (FIR or IIR) and an index number in the range 0 to 11. Script users also have access to two additional temporary filters with index number -1 (one for FIR and the other for IIR filters) that they can set and use for channel filtering operations without changing the standard filter set. You must create a temporary filter before you can apply it.

FIR filter details

The graph in the details area displays the ideal and actual frequency response of the filter. The ideal response appears as blue solid lines for each pass band linked by dotted lines that mark transition gaps between the bands. All transition gaps have the same width. The calculated frequency response is drawn as solid black lines and is greyed when the filter specification has changed and the response has not yet been calculated.



Details section of the FIR filter dialog

The mouse pointer changes to indicate the feature it is over: ⇹ for a pass band or stop band, ⇿ for a transition gap and ⇼ for a band corner. The small circles can be dragged sideways to change the slope of the band edges or you can edit the band edges as numbers in the Frequency panel on the right. You can also drag the bands and the transition gaps sideways.

Set

If you edit the numbers in the Frequency panel, the Set button is enabled so you can force a recalculation of the filter.

-3dB point

The filters produced by the program are not defined in terms of -3dB corner frequencies and n dB per octave, as is often the case for traditional analogue filters. The 3dB point column is present to help users who are more comfortable describing filter band edges in terms of the 3 dB point.

Gain in dB

The Gain in dB check box sets the y axis scale to dB if checked, linear if not checked. Using dB is usually the more convenient, except when working on a differentiator.

Log frequency

You can choose to display the frequency axis as the logarithm of the frequency, which gives you more resolution at low frequencies. However, this can make working with FIR filters more awkward as it removes the visual symmetry of the transition bands. In log mode, the frequency axis extends down to 0.001 of the data sample rate.

Continuous update

If you check the Continuous update box, the filter is updated while you drag the filter features around. If you have a slow computer and this feels ponderous you can clear the check box, in which case the filter is not recalculated until you stop changing features.

Free parameters

If you check the Free parameters box, dragged features are not limited by the next band and will push bands along horizontally. If you clear the box, the horizontal motion of a dragged feature is limited by the next moveable object.

Length, Auto and traffic lights

To the right of the frequency response display is a slider that controls the number of filter coefficients. In general, the more coefficients, the better the filter. However, the more coefficients, the longer it takes to

compute them and the longer to filter the data. If you check the Auto box, the program will adjust the number of coefficients for you to produce a useful filter (with around 70 dB cut between a pass band and a stop band). The "traffic light" display above the slider shows green if the filter is good, amber if the result is usable but not ideal, and red if the result is hopeless.

An FIR filter of length `n` uses the n/2 points before and after each input point to produce each output point. When there is no input data available before or after an input point, the filter uses a duplicate of the nearest point as an estimate of the data value. The `n`/2 output points next to any break in the input data should not be used for any critical purpose.

FIR Filter types

The Type field sets the arrangement of filter bands. If you need a filter that is not in this list you can generate it from the script language with the `FIRMake()` command. There are currently 12 different filter types:

All pass

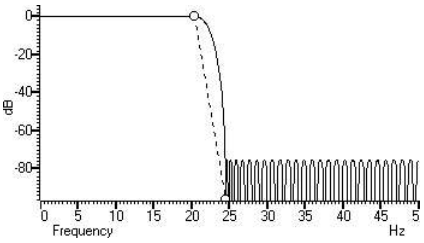
This has no effect on your signal. This filter type covers the case where you apply a low pass filter designed for a higher sampling rate to a waveform with a much lower sampling rate, so that the pass band extends beyond half the sampling frequency of the new file.

All stop

This removes any signal; the output is always zero. This filter type is provided to cover the case where you apply a high pass filter designed for a higher sampling rate to a waveform with a much lower sampling rate, so that the stop band extends beyond half the sampling frequency of the new file.

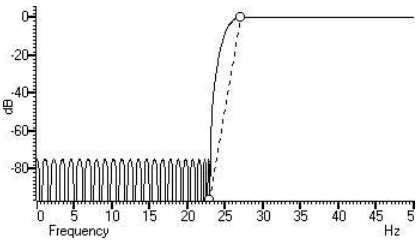
Low pass

This filter attempts to remove the high frequencies from the input signal. The filter has three bands: a pass band starting at 0 Hz, a stop band ending at half the sample rate and a transition band between them. The Frequency field holds one editable number, Low pass, the frequency of the upper edge of the pass band. The stop band starts at this frequency plus the value set by the Transition gap field. The number of coefficients needed to realise the filter is determined by the narrowest of the bands.



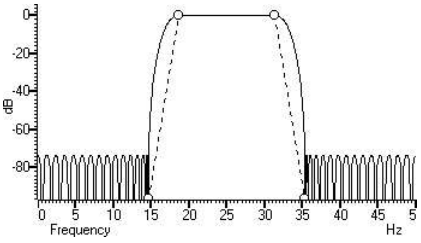
High pass

A high pass filter removes low frequencies from the input signal. The filter has three bands: a stop band starting at 0 Hz, a pass band ending at half the sample rate and a transition band between the stop and pass bands. The Frequency field holds one editable number, High pass, the frequency of the lower edge of the pass band. The stop band starts at this frequency less the value set by the Transition gap field. The narrowest band determines the number of coefficients needed to realise the filter.

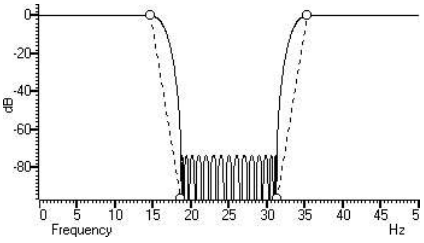


Band pass

A band pass filter passes a range of frequencies and removes frequencies above and below this range. The filter has five bands: two stop bands, two transition bands and one pass band. The Frequency field has two editable numbers, Low and High, which correspond to the two edges of the pass band. The stop band below runs up to Low-Transition gap, and the stop band above from High+Transition gap to one half the sampling rate.

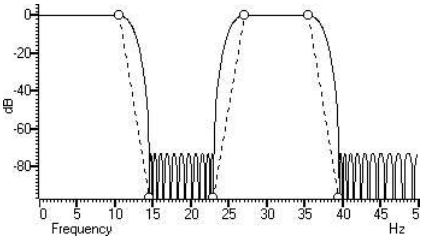


Band stop



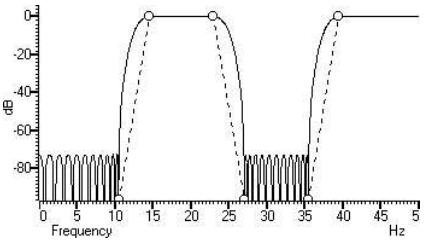
A band stop filter removes a range of frequencies. The filter has five bands: two pass bands, two transition bands and one stop band. The Frequency field has two editable numbers, High (the upper edge of the first pass band) and Low (the lower edge of the upper pass band). The stop band between the two pass bands runs from High+Transition gap up to Low-Transition gap. If you are trying to remove a single frequency you should consider using an IIR resonator notch filter. If you are trying to remove mains interference, see the CED web site and search for "hum remove".

One and a half low pass



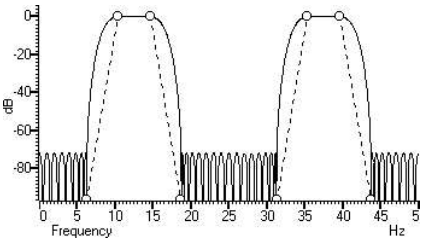
This filter has two pass bands, the first running from zero Hz and the second in the frequency space between the upper edge of the first pass band and one half the sampling rate. There are also two stop bands and three transition bands. The Frequency field has three editable numbers: Band 1 high, Band 2 low and Band 2 high. These numbers correspond to the edges of the pass bands. You are unlikely to need this filter in standard applications.

One and a half high pass



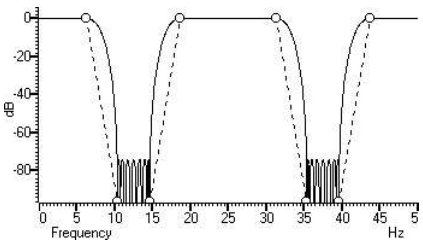
This filter has two pass bands. The second runs up to one half the sampling rate. The first band lies in the frequency space between 0 Hz and the lower edge of the second band. It also has two stop bands and three transition bands. The Frequency field has three editable numbers: Band 1 low, Band 1 high and Band 2 low. These numbers correspond to the edges of the pass bands. You are unlikely to need this filter in standard applications.

Two band pass



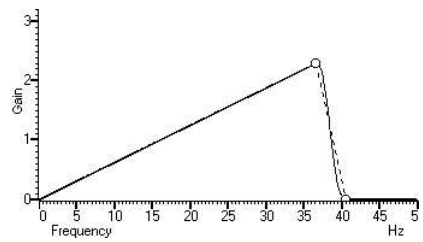
This filter passes two frequency ranges and rejects the remainder. It is implemented by three stop bands, two pass bands and four transition bands. Both 0 Hz and one half the sampling frequency are rejected. The Frequency field has 4 numeric fields: Band 1 low, Band 1 high, Band 2 low and Band 2 high. These fields correspond to the four edges of the two pass bands. You are unlikely to need this filter in standard applications.

Two band stop



This filter passes three frequency ranges and rejects the remainder. It is implemented by three stop bands, two pass bands and four transition bands. Both 0 Hz and one half the sampling rate are passed. The Frequency field has 4 numeric fields: Band 1 high, Band 2 low, Band 2 high and Band 3 low. These fields correspond to the four edges of the three bands. You are unlikely to need this filter in standard applications.

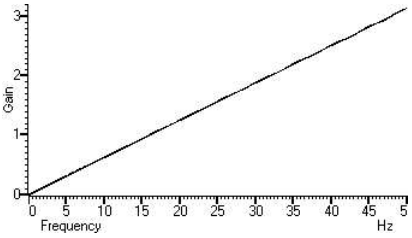
Low pass differentiator



This filter is a combination of a differentiator (that is the output is proportional to the rate of change of the input) and a low pass filter. The y axis scale in the image to the right is linear, rather than in dB (although you can display it in dB if you wish). There is one editable number in the Frequency field, Low pass, the end of the differential section of the filter. The filter is implemented as three bands: a differentiation band starting at 0, a transition band and a stop band running up to half the sampling rate. The number of coefficients needed to realise the filter depends on the narrowest band.

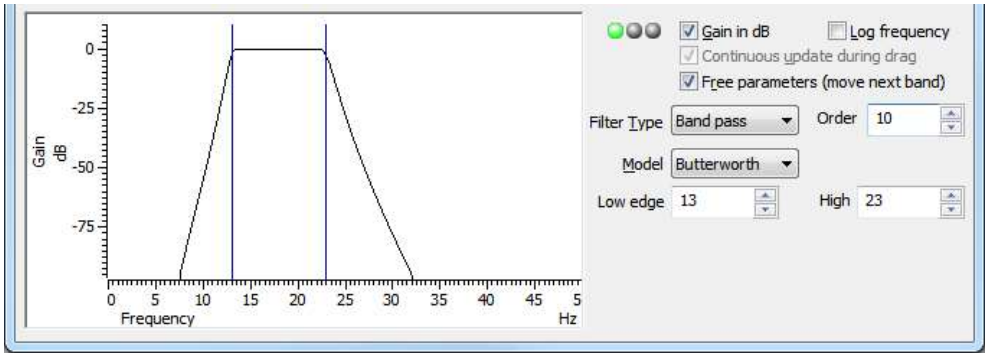
Differentiator

The output of the filter is proportional to the rate of change of the input. The y axis scale in the image to the right is linear, rather than in dB (although you can display it in dB if you wish). The Frequency field is empty as there is only one band and it extends from 0 Hz to half the sampling rate. The number of coefficients determines the ripple (the deviation from the ideal of a linear relationship between frequency and gain). This type of filter must have an even number of coefficients, and the output is displaced 0.5 of a sample from the input.



IIR filter details

We describe IIR filters in terms of a filter type (low pass, high pass, band pass or band stop), the analogue filter model that they are based on (Butterworth, for example), the corner frequencies and the filter order (which determines the steepness of the cut-off outside the desired pass bands). Filters based on Chebyshev designs also require a ripple specification and resonators require a Q factor.



Details section of the IIR filter dialog

The graph in the details area displays the corner frequencies and the frequency response of the filter. You can adjust the filter by clicking and dragging in this area or by editing the filter parameters as text. The mouse pointer changes to indicate the feature it is over: ↔ for a corner frequency and ↕ for an adjustable parameter (ripple or Q factor).

Gain in dB

The Gain in dB check box sets the y axis scale to dB if checked, linear if not checked. Using dB is usually the more convenient.

Log frequency

You can choose to display the frequency axis as the logarithm of the frequency, which gives you more resolution at low frequencies. The display extends to 0.0001 of the sample rate. If you need a corner frequency below this you should consider sampling the signal more slowly in the first place. Alternatively, low pass filter the signal and then use a channel process to down sample it before filtering.

Continuous update

This is always checked for IIR filters, and is greyed out.

Free parameters

If you check the Free parameters box, corner frequencies are not limited by the next corner, and will push them along. If you clear the box, corner frequencies cannot be moved past each other.

Traffic lights and messages

The traffic lights show green if the filter appears to be OK, amber if the filter may be unstable and red if the filter calculation failed, the filter is unstable or if one of the input parameters is illegal. There will usually be an explanatory message in the lower right hand corner of the dialog explaining the problem. In the amber state, the

filter may still be usable; you can look at the frequency response and the filtered data to see if the result is acceptable.

Filter Order

In an analogue filter, the filter order determines the sharpness of the filter, for example Butterworth and Bessel filters tend towards 6n dB per octave, where n is the filter order. In a digital filter, the order determines the number of filter coefficients. The higher the order, the sharper the filter cut-off and also, the more likely the filter is to be unstable due to problems in numerical precision. You can set filter orders of 1 to 10. You should always use the lowest order that meets your filtering criteria. Phase non-linearity gets worse as the filter order increases. The work required (time taken) to apply a filter is proportional to the filter order; it is doubled for band pass and band stop filters compared to low pass and high pass filters. The filter order is always 2 for a resonator.

Filter type

There are four filter types: Low pass, High pass, Band pass and Band stop. However, if you set the filter model to Resonator, there are only two types, being Band pass (this creates a resonator filter) and Band stop (this creates a notch filter). For all filter models except Chebyshev type 2 and Resonator, the frequencies given are the points at which the filter achieves a cut of 3 dB. For Chebyshev type 2 filters, the frequencies are the point at which the filter cut reaches the value set by the Ripple parameter. For Resonators, the frequency is the centre frequency of the resonator.

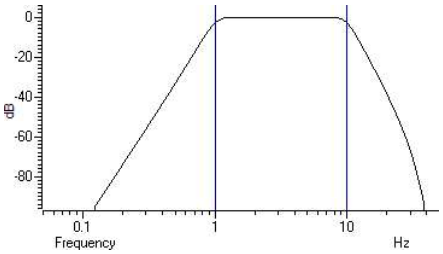
Low pass	Use this to remove high frequencies and pass low frequencies. This has a single corner frequency.
High pass	Use this to remove low frequencies and pass high frequencies. This has a single corner frequency.
Band pass	This passes a range of frequencies between the Low edge and the high edge. If the filter model is Resonator, then this has a single Centre frequency.
Band stop	This removes a range of frequencies between the Low edge and the high edge. If the filter model is Resonator, then this has a single Centre frequency.

Filter Model

The IIR filters we provide are digital implementation of standard analogue filter models. The following descriptions use fifth order 1 to 10 Hz band pass filters on 100 Hz data as examples (except for the Resonator filters). The five filter models are:

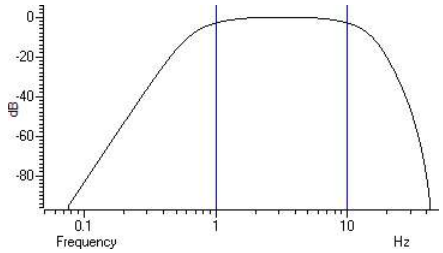
Butterworth

The Butterworth filter has a maximally flat pass band, but pays for it by not having the steepest possible transition between the pass band and the stop band. This is a good choice for a general-purpose IIR filter, but beware that the group delay can get quite bad near the corners, especially for high-order filters. Put another way, the shape of a signal can be significantly distorted. If preserving the signal shape is important for your application, you could consider using a Bessel filter or an FIR filter.

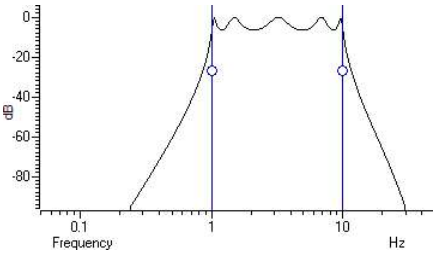


Bessel

An analogue Bessel filter has the property that the group delay is maximally flat, which means that it tends to preserve the shape of a signal passed through it. The price you pay for this property is that it leads to filters with a gentle cut-off. When realised as a digital filter, the constant group delay property is compromised; the higher the filter order, the worse the group delay. This is likely to be the second filter type to consider if you have rejected the use of a Butterworth filter.

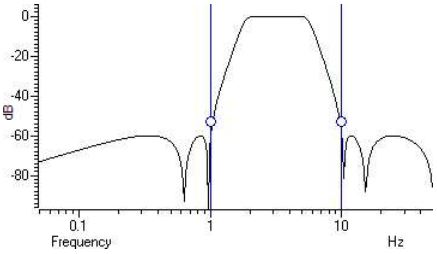


Chebyshev type 1



Filters of this type are based on Chebyshev polynomials and have the fastest transition between the pass band and the stop band for a given ripple in the pass band and no ripples in the stop band. You can adjust the pass band ripple (in dB) by dragging the small circles vertically or with the Ripple field to the right of the displayed frequency response. You will likely select this type of filter when having no signal in the stop band is more important to you than the flatness of the pass band and you do not care about preserving the signal shape.

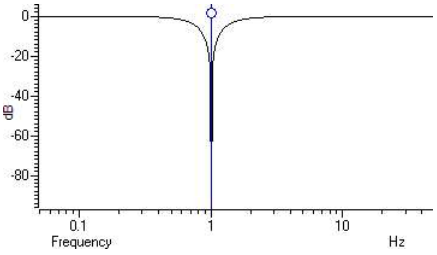
Chebyshev type 2



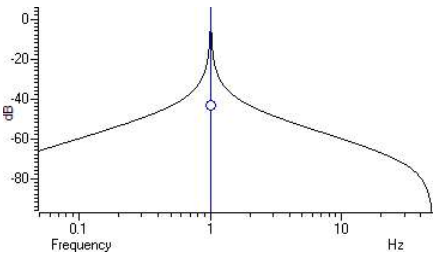
Filters of this type are defined by the start of the stop band and the stop band ripple (the minimum cut in the stop band). They have the fastest transition between the pass and stop bands given the stop band ripple (actually the minimum stop band cut) and no ripple in the pass band. Drag the small circles to adjust the ripple, or use the Ripple field to the right of the frequency response. These filters can be useful when you need a sharp cut off and a flat pass band and do not care about preserving the signal shape. Note that the filter corner frequencies and ripple are defined for the stop band, not the pass band.

Resonator

Resonators are defined by a centre frequency and a Q factor. The Q is the centre frequency divided by the width of the resonator at the -3 dB point. You can have a band stop or a band pass resonator. The Q is adjusted by dragging the small circle or with the Q field to the right of the displayed frequency response.



Band stop resonators are also called Notch filters. The higher the `Q`, the narrower the notch. Notch filters are sometimes used to remove mains hum, but if you do this you will likely need to set notches at the first few odd harmonics of the mains frequency and notch filters are not recommended if you care about the phase response of the signal in the vicinity of the notch. There are much better ways to remove hum than this, see for example the scripts for this purpose on our web site. The example has a very low Q (1.24) to make the filter response visible.



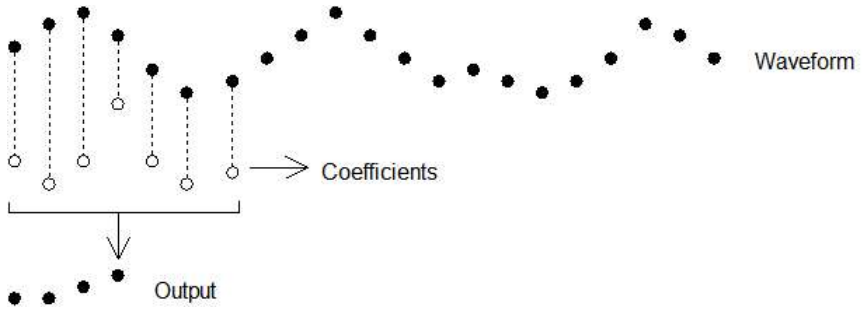
A band pass resonator is the inverse of a notch. Band pass resonators are sometimes used as alternatives to a narrow bandpass filter. The example has a Q of 100. The higher the Q set for a resonator, the longer it will take for the output to stabilise at the start of the filter output. Resonators, like Notches, are quite unpleasant filters in the way that they distort signal frequencies around central frequency. A resonator can be useful to detect the presence of a frequency component, but it is unlikely that the shape of the filtered waveform will have any meaning.

FIR filters and scripts

The `FIRMake()`, `FIRQuick()` and `FiltCalc()` script commands and the Analysis menu Digital filters… dialog generate FIR (Finite Impulse Response) filter coefficients suitable for a variety of filtering applications. The generated filters are optimal in the sense that they have the minimum ripple in each defined band. These filter coefficients are used to modify a sampled waveform, usually to remove unwanted frequency components. The algorithmic heart of the filter coefficient generation is based on the well-known FORTRAN program written by Jim McClellan of Rice University in 1973 that implements the *Remez exchange algorithm* to optimise the filter.

The theory of FIR filters is beyond the scope of this document. Readers who are interested in learning more about the subject should consult a suitable text book, for example *Theory and Application of Digital Signal Processing* by Rabiner and Gold published by Prentice-Hall, ISBN 0-13-914101.

FIR filtering algorithm



This diagram shows the general principle of the FIR filter. The hollow circles represent the filter coefficients, and the solid circles are the input and output waveforms. Each output point is generated by multiplying the waveform by the coefficients and summing the result. The coefficients are then moved one step to the right and the process repeats.

From this description, you can see that the filter coefficients (from right to left) are the *impulse response* of the filter. The impulse response is the output of a filter when the input signal is all zero except for one sample of unit amplitude. In the example above with 7 coefficients, there is no time shift caused by the filter. With an even number of coefficients, there is a time shift in the output of half a sample period.

Frequencies

The Analysis menu Digital filters… command deals with frequencies in Hz as this is comfortable for us to work with. However, if you calculate a FIR filter for one sampling rate, and apply the same coefficients to a waveform sampled at another rate, all the frequency properties of the filter are scaled by the relative sampling rates. That is, the frequency properties of an FIR filter are invariant when expressed as fractions of the sampling rate, not when expressed in Hz.

It is usually more convenient when dealing with real signals to describe filters in terms of Hz, but this means that each time a filter is applied to a waveform the sampling rate must be checked. If the rate is different from the rate for which the filter was last used, the coefficients must be recalculated. Unless you use the `FIRMake()` script command, Spike2 takes care of all the frequency scaling and recalculation for you. The remainder of this description is to help users of the `FIRMake()` script command, but the general principles apply to all the digital filtering commands in Spike2.

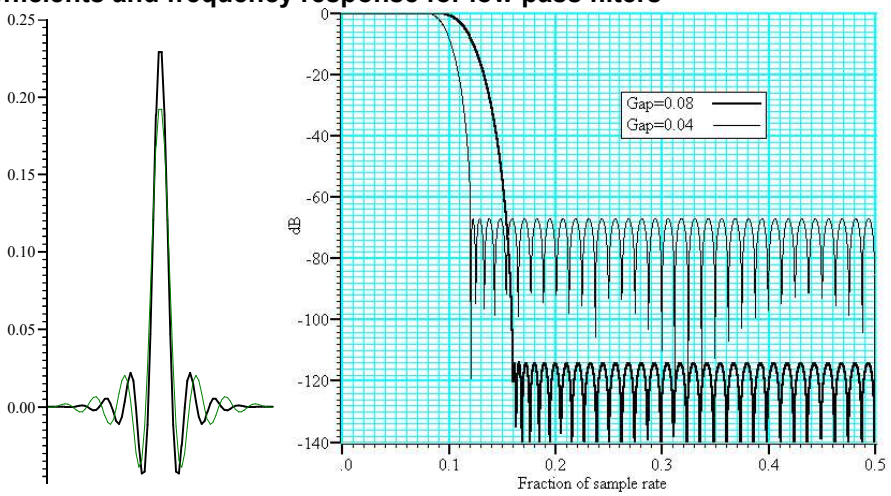
Users of the `FIRMake()` script command must specify frequencies in terms of fractions of the sample rate from 0 to 0.5. For example, if you were sampling at 10 kHz and you wanted to refer to a frequency of 500 Hz, you would call this 500/10000 or 0.05.

Example filter

The heavy lines in the next diagrams show the results obtained by `FIRMake()` when it designed a low pass filter with 80 coefficients with the specification that the frequency band from 0 to 0.08 should have no attenuation, and that the band from 0.16 to 0.5 should be removed. We can specify the relative weight to give to the ripple in each band. In this case, we said that it was 10 times more important that the *stop band* (0.16 to 0.5) should pass no signal than the *pass band* should be completely flat.

We have shown the coefficients as a waveform for interest as well as the frequency response of the filter. The shape shown below is typical for a band pass filter. One way of understanding the action of the FIR filter is to think of the output as the correlation of the waveform and the filter coefficients.

Coefficients and frequency response for low pass filters



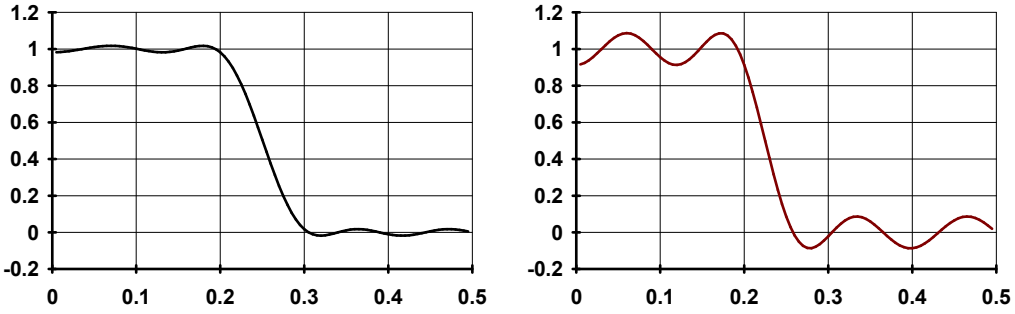
The frequency response is shown in dB, which is a logarithmic scale. A ratio *r* is represented by 20 log10(*r*) dB. A change of 20 dB is a factor of 10 in amplitude, 6 dB is approximately a factor of 2 in amplitude. The graph shows that a frequency in the stop band is attenuated by over 110 dB (a factor of 300,000 in amplitude with respect to the signal before it was filtered).

Because we didn't specify what happened between a frequency of 0.08 and 0.16 of the sampling rate, the optimisation pays no attention to this region. You might ask what happens if we make this transition gap smaller. The lighter line in the graph shows the result of halving the width of the gap by making the stop band run from 0.12 to 0.5. The filter is now much sharper. However, you don't get something for nothing. The attenuation in the stop band is reduced from 110 dB to around 70 dB. Although you cannot see it from the graph, the ripple in the pass band also increases by the same proportion (from 1 part in 30,000 to 1 part in 300).

We can restore the attenuation in the stop band by increasing the number of coefficients to around 120. However, there are limits to the number of coefficients it is worth having (apart from increasing the time it takes to calculate the filter and filter the data). Although the process used to calculate coefficients uses double precision floating point numbers, there are rounding errors and the larger the number of coefficients, the larger the numerical noise in the result.

Because the waveform channels are stored in 16-bit integers, there is no point designing filters that attenuate any more than 96 dB as this is a factor of 32768. Attenuations greater than this would reduce any input to less than 1 bit. If you are targeting data stored in real numbers this restriction may not apply.

It is important that you leave gaps between your bands. The smaller the gap, the larger the ripple in the bands.



This is illustrated by these two graphs. They show the linear frequency response of two low pass filters, both designed with 18 coefficients (we have used so few coefficients so the ripple is obvious). Both have a pass band of 0 to 0.2, but the first has a gap between the pass band and the stop band of 0.1 and the second has a gap of 0.05. We have also given equal weighting to both the pass and the stop bands, so you can see that the ripple around the desired value is the same for each band.

As you can see, halving the gap has made a considerable increase in the ripple in both the pass band and the stop band. In the first case, the ripple is 1.76%, in the second it is 8.7%. Halving the transition region width increased the ripple by a factor of 5.

In case you were worrying about the negative amplitudes in the graphs, a negative amplitude means that a sine-wave input at that frequency would be inverted by the filter. The graphs with dB axes consider only the magnitude of the signals, not the sign.

FIRMake filter types

`FIRMake()` can generate coefficients for four types of filter: Multiband, Differentiators, Hilbert transformer and a variation on multiband with 3 dB per octave frequency cut. The other routines can generate only Multiband filters and Differentiators.

Multiband filters

The filter required is defined in terms of frequency bands and the desired frequency response in each band (usually 1.0 or 0.0). Bands with a response of 1.0 are called *pass bands*, bands with a response of 0.0 are called *stop bands*. You can also set bands with intermediate responses, but this is unusual. The bands may not overlap, and there are gaps between the defined bands where the frequency response is undefined. You give a weighting to each band to specify how important it is that the band meets the specification. As a rule of thumb, you should make the weight in stop bands about ten times the weight in pass bands.

`FIRMake()` optimises the filter by making the ripple in each band times the weight for the band the same. The ripple is the maximum error between the desired and actual filter response in a band. The ripple is usually expressed in dB relative to the unfiltered signal. Thus the ripple in a stop band is the minimum attenuation found in that band. The ripple in a pass band is the variation of the frequency response from the desired response of unity. In some situations, for example audio filters, quite large ripples in the pass band are tolerable but the same ripple would be unacceptable in a stop band. For example, a ripple of -40 dB in a pass band (1%) is inaudible, but the same ripple in a stop band would allow easily audible signals to pass. By weighting bands you can increase the attenuation in one band at the expense of another to suit your application.

Differentiators

The output of a differentiator increases linearly with frequency and is zero at a frequency of 0. The differentiator is defined in terms of a frequency band and a slope. The frequency response at frequency *f* is *f* * *slope*. The slope is usually set so that the frequency response at the highest frequency is no more than 1.

The weight given to each frequency within a band is the weight for that band divided by the frequency. This gives a more accurate frequency response at low frequencies where the resultant amplitude will be the smallest.

Although you can define multiple bands for a differentiator, it is unusual to do so. Almost all differentiators define a single band that starts at 0. Occasionally a differentiator followed by a stop band is needed.

Hilbert transformers

A Hilbert transformer is a very specialised form of filter that causes a phase shift of -pi/2 in a band, often used to separate a signal from a carrier. The theory and use of this form of filter is way beyond the scope of this document. Unless you know that you need this filter type you can ignore it.

Multiband with 3dB octave cut

This is a variation on the multiband filter that can be used to filter white noise to produce band limited pink noise. The filter is identical to the band pass filter except that the attenuation increases by 3 dB per octave in the band (each doubling of frequency reduces the amplitude of the signal by a factor of the square root of 2). It is used in exactly the same way as the multiband filter.

Low pass filter example

A waveform is sampled at 1 kHz and we are interested only in frequencies below 100 Hz. We would like all frequencies above 150 Hz attenuated by at least 70 dB.

A low pass filter has two bands. The first band starts at 0 and ends at 100 Hz, the second band starts at 150 Hz and ends at half the sampling rate. Translated into fractions of the sampling rate, the two bands are 0-0.1 and 0.15 to 0.5. The first band has a gain of 1, the second band has a gain of 0. We will follow our own advice and give the stop band a weight of 10 and the pass band a weight of 1. We will try 40 coefficients to start with, so a possible script is:

```
var prm[5][2] :=   'Array for parameters
{
'   start  end gain weight
    {0.00, 0.1, 1.0,  1.0}, ' Band 0 (pass band, gain = 1)
    {0.15, 0.5, 0.0, 10.0}  ' Band 1 (stop band, gain = 0)
};
var coef[40];      'Array for the coefficients
FIRMake(1, prm[][], coef[]);
PrintLog("Pass Band ripple=%.1fdB Stop band attenuation=%.1f\n",
         prm[4][0], prm[4][1]);
```

If you run this, the log view output is:

```
Pass Band ripple=-28.8dB Stop band attenuation=-48.8
```

The attenuation in the stop band is only 48 dB, which is not enough. The ripple in the pass band is around 3% of the signal amplitude. We can increase the stop band attenuation in three ways: by increasing the number of coefficients, by giving the stop band more weight, or by making the gap larger between the bands.

We don't want to give the stop band more weight; this would increase the ripple in the pass band. We could probably reduce the width of the pass band a little as the attenuation of the signal tends to start slowly, but we will leave that adjustment to the end. The best way to improve the filter is to increase the number of coefficients. If we increase the size of `coef[]` to 80 coefficients and run again, the output now is:

```
Pass Band ripple=-58.7dB Stop band attenuation=-78.7
```

Formula for required number of coefficients

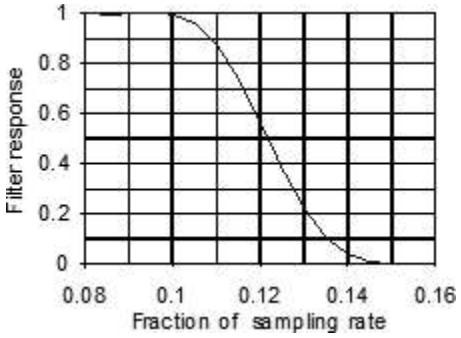
This is much closer to the filter we wanted. You might wonder if there is a formula that can predict the number of coefficients based on the filter specification. There is no exact relationship, but the following formula, worked out empirically by curve fitting, predicts the number of coefficients required to generate a filter with equal weighting in each of the bands as a function of the minimum transition gap width between the bands and is usually accurate to within a couple of coefficients. The formula can be applied when there are more than two bands, but becomes less accurate as the number of bands increase. The interactive filters always generate filters with equal sized transition gaps.

```
' dB     is the mean ripple/attenuation in dB of the bands
' deltaF is the width of the transition region between the bands
' return An estimate of the number of coefficients
Func NCoefMultiBand(dB, deltaF)
return (dB-23.9*deltaF-5.585)/(14.41*deltaF+0.0723);
end;
```

In our example we wanted at least 70 dB attenuation, and we weighted the stop band by a factor of 10 (20 dB). This causes a 10 dB improvement in the stop band at the expense of a 10 dB degradation of the pass band. Thus to achieve 70 dB in the stop band with the weighting, we need 60 dB without it. If we set these values in the formula (`dB` = 60, `deltaF`=0.05) , it predicts that 67.13 coefficients are needed. If we run our script with 67 coefficients, we get 70.9 dB attenuation, which is close enough!

A final finesse

If we look at the frequency response of our filter in the area between the pass band and the stop band, we see that the curve is quite gentle to start with. If you are used to using analogue filters, you will recall that the corner frequency for a low pass analogue filter is usually stated to be the frequency at which the filter response fell by 3 dB which is a factor of root 2 in amplitude (when the response falls to 0.707 of the unfiltered amplitude).



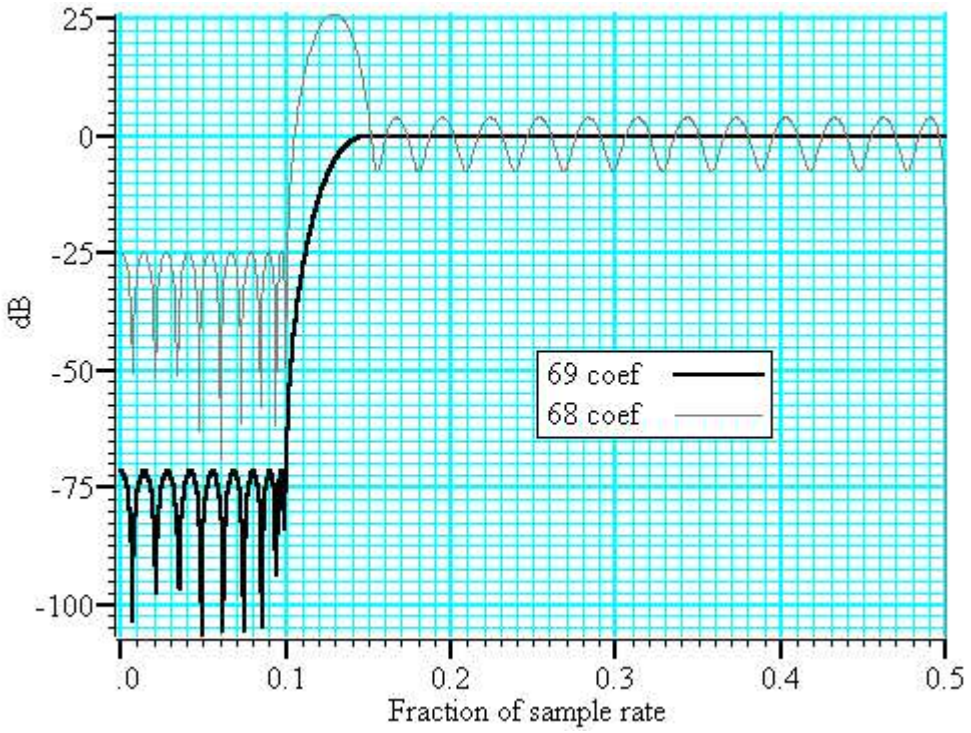
If we use the analogue filter definition of corner frequency, we see that we have produced a filter that passes from 0 to 0.115 of the sampling rate, and we wanted from 0 to 0.1, so we can move the corner frequency back. This will increase the attenuation in the stop band, and reduce the filter ripple, as it widens the gap between the pass band and the stop band. If we move it back to 0.085, the attenuation in the stop band increases to 84 dB. Alternatively, we could move both edges back, keeping the width of the gap constant. This leaves the stop band attenuation more or less unchanged, but means that the start of the stop band is moved lower in frequency.

High pass filter

A high pass filter is the same idea as a low pass except that the first frequency band is a stop band and the second band is a pass band. All the discussion for a low pass filter applies, with the addition that **there must be an odd number of coefficients**. If you try to use an even number your filter will be very poor indeed. The example below shows a script for a high pass filter with the same bands and tolerances as for the low pass filter. We have added a little more code to draw the frequency response in a result view.

```
var prm[5][2] := 'Array to describe the filter
{
'   start  end gain weight
    {0.00, 0.1, 0.0, 10.0},
    {0.15, 0.5, 1.0,  1.0}
};
var coef[69];
FIRMake(1, prm[][], coef[]); 'Generate the filter
const bins% := 1000;
var fr[bins%];
FIRResponse(fr[], coef[], 0); 'Generate frequency response

SetResult(bins%, 0.5/(bins%-1), 0, "Fr Resp", "Fr","dB");
ArrConst([], fr[]); 'Copy the response to result view
YRange(0, -100, 25);
DrawMode(1,2,0,1); 'Set Lines
WindowVisible(1);
```

Effect of odd and even coefficients



The graph shows the results of this high pass filter design with 69 coefficients, which gives a good result, and with 68 coefficients, which does not. In fact, if we had not given a factor of 10 weight (20 dB) to the stop band, the filter with 68 coefficients would not have achieved any cut in the stop band at all!

The reason for this unexpected result is that we have specified a non-zero response at the Nyquist frequency (half the sampling rate). If you imagine a sine wave with a frequency of half the sample rate, each cycle will contribute two samples. The samples will be 180° out of phase, so if one sample has amplitude *a*, the next will have amplitude *-a*, the next *a* and so on. The filter coefficients are mirror symmetrical about the centre point for a band pass filter, so with an even number of coefficients, the result when the input waveform is *a*, *-a*, *a*, *-a*... is

0. Another way of looking at this is to consider that a filter with an even number of coefficients produces half a sample delay. The output halfway between points that are alternately +*a* and -*a* must be 0.

You can use the formula given for the low pass filter to estimate the number of coefficients required, but you must round the result up to the next odd number.

General multiband filter

You can define up to 10 bands. However, it is unusual to need more than three. The most common cases with three bands are called band pass and band stop filters. In a band pass filter, you set a range of frequencies in which you want the signal passed unchanged, and set the frequency region below and above the band to pass zero. In a band stop filter you define a range to pass zero, and set the frequency ranges above and below to pass 1.

You must still allow transition bands between the defined bands, exactly as for the low and high pass filters, the only difference is that now you need two transition bands, not one. Also, if you want a non-zero response at the Nyquist frequency, you must have an odd number of coefficients.

For our example we will take the case of a signal sampled at 250 Hz. We want a filter that passes from 20 to 40 Hz (0.08 to 0.16) with transition regions of 7.5 Hz (0.03). If we say it is 10 times more important to have no signal in the stop band than ripple in the pass band, and we want 70 dB cut in the stop band we will get 50 dB ripple in the pass band (because a factor of 10 is 20 dB). To use the formula for the number of coefficients we need the mean attenuation/ripple in dB and the width of the transition region. The two stop bands have an attenuation of 70 dB and the pass band has a ripple of 50 dB, so the mean value is (70+50+70)/3 or 63.33 dB. We have two transition regions (both the same width). In the general case of transition regions of different sizes, use the smallest transition region in the formula. Plugging these values into the formula predicts 113 coefficients, however only 111 are needed to achieve 70 dB as demonstrated in the following example.

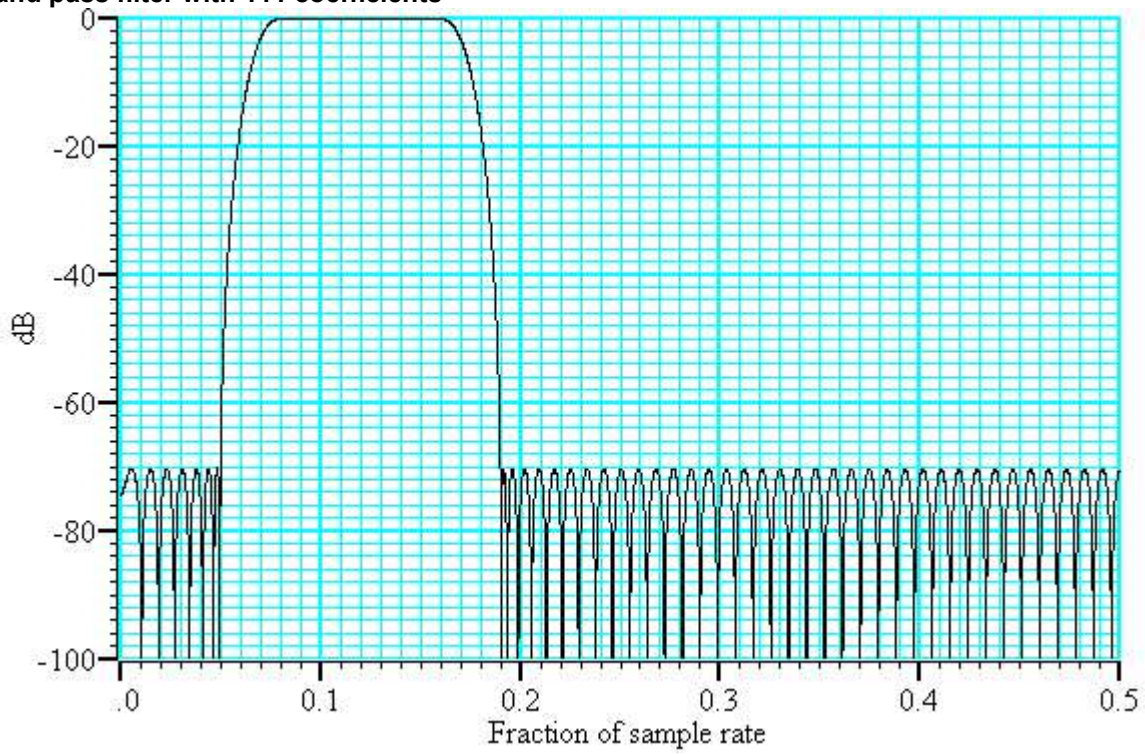
```
var prm[5][3] :=
{
'   start   end gain weight
    {0.00, 0.05, 0.0, 10.0},     ' Band 0 (stop, func = 0)
    {0.08, 0.16, 1.0,  1.0},     ' band 1 (pass, func = 1)
    {0.19, 0.50, 0.0, 10.0}      ' band 2 (stop, func = 0)
};
var coef[111];       ' 111 coefficients needed
FIRMake(1, prm[][], coef[]); 'Select a Multi-band filter
PrintLog("Ripple: %4.1f dB", prm[4][]); 'Print the ripple
```

The printed output is:

```
Ripple: -70.4,-50.4,-70.4 dB
```

Which translates as the attenuation in the two stop bands is just over 70 dB and a pass band ripple of -50 dB.

Band pass filter with 111 coefficients



Differentiators

A differentiator filter has a gain that increases linearly with frequency over the frequency band for which it is defined. There is also a phase change of 90 degrees (pi/2)]] between the input and the output.

Ideal differentiator with slope of 2.0

You define the differentiator by the number of coefficients, the frequency range of the band to differentiate and the slope. The example above has a slope of 2. Within each band (normally only 1 band is set) the program optimises the filter so that the amplitude of the ripple (error) is proportional to the response amplitude. A differentiator is normally defined to operate over a frequency band from zero up to some frequency `f`. If `f` is 0.5, or close to it, you must use an even number of coefficients, or the result is very poor. You can estimate the number of coefficients required with the following function:

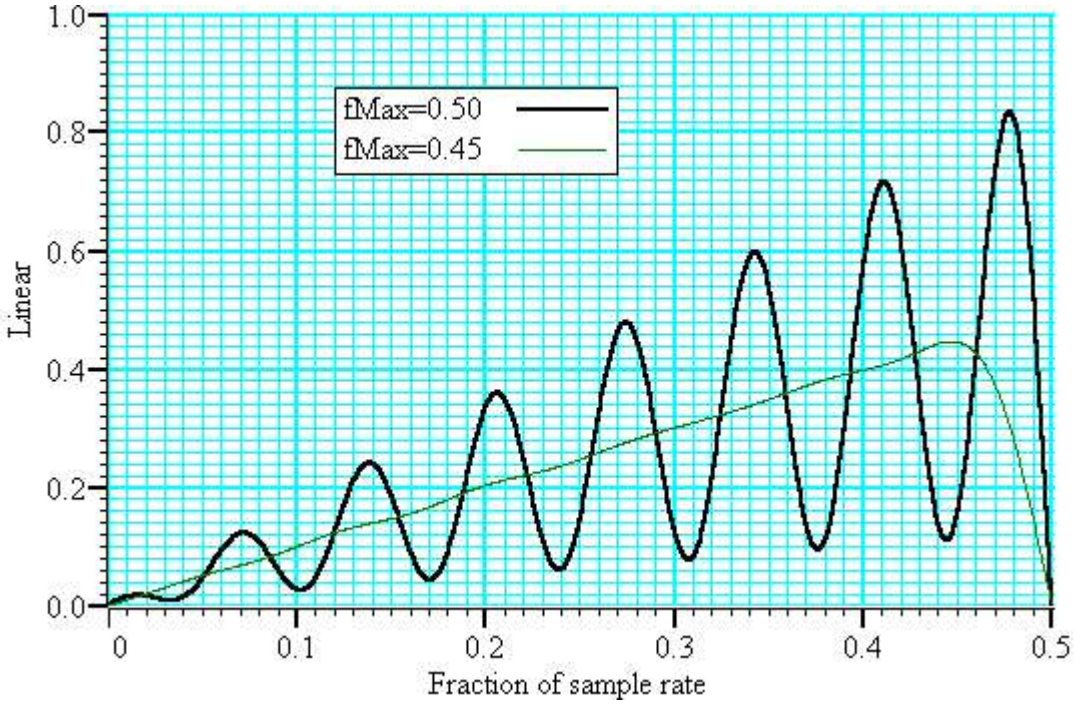
```
' dB    the proportional ripple expressed in dB
' f     the highest frequency set in the band
' even% Non-zero if you want an even number of coefficients
func NCoefDiff(dB, f, even%)
if (f<0) or (f>0.5) then return 0 endif;
f := 0.5-f;
var n%;
if (even%) then
   n% := (dB+43.837*f-35.547)/(0.22495+29.312*f);
   n% := (n%+1) band -2;   'next even number
else
   if f=0.0 then return 0 endif;
   n% := dB/(29.33*f);
   n% := n% bor 1;      'next odd number
endif;
return n%;
end
```

For an even number of coefficients this is unreliable when `f` is close to 0.5. For an odd number, no value of `n` works if `f` is close to 0.5.

These equations were obtained by curve fitting and should only be used as a guide. To make a differentiator that uses a small number of coefficients, use an even number of coefficients and don't try to span the entire frequency range. If you cannot tolerate the half point shift produced by using an even number of coefficients and must use an odd number, you must set a band that stops short of the 0.5 point. Remember, that by not specifying the remainder of the band you have no control over the effect of the filter in the unspecified region. However, for an odd number of points, the gain at the 0.5 point will be 0 whatever you specify for the frequency band.

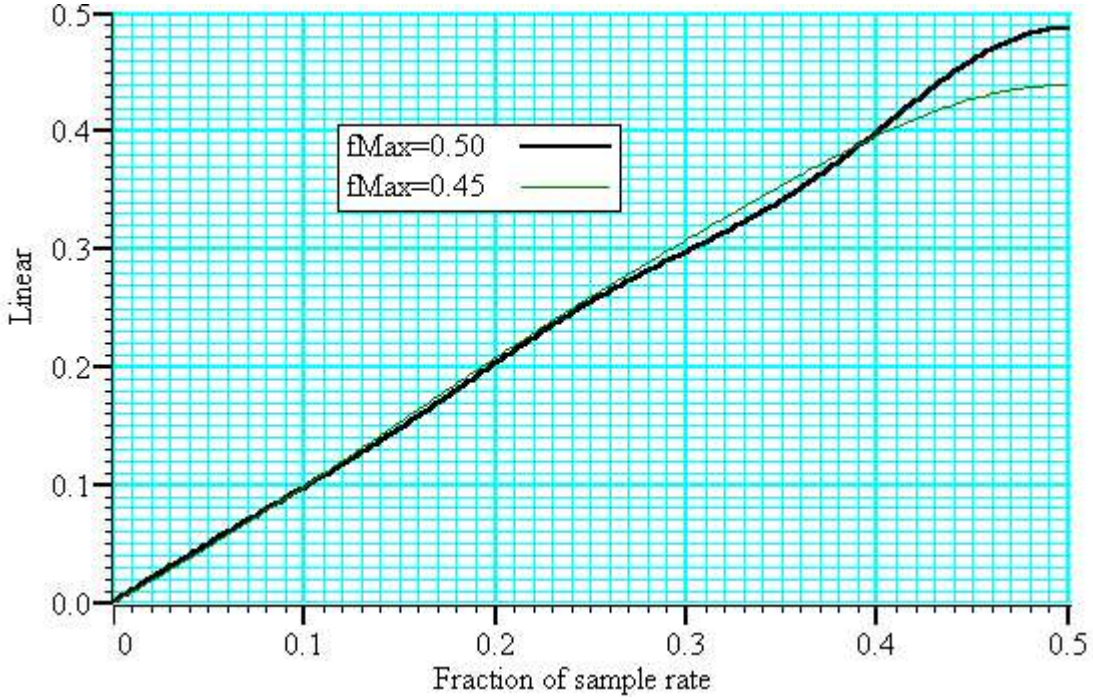
The graph below shows the effect of setting an odd number of coefficients when generating a differentiator that spans the full frequency range. The second curve shows the improvement when the maximum frequency is reduced to 0.45.

Differentiators with 31 coefficients



If you must span the full range, use an even number of coefficients. The graph below shows the improvement you get with an even number of coefficients. The ripple for the 0.45 case is about the same with 10 coefficients as for 31.

Differentiators with 10 coefficients



A script to generate a 10 coefficient differentiator spanning 0 to 0.45 is:

```
'            band start   end slope weight
var prm[4][1] := {{0.00, 0.45,  1.0, 1.0}};    ' 1 band for differentiator
var coef[10];       ' 10 coefficients only to demonstrate ripple
FIRMake(2, prm[][], coef[]); 'Type 2 for differentiator
```

Hilbert transformer

A Hilbert transformer phase shifts a band of frequencies from F_{low} to F_{high} by pi/2. The target magnitude response in the band is to leave the magnitude unchanged. F_{low} must be greater than 0 and for the minimum magnitude overshoot in the undefined regions, F_{high} should be 0.5-F_{low}. The magnitude response at 0 is 0, and if an odd number of coefficients is set, then the response at 0.5 is also 0. This means that if you want F_{high} to be 0.5 (or near to it), you must use an even number of coefficients.

There is a special case of the transformer where there is an odd number of coefficients and F_{high} = 0.5-F_{low}. In this case, every other coefficient is 0. This is of no help to Spike2, but users who write their own software can use this fact to halve the number of operations required to apply a filter.

It is extremely unlikely that a Hilbert transformer will be of any practical use in the context of Spike2, so we do not consider them further. You can find more information about this type of filter in *Theory and Application of Digital Signal Processing* by Rabiner and Gold.

19: Programmable signal conditioners

Programmable signal conditioners

The Spike2 signal conditioner control panel supports the CED 1902 *mk III* and *IV*, The Power1401 gain option, the Axon Instruments CyberAmp and the Digitimer D360, D360R and D440 signal conditioners. The D360 is not supported in 64-bit builds of Spike2.

Form Spike2 version [10.17] onwards you can run different conditioner types simultaneously. For Spike2 version [10.20] you can adjust more than one type of signal conditioner during sampling.

You can open the conditioner control panel from either the sampling configuration channel parameters dialog (when the channel type is waveform or WaveMark) or from the Sample menu. The 1902 and CyberAmp signal conditioners are controlled through a serial port which is set in the Edit menu Preferences… option. Signal Conditioners can be programmed from the script language using the `CondXxxx()` family of commands.

What a signal conditioner does

Many input signals from experimental equipment are too small, or are masked by high and or low frequency noise, or are not voltages and cannot be connected directly to the 1401. A signal conditioner takes an input signal and amplifies, shifts and/or filters it so that the data acquisition unit can sample it effectively.

Signal conditioners may also have specialist functions, for example converting transducer inputs into a useful signal, or providing mains notch filters. The CED 1902 has options for isolated inputs and specialised front ends include ECG with lead selection, magnetic stimulation artefact clamps and EMG rectification and filtering. The only option for the Power1401 is Gain. You should consult the documentation supplied with your signal conditioner to determine the full range of capabilities.

Communication preferences

The basic communication parameters are set in the Edit menu Preferences dialog. Most supported programmable signal conditioners are controlled through communication (serial) ports. No port is used if the conditioner support is not loaded or None is selected in the preferences or for the Power1401 gain option. Check the *Dump errors…* box to write diagnostic messages to the Log window.

Control panel

The control panel is in two halves. The left-hand half holds the controls that change the conditioner settings, the right-hand half displays data from the conditioner. The right-hand half is omitted if Spike2 is sampling data, or if the 1401 is not available for any other reason.

If the right-hand half is present, the Volts check box causes the data to be displayed in Volts at the conditioner input in place of user units as defined by the Channel parameters dialog. The number at the bottom right is the mean level of the signal in the area marked above the number.

Signal conditioners differ in their capabilities. Not all the controls listed below may appear for all conditioners. The controls are:

Port

This is the physical 1401 port that the conditioner is attached to. If you open the conditioner dialog from the channel parameters dialog you cannot change the port.

Input

If your signal conditioner has a choice of input options, you can select the input to use with this field. The choice of input may also affect the ranges of the other options.

Gain

This field sets the gain for the signal selected by the Input field. Spike2 tracks changes of gain (and offset) and changes the channel gain and offset in the sampling configuration to preserve the y axis scale. You should

adjust the gain so that the maximum input signal does not exceed the limits of the data displayed on the right of the control panel. When Spike2 is sampling, the gain and offset are fixed once you have written data to disk.

This is the only editable field for the Power1401 ADC Gain option.

Offset

Some signals are biased away from zero and must be offset back to zero before they can be amplified. If you are not interested in the mean level of your signal, only in the fluctuations, you may find it much simpler to AC couple (1902) or high-pass filter (CyberAmp) the signal and leave the offset at zero. If Spike2 is sampling you cannot change the offset once data has been written to disk.

Low pass filter

A low-pass filter reduces high-frequency content in your signal. Filters are usually specified in terms of a corner frequency, which is the frequency at which they attenuate the power in the signal by a factor of two and a slope, which is how much they increase the attenuation for each doubling of frequency. Sampling theory tells us that you must sample a signal at a rate that is as least twice the highest frequency component present in the data. If you do not, the result may appear to contain signals at unexpected frequencies due to an effect called aliasing. As the highest frequency present will be above the corner frequency you should sample a channel at several times the filter corner frequency (probably between 3 and 10 times depending on the signal and the application).

You can choose a range of filter corner frequencies, or you can choose to have the data unfiltered (for use when the signal is already filtered due to the source).

High pass filter

A high-pass filter reduces low-frequency components of the input signal. The high-pass filters area is specified in the same way as low-pass filters in terms of a corner frequency and a slope, except that the slope is the attenuation increase for each halving of frequency. If you set a high-pass filter, a change in the mean level of the signal will cause a temporary change in the output, but the output will return to zero again after a time that depends on the corner frequency of the filter. The lower the corner frequency, the longer it takes for mean level change to decay to zero.

Notch filter

A notch filter is designed to remove a single frequency, usually set to the local mains power supply (50 Hz or 60 Hz, depending on country).

The remaining options are for the 1902 only:

AC couple

This is present for the 1902 only, and can be thought of as a high-pass filter with a corner frequency of 0.16 Hz. However, it differs from the high-pass filters as it is applied to the signal at the input; the high-pass filters in the 1902 are applied at the output.

Trigger source

If a conditioner has a choice of trigger signals, this field lets you select one. The 1902 provides two conditioned trigger inputs, and one output. This control selects which of the inputs is connected to the output.

Filter type

If a conditioner has a choice of filter types, this is where you select the one to use. A 1902 *mk IV* with digital filters has extra fields that allow you to set the low-pass and high-pass filter types. You can choose Butterworth or Bessel characteristics and 2 pole or 3 pole filters.

Reset Calibration

This button is intended for use in the initial set up stage, before you start sampling. It adjusts the sampling configuration scale and offset so that the signal is displayed in units of Volts, millivolts or microvolts, as appropriate, depending on the gain that you have selected.

Reload 1902

This only exists for the CED 1902. The 1902 is controlled by a serial interface and to set the complete state of it would takes a noticeable time. To avoid this, we keep a record of the state that has been commanded and only transmit changes. If you click this button, we transmit all the commands to set every feature of the 1902 to match the control panel. This is provided so that you can recover a 1902 that has been powered down by accident.

Setting the channel gain and offset

If you change the gain or offset in the control panel, Spike2 will adjust the channel gain and offset in the sampling configuration to compensate so as to keep the y axis showing the same units. This means that if you change the gain, the signals will still appear in the same units in the file. However, the first time you calibrate the channel you must tell the system how to scale the signal into y axis units.

The easy way

If you want to set your channels to display in Volts, millivolts or microvolts, there is an automatic and easy way to do it:

1. Open the Sampling Configuration dialog.
2. Select the waveform or WaveMark channel and open the channel parameters dialog.
3. Click the Conditioner button to open the conditioner control panel for your signal conditioner.
4. Change the controls so that the largest signal you want to record will fit in the display window without clipping.
5. Click the `Reset Calib` button in the control panel, then OK.

You will find that the settings in the channel parameters dialog have been adjusted so that when you record, the channel will display in the correct units.

The general case

If you must calibrate your input in other units then Volts, millivolts or microvolts, you have to do it manually. For example, to set up the y axis scales in microvolts by hand you could do the following:

1. Open the Sampling Configuration dialog.
2. Select the waveform or WaveMark channel and open the channel parameters dialog.
3. Set the Units field of the Channel parameters to µV.
4. Set the Input in Volts x field to 1000000.
5. Click the Conditioner button to open the conditioner control panel.
6. Adjust the gain to give a reasonable signal.
7. Close the signal conditioner control panel.

You only need do steps 3 and 4 once. Any subsequent change to the conditioner gain (but not while sampling, see below) will adjust the channel gain to leave the units in microvolts.

For the more general case where you have a transducer that measures some physical quantity (Newtons, for example) and it has an output of 152.5 Newtons per mV. If you wanted the Y axis scaled in Newtons, you would replace steps 3 and 4 above with:

3. Set the Units field of the Channel parameters to N.
4. Set the Input in Volts x field to 0.1525.

To work this out you must express the transducer calibration in terms of Units per Volt (in this case Newtons per Volt).

If you have set an offset in the conditioner, and you want to preserve the mean signal level, you should null it out by changing the offset in the Channel parameters dialog.

Changing gain or offset while sampling

Ideally, you will set up the conditioner settings before you start to sample, and not change them during sampling. However, if you have set the initial gain too high or too low, or you need to null out an offset, you may have to change the settings during.

Waveform and WaveMark channels
Spike2 Waveform and WaveMark channels are stored as 16-bit integer values that are converted into real user units by a scale factor and an offset. The scale factor and offset are set for the channel; there is no concept of changing them during sampling. We allow you to use the conditioner control panel during sampling, but if you change the scale and/or offset, this does not automatically change the channel scale and offset as this would invalidate all previously recorded data.

RealWave channels
You can sample waveform data as a RealWave channels. This has the disadvantage that the channels occupy double the space on disk (as each data point is stored in a 32-bit floating point value rather than as a 16-bit integer value). It has the advantage that you can change the scale and offset value of the channel without invalidating all previously sampled data. If you change the conditioner settings for a RealWave channel, we apply this to the input data as soon as possible; there will be a glitch in the signal when the gain changes, then the signal should continue with the correct real values.

Recording conditioner changes
If you have a TextMark channel enabled, each time you make a change *from the conditioner control panel*, a new marker will be added to the channel with a marker code of 02 and the associated text will be something like:

```
ch 2 gain 100, offs 0.001
```

You could use this information offline to recalibrate Waveform channels. Changes made using the Signal Conditioner script commands do not cause writes to the TextMark channel; if you want this, program it from the script with the `SampleText()` command.

If you do not have a TextMark channel enabled during sampling, the keyboard marker channel gets an event with code 02, so you know that a change was made.

Conditioner connections

Installation of conditioner support
Spike2 can control multiple signal conditioners; before version [10.17] they all had to be of the same type. When you install Spike2 we install software for all supported conditioners. Some conditioners also require their own support software installing. You should read the information supplied with each signal conditioner before using it in Spike2. The information below only relates to settings needed for use with Spike2, not general use of the devices.

Conditioner settings storage
Each conditioner has a unit number starting at 0 and each unit conditions the same number of data channels (1 for 1902, 8 or 2 for the CyberAmp, 8 for the Digitimer 360, see the description of each device). For both the 1902 and the CyberAmp, if the conditioner unit number is `u` and there are `n` channels per conditioner, it conditions 1401 ADC channels `u*n` to `u*(n+1)-1`. It is up to you to make the correct connections. The Digitimer devices handle multiple devices in their driver software and amalgamate them so that they appear as a continuous block of devices, starting at device 0.

Basic communication and connection information is stored in the registry in the key `HKEY_CURRENT_USER/Software/CED/CEDCond`. For each conditioner type that needs to store information there is an additional sub-key, currently `1902`, `CyberAmp`, `D360`, `D440`, `D360R`, `Power1401`. This sub-key holds integer variables:

`Port` If needed, the communication port number in the range 1 to 19. The default value is 1.

`Dump` Optional. Set to 1 to write diagnostic messages to Log window.

`First` Optional, default 0. This sets the first 1902 or CyberAmp unit number to search for. Use `First` to save time waiting for missing units to time out.

`Last` Optional, default 0. Searches stop when there is no response from a unit number greater than `Last`. Use `Last` to skip over missing units.

`Start` Available from [10.17] onwards. This sets the first ADC port that the block of devices controls.

All these values can be set interactively in the Conditioner Tab of the Edit menu Preferences option.

Ancient history

Before Spike2 [7.09], conditioner settings were saved in the file CEDCOND.INI in the system folder. We no longer support this, however the following information may be useful if you upgrade from an old version of Spike2. The file format was:

```
[General]
Port=COM1
Dump=1
First=1
Last=3
```

Supported conditioners

CED 1902
CyberAmp
Digitimer D360
Digitimer D440
Digitimer D360R
Power1401 with programmable gain

CED 1902

The CED 1902s unit numbers are set by an internal switch pack; multiple units usually have the channel number as a label on the front panel. Multiple units must have different unit numbers. It is up to you to connect the 1902 outputs to the correct 1401 ADC port. The mapping of 1902 units to ports is done with the Start, First and Last registry settings, controlled by the Edit menu Preferences command in the Conditioner tab.

If Start is set to 0, which is usually the case unless you have multiple signal conditioner types, each 1902 output should be connected to the 1401 ADC input with the same number as the 1902 unit. However, you can use the Start field to offset the ADC ports.

Usually the 1902s unit numbers start at 0 and are contiguous, in which case you do not need to set First and Last. As an example of a more complicated situation, let us suppose you have unit numbers 4, 5, 6, 7, 12, 13, 14 and 15. In this case, set First to 4 and Last to 15 for ADC ports 4-7 and 12-15. If you enable logging of information you will see that the system scans for units 4-16 and times out on units 5-11 and 16.

If the Start field were set to 3, this same arrangement would map the available conditioner channels to ADC ports 7-10 and 15-18.

CyberAmp

The CyberAmp has a device address (unit number) set by a rotary switch on the rear panel; multiple units must have different addresses. There are two types of CyberAmp: 8-channel and 2-channel. If you have multiple units, they must all have the same number of channels, or all the 8-channel units must have lower device addresses than all the 2-channel units and the channel mapping behaves as if all units had 8 channels. The mapping of CyberAmp units and channels to ports is done with the Start, First and Last registry settings, controlled by the Edit menu Preferences command in the Conditioner tab.

Users with a single signal conditioner type will usually leave the Start value set to 0 (which keeps the channel mapping backwards compatible with older versions of Spike). Normally you will have one 8-channel CyberAmp, and you will give it address 0 to support ADC ports 0 to 7. In this case you do not need to set First and Last. If you want to connect it to channels 8-15, you would set both First and the device address to 1 and then you could connect it to channels 8-15.

Here is a more complex example with three CyberAmp devices, two with 8-channels and one with 2-channels. The table shows some possible configurations (assuming you have 32 ADC inputs):

CEDCOND.INI		8-channel unit 1		8-channel unit 2		2-channel unit 3	
First	Last	Switch	ADC	Switch	ADC	Switch	ADC
0	2	0	0-7	1	8-15	2	16-17
1	3	1	8-15	2	16-23	3	24-25

2	3	2	16-23	3	24-31	-	-

Using Start to change the mapping

If you have multiple signal conditioner types you can shift the range of channels upwards by using the `Start` registry value. In this case, all the ports listed in the ADC column of the above table have the `Start` value added to them.

Problem connecting to CyberAmp

Assuming that the correct communications port is set in the Edit menu preferences, and that you installed Spike2 and selected CyberAmp support, the most common reason for lack of communication is that the rotary switch on the rear of the CyberAmp is not set to 0.

We have also seen problems on computers where the COM port has been set up in the computer BIOS as an infra-red communication device.

CyberAmp with input probes 401, 402, 405 and 414 has no output

These probes are described as differential in their specification, so you have selected the differential input setting. However these probes convert the signal to single ended. If you select the differential input, the same signal is applied to both CyberAmp differential inputs, and the result is to test the common mode rejection of the system. Select single ended or inverted inputs with these probes.

Digitimer D360

Currently, this is only supported for x86 builds of Spike2. You cannot use it with an x64 build. If Digitimer release 64-bit support we will add this to Spike2.

Install the Digitimer-supplied D360 support first and verify that it is working by running the Digitimer D360 Client application. The Digitimer D360 Client software must be at least version 4.7.0.0; upgrade to the latest version if it is older. The D360 connects to the computer with a serial line (COM port), however, our code has no control over the port; this is configured by the Digitimer code. The only registry option is `Dump`.

If you have problems getting the D360 to connect, run the Digitimer D360 client software and reset the D360 from the client software. You should then be able to connect and you can control the D360 from either the client software or from the Spike2 Conditioner Control panel.

Multiple D360 amplifiers can be used; each has 8 channels numbered 1 to 8. Channels 1 to 8 of the first D360 are assumed to be connected to ADC inputs number 0 to 7, the second D360 drives ADC ports 8 to 15 and so on. The amplifier numbers are determined by the order in which they are daisy-chained together, they should be connected in serial-number order with the lowest-numbered amplifier connected directly to the PC, the next lowest-numbered amplifier connected to the first amplifier and so on.

Digitimer D440

Install the Digitimer-supplied D440 support software first and verify that it is working by running the Digitimer control panel. Spike2 should then be able to locate and use the device. The Digitimer software does not (yet) support multiple units.

Digitimer D360R

Install the Digitimer-supplied D360R support software first and verify that it is working by running the Digitimer control panel. Spike2 should then be able to locate and use the device. If you have multiple units, they are assigned to channels in unit serial number order.

Note that the D360R can be controlled by the Digitimer control panel as well as by the Spike2 Conditioner control panel. We monitor the state of the D360R and track any changes made by the Digitimer control panel. If you use both simultaneously, as soon as you make any changes in the Spike2 control panel we stop accepting changes from the Digitimer controls until you apply the changes.

Power1401 with programmable gain

Spike2 will automatically detect the channels. The only registry option is `Dump`.

20: Test and utility programs

Test and utility programs

Spike2 includes programs that recover data from damaged 64-bit and 32-bit data files (often due to loss of power during recording) and a program to test that a 1401 is operating correctly. You can launch these programs from the File menu Utility Programs command.

The S64Fix data recovery utility

The S64Fix program scans 64-bit (`.smrx`) data files to recover data. Use SonFix for 32-bit (`.smr`) files. S64Fix can repair three types of damaged files:

- Files where sampling was interrupted and the file was not closed properly. In this case the file headers will hold sensible information, but it will likely relate to some earlier state of the sampling. Spike2 can detect this situation when you attempt to open such a file and will refuse to let you continue as any attempt to modify the file could overwrite valid data.
- Files that have been damaged. If the damage is to the start of the file, the file may well refuse to open.
- Files written before Spike2 version 8.02 where a channel has been deleted and reused. These files open without any problem, but data in the affected channel(s) can appear to be missing and may display differently, depending on where you start.

64-bit son files are designed to be relatively easy to recover. As long as the file header can be read, we can read through a file block by block and save data for blocks that appear undamaged and ignore blocks that do not make any sense to us. If a file header is damaged, as long as we can read the header from a similar, undamaged file, we can use that to recover the data from the damaged file.

Recovering from interrupted sampling

Run S64Fix Open the program to display an empty window

Browse... Select the damaged file. The Progress section of the dialog will display:

Open file: MORE_DATA - extra data on file end; maybe closed incorrectly
The son64 library can detect that the file is larger than the size stored in the file header. If there is a different message, for example WRONG_FILE, go to Recovering a damaged file.

New/Fix File You can now choose a name for a new file; do NOT use the same name as the file you are recovering. We scan the original file and write extracted data to the new file. If all goes well, the Progress bar will appear, showing progress as we read the source file. The Progress area will display any problems discovered and after reading through the entire file, a list of channels and the number of recovered blocks in each channel will appear. If all is well you can use the Restart button to recover a different file or Close to end the recovery session.

There can be an appreciable delay after fixing a file. This is because when the new file is closed, we flush all buffered data in the operating system to the disk file, and this can be many hundreds of megabytes of data (depending on the free memory in your system).

If the Progress section displays:

Open file: WRONG_FILE - attempt to open wrong file type
this means that the file you selected was not recognised. If you know that it really is the correct type then the file header is damaged. In this case you should follow the instructions for Recovering a damaged file.

Recovering a damaged file

Run S64Fix Open the S64Fix program to display an empty window

Browse... Select the damaged file. The Progress section of the dialog may display:

Open file: WRONG_FILE - attempt to open wrong file type

If it does and the file really is a data file of the correct type, the file header has been damaged. In this case click Browse... again.

Browse... (with damaged file) — If the damaged file does not open you can read the file header information from another file that holds the same set of channels. Click Browse... and open a suitable file; it must have the same channel numbers and channel types as the file you wish to rescue. If the file opens without error, you will see the file in the Replacement header file section of the dialog.

New/Fix File — Click here to set the name of the recovered file and then process the damaged file and write recovered blocks to the new file.

We can cope with damage where areas of the file have been overwritten or in cases where file copying has corrupted the file data (but the file is still readable).

Files with bad sectors (disk cannot be read)

If the file has bad sectors that cannot be read you will need to recover these first (even if the bad areas are filled with random data). We use the alignment of the data in the file as part of the data recovery, so if your recovered file is split into pieces, we will likely be able to recover data from the first piece, but not from subsequent parts unless you are lucky with the size of the gaps. The data blocks affected by the corruption will be lost.

If you have priceless data that has got chopped up into sections due to bad sectors we may still be able to recover some of it, but this can be time consuming and we would likely charge you for our time.

Recovering reuse of a deleted channel before Spike2 8.02

Run S64Fix — Open the S64Fix program to display an empty window

Browse... — Select the damaged file. The file will appear to open normally.

New File — Click here to set the name of the recovered file and then process the damaged file and write recovered blocks to the new file.

The problem with these files is that the internal tree that allows us to locate channel data efficiently does not have the correct start times of each data block. This causes the lookup process that locates the data to fail. The usual result is that the file appears to have missing data. The repair process ignores the internal tree, and links together the data blocks that it finds in the file. It is possible to get data returned beyond the end of the actual recorded data; if this should happen you can ignore it, or fix the problem by exporting the result to a new file, omitting the unwanted data at the end of the channel.

How fixing works

A 64-bit son data file is composed of data blocks of a fixed size. These are large blocks (currently 64 kB in size) that hold either header information or data for one channel. There are also smaller blocks (typically 8 kB in size) that hold channel indexing information. The block sizes are chosen to balance the requirements of high performance and reasonable memory usage. The block sizes are coded in the file header; it is possible to build the software for different block sizes for different applications, though we have no plans to do this at the moment.

The small blocks are grouped together to use a space the same size as a large block. This means we can scan a data file sequentially by reading through it in steps equal to the large block size. Each block starts with information that identifies the block type and the data channel to which it belongs (header blocks have a channel number of 65535). As the index blocks are used to locate data very quickly when reading, we can ignore them when scanning for data. This means that if the file index system gets damaged, which would stop Spike2 reading through a channel, we can still recover any undamaged data from the file.

For the scanning process to work correctly, we need a file header that as a minimum, identifies the channel numbers and channel types in the file. Given this information, we read though the file, block by block. For each block that has a recognisable header we check that the data in the block is plausible, and if this is the case, we write the data to the output file.

At the time of writing we have little experience of damaged data files, other than files that we have deliberately damaged by switching off the system power while sampling (do not try this yourselves as you run the risk of trashing your system).

If other systematic sources of error show up that require different fixing strategies we will apply them in the future.

What we cannot recover

There are several types of data that are gone forever...

Data that was never written to the disk

When we sample data, there is a data buffer of typically 8 MB that holds the most recently sampled data on each channel. This allows you to set up triggered sampling scenarios where data is usually not saved to disk, but which is marked for saving based on some criterion, often an external trigger or real-time data analysis. The buffering used for this is "circular", that is new data overwrites the oldest buffered data. We also have as list of times associated with each buffer for which data is to be saved.

When data is ejected from the circular buffer and is marked as wanted it is accumulated in a channel "write buffer". Each time a write buffer becomes full, it is written to disk (or more accurately, written to the operating system filing system). This means that during sampling, each channel has quite a bit of buffered data in memory. There is a write buffer of up to one whole block of data, holding data that is wanted and there is also a circular buffer holding data that may or may not be wanted.

The problem with this is that should the system power fail or your computer crash, you will lose all data in the circular buffers and the write buffers. To mitigate this problem, you can chose to commit data to disk at regular intervals. Each time a commit happens, all data in the circular buffers that is marked as wanted is sent to the write buffer (which may cause it to be written if it fills), then anything remaining in the write buffer is also written (creating a partially full buffer on disk). Normally we only write to disk when we have full disk blocks to write; setting commits every few seconds will be inefficient, especially in cases where you have a large number of data channels (with n channels you force a minimum of n extra writes).

Data that has been overwritten with other data on disk

This is most unusual on modern systems, but it was quite common in the early days of Windows when rogue programs (or even the operating system) could trash the filing system. If you do suffer from a disk crash and need to recover the disk, STOP USING IT. Any operation that writes data to the disk will reduce your chance of recovering disk blocks. A more common problem is to delete a file you wanted. You can often get such files back from the trash can. If you have done a thorough job of deleting (by asking the system to not save the deleted file), a data recovery expert may still be able to recover the files AS LONG AS YOU STOP WRITING DATA TO THE DISK. Such recovered files may have a few missing or overwritten blocks, but `S64Fix` should still be able to get data out of it.

Corrupted media

Writing data to any device has an error rate... (though it will likely be vanishingly small). Put another way, with any device, if you write enough data you will get errors. Usually, errors are hidden by automatic error recovery systems; some drives have a S.M.A.R.T. feature that claims to give predictive reliability information. The general rule seems to be that if you ever see bad sectors on a drive, you are likely to see more. A new drive (even allowing for the time to move everything) is far cheaper than the time and effort to recover data from a bad device.

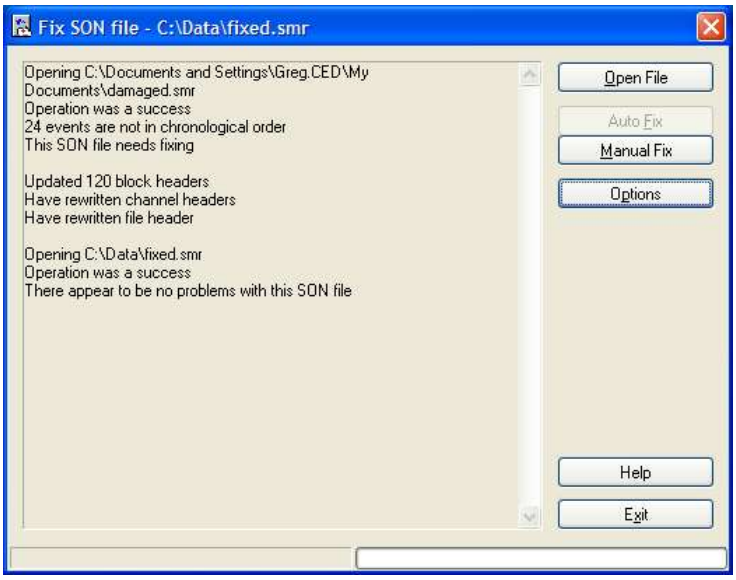
There are many programs available (some with free trial periods) that will attempt to get back data from hard drives and optical media. Search the internet for "Recover data from" followed by the type of your storage medium for a long list of products.

As long as the recovery program does not try to "help" you by cutting out bad blocks, once you have a file that is the same as the original but with a few blocks of rubbish in it, S64Fix should be able to read back the undamaged data.

A popular way to lose data is to write it to optical media and then leave them in direct sunlight... If you archive to optical media, store them somewhere dark and cool. We have also seen problems when copying large files across networks and when archiving to thumb drives.

The SonFix data recovery utility

The SonFix utility program recovers damaged 32-bit (.smr) Spike2 data files. See the S64Fix program to recover 64-bit son files. A Spike2 data file has a header followed by a channel description list, followed by data blocks. Each data block has links to the previous and next data block of the same channel. If a file becomes damaged, these links can be broken, rendering the file useless. The SonFix program scans the file, rebuilds the links for each channel and corrects the file information in the file header. There are three common types of damage that occur to Spike2 data files:



1. The file is damaged as it is sampled. This can happen if your system loses power. The next time you run Spike2 it will tell you where the data can be found and recommend that you run SonFix to repair the file.
2. The file is damaged due to some other disk accident (such as a disk crash or when archiving and restoring files to tape, CD, DVD or some other removable media). As long as the damage is not at the start of the file, the file can usually be fixed and undamaged data recovered.
3. You can generate event or digital marker data with multiple data points sharing the same data file time (when the actual time difference between items is less that one file time unit). The SON file system is not able to completely deal with this situation which may cause problems during analysis of your data files. Spike2 6.10 and later versions will warn you if this situation exists when sampling stops; SonFix can usually fix the problem by slightly adjusting the times of overlapping items.

SonFix relies on the file header and channel list being intact. However, if the program does not recognise the file as a Spike2 data file, don't despair. It is possible to patch the file header with a header from a similar file, and then recover the data. Contact the CED Help desk if you are in this situation.

If you record a very large file with the Big file option (many GB in size), it is possible that it may be too large for SonFix to recover it. This happens when the memory required to hold a list of all the data blocks (and potential data blocks) in the file exceeds the addressable memory available to a 32-bit program. We have no work around for this condition.

Using SonFix

If you have a disaster, check that your disk drive is not damaged. Writing more data to a damaged drive is likely to cause further problems and may render your data unrecoverable. If the hard disk is damaged you must repair it with a suitable disk utility program or find someone who can do this for you before running SonFix.

Next, find the data file. If you have to use a disk repair utility, you may find it useful to know that the first two bytes of a data file hold the file revision followed by the ASCII characters "(C) CED 87".

In most cases there will be no disk damage and the file will be present with a non-zero size. If your data is very valuable, you might want to save a copy of the damaged file in case you need to send us a copy for a difficult recovery. Run SonFix by double clicking its icon (it is in the same folder as the Spike2 program). Click the Open File button and select the file to repair and SonFix will tell you if it can repair the file. Click the Auto Fix button to carry out the repair. If SonFix reports that the file was repaired without any problems, it should be possible to open it with Spike2. As a final clean up, you can use Spike2 to export all the data in the repaired file to a new file; this will eliminate any unused file space.

The Options button opens a dialog that controls the tests carried out on a data file to detect damage and sets how event and marker channels are checked and repaired. If the data file is heavily damaged, it may be

necessary to use the Manual Fix option. Contact CED if you have problems fixing a file or using the manual fix tools.

What cannot be recovered

Data that was not written to the physical disk cannot be recovered. To keep the system responsive and to make high speed sampling efficient, Spike2 buffers data in memory as much as it can. The operating system also buffers data in memory before writing it to disk. However, with data in memory, a power loss or system crash will lose data.

If you use the Flush data to disk option in the sampling configuration, you can protect yourself against disaster at the cost of some loss of performance. If you do not use the flush data option you increase the risk of data loss if your system goes down.

Batch Processing

There are two ways to test or fix more than one file at a time:

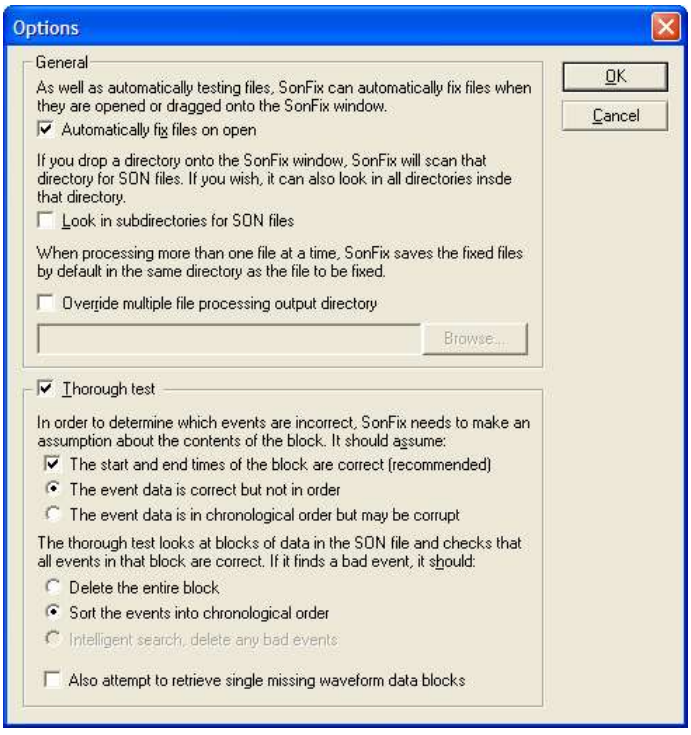
1. Use the Open File button on the main window to show a standard File Open dialog box in which you can select as many SON files as you wish to test or fix.
2. Drag files from Windows Explorer or My Computer and drop them on the SonFix window is the other way of testing or fixing several files at the same time. You can also drag a directory onto the `SonFix` window, and it will test or fix all the SON files (files with a `.smr` extension) in that directory, and optionally all the SON files in any sub-directories (this setting can be changed from the Options window).

If you want files to be fixed automatically, you can specify this in the Options window. Fixed files have `Fixed` added on to the start of the file name. The fixed files will, by default, be placed in the same directory as the damaged files, but this can be changed from the Options window.

After `SonFix` finishes testing and optionally fixing your data files, it displays a list of the files that needed fixing, and a summary of the number of files tested and fixed.

SonFix Options

To customise `SonFix`, select `Options` from the main `SonFix` window. As well as the options described above, there is also a section in the options dialog box entitled Thorough Test, which enables a complete test of the times of event and marker data. If selected, the thorough test is performed on files after the main test, and checks all events in the data file to make sure that they are in the correct order. To help with the error detection, you can select various assumptions that SonFix should make when it attempts to recover data from a damaged file:



Block start and end times are correct

You should normally select this option. It tells SonFix to assume that only the actual event data is corrupt, not the data block headers. If, when you use this option, you find that a great deal of event or marker data is deleted, it would be worthwhile trying to fix the file without this option to see if that helps.

Event data is correct but not in order
Event data is in chronological order but may be corrupted

You can choose between one of these two possibilities, which determine both how the event data is checked and also how it may be fixed. There is no correct choice here; you may simply have to try both options to see which gives the better fix. To start with, selecting data in order but corrupted is probably a better guess.

If SonFix detects event or marker data corruption, it can fix problems in these ways:

- by deleting the block containing the corrupt events (a good idea for heavily damaged files where you care more about the waveform data than event or marker data), or if the other repair option fail.
- by sorting the events into numerical order or, in the case where two events have the same time, adjusting the time of the second event to be just later (for example for a file containing RealMark, WaveMark or TextMark channels where you care more about the data they contain than the times they are stored at). If adjusting a time causes an event to be at the same time as the next event, then the next event is also moved. This repair mechanism is only available if you are assuming that event data is out-of-order but not corrupt.
- by doing an intelligent search for bad events and deleting the minimum number of events to leave the data in order. This is usually the best option. It is only available if you are assuming that event data is in order but corrupted.

Also attempt to retrieve missing single waveform data blocks

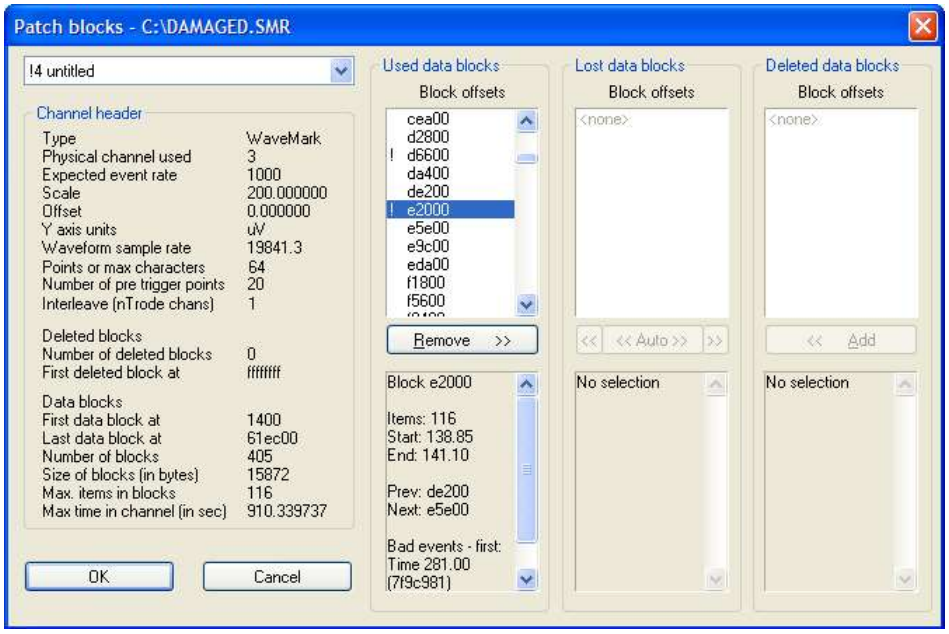
If a waveform data block has a corrupted header, it may not appear in any block list. If the used list has a gap where the forward and backward pointers both agree on the position of a missing block, and no other channel claims it, SonFix can recreate the block header. This is possible because waveform data is sampled at a known rate; this cannot be done for event or marker data. This option takes advantage of this to retrieve single missing blocks of waveform data. Retrieved data blocks may contain bad waveform data, but recovery removes gaps in the channel, which could be useful.

Options to select when fixing duplicated event times

If you are here because Spike2 reported duplicated event times after sampling, the options to select are: *Block start and end times are correct*, *Event data is correct but not in order* and *Sort the events into chronological order*.

Manual Fix

The Manual Fix button leads to the Patch blocks dialog in which you can inspect the lists of used, deleted and lost data blocks for each channel and other channel details. You can also move blocks between these three lists for each channel. Unless you are certain that you know what you are doing it is best to only use Auto Fix; the manual fix tools are intended for use primarily by CED personnel or under CED direction.



The Patch blocks dialog shows information for one channel at a time; the channel selector is at the top left of the dialog. Channels with detected problems have an exclamation mark before the channel number. You should be able to ignore channels without such an indicator.

The channel header information is displayed on the left side of the dialog. The information shown includes general channel information such as the channel type and sampling rate, information about any deleted blocks (these are disk blocks that held data when the channel was deleted at some point in the past, and have not yet been reused) and details of the channel data blocks themselves.

The detailed data block information is shown at the right of the dialog. All data blocks are multiples of 512 bytes in size. When SonFix scans the data file, it searches for data that looks like the start of a block at each 512 byte boundary in the file. The block headers hold the channel number to which the block belongs. When it finds a block, it can use the block size information in the file header to predict where the next block could start. The scan of the file allows SonFix to build up block lists for each channel:

Used data blocks

Used data blocks are blocks of data that are correctly linked into the channel and that do not show any problems. For a channel that is OK, all of the data blocks will be shown in the Used list (with possibly some in the deleted list).

Lost data blocks

Lost data blocks are blocks that apparently make up part of the channel data (because they have this channel number in the block header) but which are not connected properly to the rest of the channel data or the header information does not make sense. Fixing a channel usually comes down to moving these lost blocks into either the Used list or into the Deleted list.

Deleted data blocks

Deleted blocks are blocks forming part of the deleted channel data. When you delete a channel in a data file, the chain of blocks belonging to the channel are placed in the channels deleted block list. If you reuse the channel with the same block size, the deleted blocks are reused. If you re-use the channel with a different block size, the deleted blocks are abandoned. The presence of deleted blocks does not indicate a problem.

Each set of information shows a list of data blocks above, and information about a single (selected) data block below. The list of data blocks shows each data block's address within the disk file as a hexadecimal (base sixteen) number. Blocks that are deemed invalid (either in relationship to the adjacent blocks in the list or as a result of internal consistency checks) are shown with a preceding exclamation mark, blocks that make up part of a coherent chain of blocks descending from an initial block are shown indented to the right with the initial block not being indented. The block information shown in the area below shows the number of items, the time range covered by the block, the previous and next blocks in the chain and details of any errors detected.

Use the buttons at the bottom of the block lists to move blocks between lists. The Remove button below the Used list moves the selected block into the deleted list, while the Add button in the deleted list moves a block into the Used list. However these options are rarely required compared to the buttons below the Lost list. The button marked '< <' moves a block, or set of blocks, into the Used list. Normally it will only move the selected block but, if this block is the start of a coherent set of blocks all shown indented to the right, then all of the blocks within this coherent set are also moved. The button marked '> >' does the same thing, but it moves blocks into the Deleted list.

The third button is marked '< < Auto > >'. This does the following:

1. Attempts to move all of the lost blocks into the used list, as long as this will not result in a corrupt or invalid set of blocks.
2. If step 1 fails, finds the largest single coherent set of blocks in the lost list that can be moved into the used list without problems and moves it.
3. If no blocks can be moved into the used list without problems, all of the lost blocks are moved into the deleted list.

This corresponds roughly to the process carried out by the Auto Fix button in the main SonFix window, though the Auto Fix button repeatedly executes the three steps shown above until the lost list is empty.

It is not possible to describe how to use the Patch blocks dialog completely as the steps required will vary according to the file damage. Roughly speaking, you should attempt to first find out which lost blocks are hopelessly corrupted and move them to the deleted list, then move all of the others to the used list. Once all of the rubbish is in the deleted list the Auto button should do this for you. It is very strongly recommended that you make a backup copy of the damaged data file before attempting a manual fix so that mistakes can be corrected.

Try1401 test program

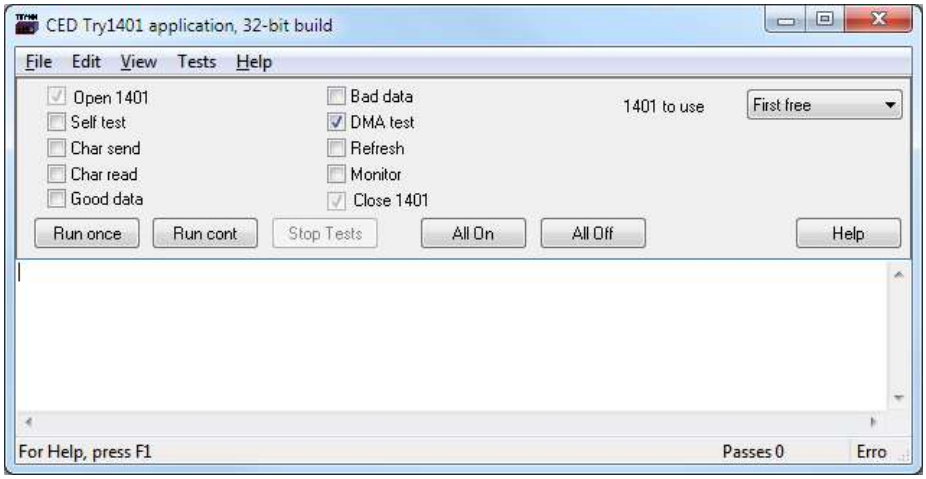
If you are having problems sampling data, and you suspect that the problem lies with the CED 1401, interface card or device driver, then it is well worth running the Try1401 program, found in the same folder as Spike2 (the program file is called `Try1432.exe` on Windows systems).

The Try1401 program exercises the 1401 in much the same way as Spike2 does, but it also checks every byte of data transferred and reports errors in a way that is useful to CED engineers, particularly when there are data transfer problems. Unfortunately, the sternest test of data transfer we know of is the Spike2 program, so it is possible for Try1401 to pass a system that fails in Spike2 (but this is most unusual).

The Try1401 program can also run the 1401 self-test and decode any errors. These can vary from simple problems with the calibration of the 1401 inputs and outputs (usually not a serious problem), to serious internal errors associated with damaged components. You should select the self-test option if the 1401 flashes the Test LED on power up when no other external equipment is connected to the 1401 inputs.

Running Try1401

Double click the program icon to start the program. You have a choice of test options and of running once or continuously. Errors are written to the text window, and can be copied, cut and pasted. If a single pass of the test doesn't show a problem, run the test continuously and leave it for several hours.



Most of the tests are to check the data path between the host computer and the 1401. If you have a problem with your 1401, the CED engineer who helps you to sort it out will most likely ask you to run this program and email the results. For a confidence check, the only option that is needed is the DMA test. The remaining tests help CED engineers to narrow down problems. The individual options are:

Self test

This runs the internal 1401 self test and interprets the results. Please remove all connections from the 1401 except the power cord and the data interface cable as other connections can cause self-test failures.

Char send

This checks the general data path from the host computer to the 1401. If you have a USB connection, problems indicate some sort of installation failure. For the other interface cards data corruption problems are rare, and indicate either a damaged data cable, or bent pins on the 1401 or the host interface card. A time out error in this test usually indicates an interrupt problem.

Char read

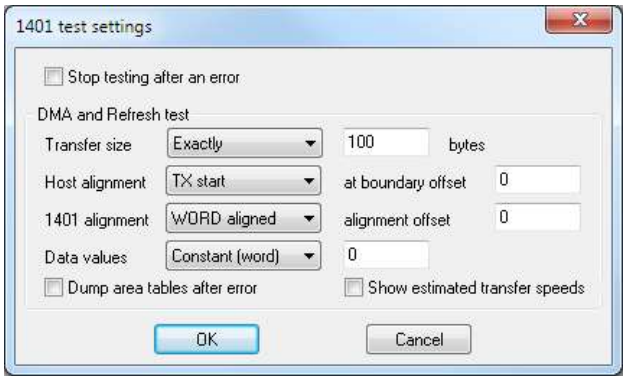
This is similar to *Char send*. It checks the general data path with an emphasis on reading back data.

Good data/Bad data

These are further variations on the *Char send* and *Char read* tests. You can usually skip these tests unless a CED engineer asks you to run them.

DMA test

This test checks out high-speed block data transfer between the 1401 and your computer. There is no point running this test if the *Char send* or *Char read* tests have failed. This test transfers data blocks of various sizes and alignments and checks that the data transferred correctly. Unfortunately, Spike2 is the sternest test of data transfers that we know of, but this test will usually detect any transfer problems. There are additional options in the Tests menu DMA settings… command:



You will not normally need to set any of the values in this dialog unless a CED engineer needs additional data to diagnose a data transfer problem. The fields in the dialog are:

Transfer size can be set to: Unconstrained, Exactly, Greater than or Less than. For all sizes except Unconstrained you must provide a byte count. For all except Exactly mode, it chooses random sizes within the constraint you have imposed.

The data for transfer is held in memory. This memory lies in a continuous block of virtual address space. However, the physical memory that makes up this address space may be mapped anywhere in computer memory. This usually means that a data transfer is broken up into sub-blocks of contiguous physical memory. Alignment relates to the position of data blocks relative to starts and ends of these sub-blocks. This field only applies to ISA and PCI interface cards using DMA transfers; we have no knowledge of alignment for the USB interface. You can choose from Unconstrained, TX start, TX end, RX start and RX end. TX = transmit to 1401, RX = receive from 1401. The at boundary offset field sets the relative position of the transfer start or end to sub-block boundaries. You can set from –4095 to +4095 (but useful values are usually in the range ± 100.

Data values can be set to be Random, or you can choose to set values based on bytes, words (2 bytes) or longs (4 bytes) and the values can be a constant, an upward ramp or a downward ramp.

Check the Dump area tables after error box to print a lot of extra info about any failing DMA transfer. This can help a CED engineer to diagnose a fault.

Show estimated transfer speeds prints the approximate number of kB per second for each transfer. The rate depends quite strongly on the size of the transfer as there is a time overhead to setting up the transfers and detecting that they are done. There will always be some transfers of 250,000 bytes and these give the best estimate of the available maximum speed.

File menu

There are options in the File menu to update the Power1401 and Micro1401 mk II or -3 flash memory. These are described in the Owner's manual for your 1401. You can get the downloads from our web site. See the About Spike2 page for links to the monitor downloads.

There is also the 1401 info… command. This prints out information about the 1401 device driver and the 1401, for example:

```
1401 type       = Micro1401
Monitor revision is 20.14
1 megabyte base memory
Micro1401 main card is 2501-01 C-211
Block transfers use DMA, multiple transfer areas
ADC channel sequencer is FIFO at 4 MHz
Supports up to 256 channels, 3uS ADC block
No extra ROM in spare slot
```

21: Multimedia recording

Multimedia recording

The s2video application and file names

You can use the `s2video` application to record multimedia files automatically whenever you sample a Spike2 data file. This description is updated for version 2, first released in March 2019. Version 2 is a significant upgrade and uses a different file format (`.mp4`) by default, though it can also generate `.avi` files (as done by the previous versions). Versions of Spike2 before [9.06, 8.16] expect `.avi` files for review; later versions will review either `.avi` or `.mp4` files.

This version of s2video also addresses problems with computers with multiple display screens displaying a black rectangle rather than the correct image when the preview window was dragged to a different screen.

You can run up to 4 instances of this application at a time. Each instance will record one video and/or one audio track. The quality of the recording (video resolution and frame rate and audio sample rate) will depend on the hardware and software that you have installed. You can use codecs to compress your multimedia data either on-line during capture or off-line after capture ends.

> *Important:* By saying that you can run up to 4 instances, we mean that the software allows this; it does NOT mean that we guarantee that your system will have sufficient bandwidth or disk writing speed or CPU power to do it.

The multimedia data files are saved alongside the Spike2 data files. You can view these files within Spike2 using the View menu Multimedia Files command. From Spike2 version [10.03], the `Listener()` script command counts the number of active listener devices.

File names

Spike2 samples data to a temporary file. When sampling ends and the user chooses to save the file, it is given a permanent name. The multimedia recorder tracks the Spike2 activity and names the data files to match Spike2. For example, if the final Spike2 name is of the form:

```
C:\PathToData\data\data23.smrx        (or data23.smr)
```

and if there were two instances of `s2video` running, they would name their files:

```
C:\PathToData\data\data23-1.mp4       (or data23-1.avi)
C:\PathToData\data\data23-2.mp4
```

Availability of files

Multimedia files generated by `s2video` are not available for use until the Spike2 data file has been saved. If they require off-line compression, this is done as a background task so that data capture can continue. In this case, the multimedia files are not available for review until the compression task has been completed. Alternatively, you can store files uncompressed (they will likely be huge and take a long time to move between drives) and then use `mp4comp` or `avicomp` to compress them at a more convenient time.

Video data

Video generates a lot of data. A high resolution display of 1280x720 with RGB data using 8-bits per colour at 30 frames per second generates 1280x720x3x30 bytes of data per second, which is 83 MB per second, close to 5GB per minute. By taking advantage of the fact that our eyes are less sensitive to edges in colour than in monochrome, formats such as YUV reduce this rate to 3.3 GB per minute, but this is still a lot of data. Fortunately, most frames are very similar to the previous frame, and adjacent pixels are often very similar, so it is usually possible to compress images substantially. When time is not of the essence (for example when compressing a film for transmission), very large compressions ratios are possible, in excess of a factor of 200. However, for real time work without specialist hardware, more modest ratios are achieved (maybe a factor of 10 to 20). Compression and decompression are typically done with codecs (from **co**mpress/**dec**ompress). Please read the codec information carefully and select a suitable one.

Video data is stored as frames, either as key frames that contain all the data needed to render them, or as frames that hold the difference from the previous frame in some way. If the video source does not compress the data (for example it offers RGB or YUV format data), all frames are key frames. If it generates MPEG data, this will be compressed (to some extent) and will contain key frames and non-key frames.

Video source devices announce the *frame rates* that they support, in Frames Per Second (FPS), and in theory, they deliver frames at this rate. In practice, they may deliver frames at a rate that is very different (usually slower) and the rates may vary during capture (for example with the illumination of the scene). For example, my (very cheap) web cam reports 30 FPS (frames per second), but at 640x400 resolution it actually generates 25 FPS unless the scene is not well lit, when the rate drops to a few frames per second.

Frames have a time stamp that records when they were sampled. They can also have a sequential frame number that is ideally the time offset from the start of sampling divided by the frame interval (reciprocal of the frame rate). However, there can be adjustments of this to cope with frame intervals not matching the advertised frame rate, making frame numbers far less reliable than frame time stamps. If there are gaps in the frame numbers, the system counts the gaps as 'dropped/missing frames', even though the frames may never have existed in the first place.

File formats

Multimedia files contain streams of data for video, audio, metadata (titles, commentaries and so on); many different data types can be stored. There are many container formats, in Windows you will see files with the extensions: `avi`, `mp4`, `mov`, `asf`, `wma`, `wmv`, `wm`, `mpg`, `mpeg`... the list is endless. We originally supported only the `avi` (Audio Visual Interleave) format; more recently we also support (and recommend) the `mp4` container format.

Timing and MP4 files

The `.mp4` files act as a container for streams of data; in our case we are interested in audio and video data. The `.mp4` container format preserves the time stamps of the source data video frames, so it should not matter if frames get dropped or are given frame numbers with gaps; the output file should still accurately represent the timing intervals of the source frame stream. Note that the time of the very first frame in the video is always 0. Additional frames can be lost if the system cannot process the data stream and write it to disk fast enough, but the frames that do make it into the final file should still be correctly timed.

From S2Video 2.03 onwards we adjust frame times to allow for the delay between Spike2 starting to record and the first frame reaching the capture system. The first frame time in the file is always 0, so we adjust the first frame interval to make frames after the first match the actual times; the very first frame time is not useful.

Timing and AVI files

Previous implementations of s2video wrote `.avi` files; this was the only format available to us when we originally wrote s2video. This format describes a list of frames at a fixed time interval; it does not store frame time stamps. Frame numbers are the only useful timing information, and missing frames are coded in the output stream as frames with no data. To generate an output file with reliable timing relies on all components in the data stream from camera to output file keeping track of time and frame number and inserting empty frames (or dropping frames) to make the output file time match the file frame rate and frame number.

Problems can also arise when codecs (devices used to compress and decompress data to keep file to a manageable size) are used. Some codecs ignore the dropped/missed frames, resulting in files that run at the wrong rate.

From S2Video 2.03 onwards we measure the delay between Spike2 starting to record and the first frame reaching the capture system. Unless you override this in the Settings menu Configuration, we store this delay as the multimedia offset in the AVI file.

You can choose to generate `.avi` files with an option in the Configuration menu. We provide this option for reasons of backwards compatibility. However, we strongly suggest that you do not use this format.

Multimedia system requirements

To run Spike2 data capture and multimedia capture simultaneously, you need a suitably powerful computer. In August 2023, a new system will likely have four or more processor cores, 8 GB or more of memory and a 2 TB hard disk or a SSD and run Windows 10 or 11. This is the type of system you need. The software will take advantage of multiple processors and hyper-threading.

The Multimedia data capture and replay software (used for video recording and playback) is based on Microsoft DirectShow, which is installed on your computer. The `dxdiag` utility will report the DirectX version and test it (open the Start menu, select Run, type in `dxdiag` and click OK).

To record video, you will need a web cam or a video camera. There is a very wide variety of these available. You will need to install drivers for your camera; these will normally be supplied with it. Alternatively, your system may recognise the camera when you plug it in and locate the correct drivers for you.

Be aware that you get what you pay for. Cheap web cams can work very well, but they can also have unexpected problems. For example, if your subject is not brightly lit, many web cams will increase their frame exposure time to compensate, and if the exposure time is too long, the frame rate will reduce.

For video, you will also need a video codec. Choosing the right codec is one of the most important steps in getting recording to work usefully. The choice of codec is described here.

To record audio you need a sound card in your computer or a video device that also records sound. If you record audio, this may cause the DirectShow system to take its timing from the audio device rather than referring it to the Spike2 sampling clock. If the audio clock and the sampling clock run at differing rates, the video timing will then 'drift' with respect to the sampling.

If both your camera/audio device and 1401 link to the computer by USB, and your computer does not have independent USB ports, the two devices will compete for bandwidth. This can degrade the maximum sampling rate of the 1401 and the maximum frame rate of the video.

If your computer has both USB 1 and USB 2 ports, beware of plugging a video device (or a 1401!) into a USB 1 port. This severely reduces the bandwidth. Even modest video needs a bandwidth of many MB per second and USB 1 will limit you to just over 1 MB per second. This is unlikely to be an issue for modern computers.

Problems

One of the good things about using DirectShow is that it allows the use of an extremely wide range of video and audio devices. Sadly, this also leads to problems. Although most companies ship good-quality software drivers for their multimedia products, not all ship DirectShow drivers. Ironically, almost all high-volume, consumer (i.e. cheap) web-cams and video cameras work with DirectShow, but some low-volume, specialised (i.e. expensive) devices do not. We regret that there is nothing we can do about this. If your device has a DirectShow driver (shows up as a possible source device in s2video) but will not connect, if you send it to us with the relevant software, we may be able to diagnose the problem, which may lead to a fix.

The s2video Edit menu has a Copy Graph option that generates a text description of the DirectShow internal connections that can help us to diagnose problems.

Getting started

When you run the `s2video` application for the first time, there will be no devices selected. The window title Spike2 Video Capture 1 means that this is the first instance of the application. If you open another, it will be number 2, and so on. The application stores information in the system registry for each instance. This includes the window position, the video and audio recording devices and any codecs that you select to compress the multimedia data so that it occupies less disk space.

Your first task is to choose the devices to record from. Click the Settings menu and then select the Video Device item. A new menu will pop-up with a list of the possible video sources plus No Video Device, which will have a tick next to it. Choose your capture device.

The next task is to select an audio device (if you need one). If you select an audio device, an audio level meter appears in the dialog. You do not have to select an audio device; recording video only is allowed. If you set an audio device, this may force DirectShow to take timing information from the audio card rather than from the Spike2 sampling device.

You must set at least one device to sample data. Once you have set your devices, new items for Video Device Properties, Video Capture, Video codec configuration and Audio Device Properties are added to the menu.

Once the devices are selected, use the Configuration dialog to choose video and audio codecs. If your chosen codec is fast enough, you can compress data on the fly. Alternatively, you can choose to add newly sampled data to a queue of files for compression after capture. You can opt for no compression, but uncompressed video files will tend to be very large.

Each time you run the program to capture video it is important that you open the Settings menu Video Capture dialog, set the desired configuration and click OK. This should ensure that the camera runs with the settings you select.

This software is written to work with a very wide range of hardware devices. We cannot predict what you will see on the screen in the property dialogs for the devices as the device manufacturers define them. We can tell you the types of control to expect from our experiences working with a range of different cameras and sound devices. If you have problems setting up your video or audio device please refer to the documentation that was supplied with them. We will be able to offer general guidance only.

Important
Always check that the video setup is working correctly before you rely on it for an important recording. It is a good idea to run a recording session with known data so that you can verify that when you review the data, everything lines up to the timing accuracy you require. One way to align video is to point the camera at the 1401 front panel and use the output sequencer to turn one of the front panel digital output LEDs on and off every 5 seconds. Connect this to one of the digital inputs and record it as a Level event channel for a suitable period. When you review the result you can check that the images align with the LED on and off times. Remember that timing can be no more accurate than the video frame rate allows. Also, there may be a fixed time offset due to delays through the video recording system (which can be positive or negative); you can set an offset in the replay window to compensate for this.

File menu

This contains a single option, Exit, which will close the application unless you are recording or compressing data files.

Edit menu

This menu holds a single option:

Copy Graph
This copies the underlying DirectShow filter graph as text to the clipboard. If you have problems we may ask you to send us the output of this to help us understand what is going on. Here is an example from my computer running with a web cam (*Live! Cam Socialize (VF0640)*):

```
Spike2 Video Capture 2.07 April 2021
Build: Debug x86 Apr 14 2021 09:29:23
Operating system: Windows 10 x64 build 19042
Run: 14:40:39 Wednesday, April 14, 2021
Always on top: Yes
Show Preview: Yes
Disable drift: No
Disable video in properties: No
Disable frame dropper: No
Time stamp in frame dropper: Yes
Disable EVR: No
Use slow rate: No
Slow rate in Hz: 1.00
Max rate in Hz: 200.00
Video codec: Xvid MPEG-4 Codec
Audio codec: WM Speech Encoder DMO
Audio uses Video: Yes
Video compression: After
Audio compression: After
Drop down window height: 519
Window rectangle: (2631, 297, 3553, 816), (922 x 519)
Force AVI: No
Raw Video: No

Capture: Microphone (VF0640 Live! Cam Socialize)
Microphone (VF0640 Live! Cam Socialize)
    Smart Tee 0003
        Mux
            File Writer
```

```
        Audio Stats Filter

Capture: Live! Cam Socialize (VF0640)
Live! Cam Socialize (VF0640)
    Smart Tee
        Capture Frame Dropper
            Mux
                File Writer
        Preview Frame Dropper
            AVI Decompressor
                Enhanced Video Renderer

Preview: 24.9145 FPS (40.1373 ms, sd 7.87152 ms)
Capture: 24.9276 FPS (40.1162 ms, sd 2.57834 ms) 2408 frames, 0 missing, first 1, last 2408
```

General information
The information starts by identifying the version of S2Video, the operating system it is running on and the time and date.

Settings
The next block of information is the settings that are active.

DirectShow graph
The Settings are followed by a description of the multimedia components that are linked together to capture your data. Each capture device that you selected is announced by a line starting with `Capture:` followed by the device name, then a tree of the filters that are connected together to generate the result. If something went wrong during graph building you may also see entities with the name `Orphan:`, which indicate items that failed to connect. You will see that the `Mux` and `File Writer` devices are common to both the video and audio routes as the `Mux` (multiplexor) is the point where the two paths (audio and video) merge before being written to disk.

Frame information
The two lines starting `Preview:` and `Capture:` give information about the observed Frames Per Second through the Frame Dropper Filters followed by the equivalent value as an interval in milliseconds and the standard deviation of this value.

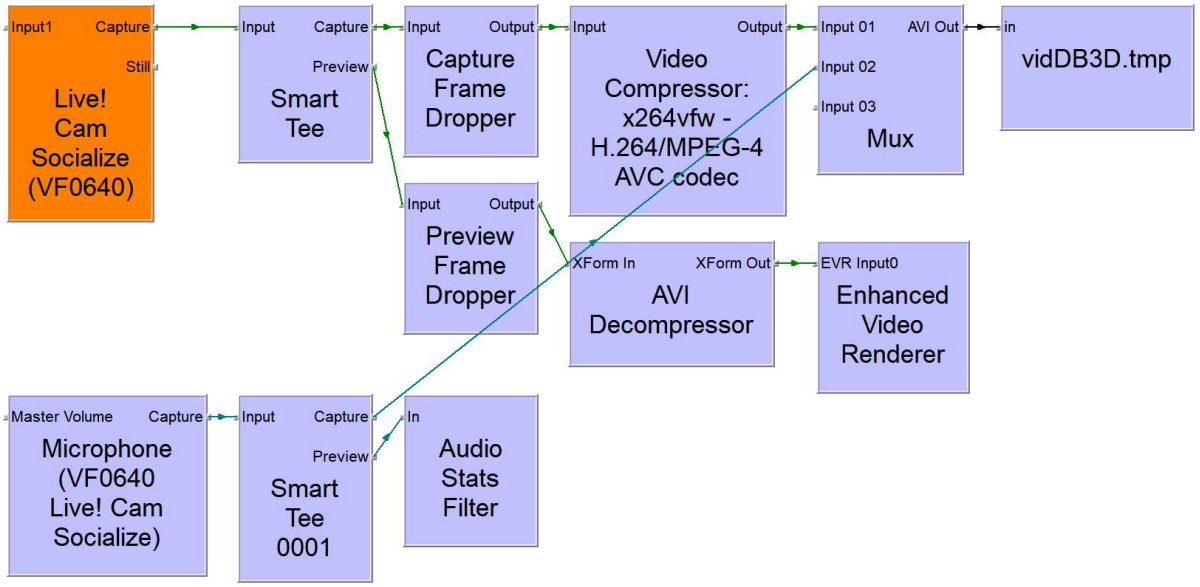
The Preview path is given a low priority (so may well not see all frames, which can lead to a high standard deviation on the mean frame period).

The Capture path should see all the frames generated by the camera unless the queue of data to be written stalls, in which case frames are dropped before they reach this filter, resulting in large `sd` (standard deviation), values. In this case, all the frames were captured at almost exactly 25 FPS with very little timing jitter. The `first` and `last` values are the frame numbers of the first and last frames passed through the capture system.

The Capture Frame Dropper saw 2408 real frames and all frames were passed to the Frame dropper filters.

More than you wanted to know
The remainder of this topic describes the graph and the details of the frame information. You can safely ignore it unless you are having problems and would like to understand more about what is going on in the S2Video application.

Graph details
Items at the same level of indentation indicate levels of connection. There is a useful free tool available on the internet, *GraphStudioNext*, that can be used to visualize the state of a running S2Video program. There are two variants of this (32-bit and 64-bit). Currently the 32-bit version must be used as we provide a 32-bit version of S2Video. If you load this program while S2Video is active and use the *File* menu *Connect to Remote Graph* option you will get something like the following (it can look different depending on the camera type and the options you select), but the basic features (sources on the left and targets on the right) will be the same:



Note that this example graph (using an AVI file format and the x264 video compressor) does not match the Copy Graph output, above. The arrows show the direction of data flow between the filter *Pins* (a pin is a possible connection point of a filter). The various filters that can appear in the graph are:

Filter	**Purpose**
Camera	In this case `Live! Cam Socialize (VF0640)`, but the name depends on the device. If you capture audio only this device will be absent.
Smart Tee	This is added when a source device does not have separate *Capture* and *Preview* pins. It splits a data stream into two identical streams. Capture pins take precedence over Preview pins when resources (such as CPU time) are scarce. If the source already has a preview pin, this is not added.
Capture Frame Dropper	This is a CED-written component that can drop frames to achieve a lower frame rate than the camera generates (for the Slow frame rate setting) and can time stamp frames. It also generates statistics on frame intervals. You can omit this with a Configuration option. If the video stream is compressed, starting and stopping the stream is deferred until the next Key frame (otherwise the data stream is nonsense).
Preview Frame Dropper	This is similar to the Capture Frame Dropper, but is used for Preview streams.
Video Compressor	This is present if you select a codec for on-line video compression.
MPEG Decoder	Only if your camera source is MPEG. If your camera does not have a Preview pin, this is added to the Preview stream to uncompress the data for the Preview display. It will also be added to the Capture path if you select a video codec as these need uncompressed data to work on.
Mux	The multiplexor. This combines the video and audio stream (if both are used) and formats them for the output file. This links to the File Writer block.
File Writer	When not capturing, this writes to a dummy file with a random name (`vidDB3D.tmp` in this case). When capturing it captures to the target file. We use a dummy file as it is usually much quicker to swap between files than disconnected this device, rebuild the graph and start again, which can take several seconds before we are in a suitable state to record data.
AVI Decompressor	Present when the Preview output needs modification before passing to the Video Renderer.
Video Renderer	We use the Enhanced Video Renderer by default, but you can disable this in the Configuration which causes us to use the default Video Renderer. The Enhanced Video Renderer should be able to take advantage of hardware acceleration in your graphics card. The option to disable it was added for backwards compatibility (but we have also found the default renderer to be a better choice on some Windows 10 systems). S2Video will also disable it if you run with a remote desktop.

Colour Space Converter	May be added if the video stream data format does not match that of the (Enhanced) Video Renderer.
Audio Compressor	This is present if you select a codec for on-line audio compression.
Audio Stats Filter	This is the device that calculates the values for the volume bar (below the video window).
DV Splitter	Added when a Digital Video source device is used. We have only tested this with one example of such a device many years ago, so it is quite likely to not work! If you have problems with such a device and can send it to us (with the support software), we will attempt to get it working.
ffdshow Raw Video filter	This is present if you have downloaded the ffdshow filter and checked the Add Raw video box in the Configuration menu. It is used to change camera output formats into ones compatible with DirectShow.

Frame numbering and AVI files

As far as we can tell, frame numbers are assigned to frames by dividing the frame time stamp by the time per frame claimed by the video source. However, we do not know the exact mechanism used for this, nor how the system copes when frames arrive at a faster rate than advertised (when frame numbers would be duplicated). This frame number determines where in the output AVI format file a frame is placed. If frame numbers are not sequential, empty frames are inserted in the output.

Taking my camera, which claims 30 FPS, but in a 50 Hz mains environment actually generates 25 FPS, as an example:

Frame time (ms)	0	40	80	120	160	200
Frame number	0	1	2	3	4	6
Calculation	0/33.3	40/33.3	80/33.3	120/33.3	160/33.3	200/33.3

In this example, we get five consecutive frames numbers, then one 'missed' frame. In the (unusual) case of a camera generating frames faster than advertised, this causes duplicated frame numbers. We do not know what DirectShow does in this case; it may throw the frame away, or it may number it consecutively. Either way, this leads to uncertainty in the frame times if we do not have frame time stamps (as is the case with AVI files).

We have observed that some codecs do not cope well with missing frame numbers - we have seen codecs that ignore missing frames and others that try to insert frames to compensate. This leads to uncertainties in timing. This is why we much prefer to use the `.mp4` format where the frame times are stored as part of the file and no assumption is made of a constant frame rate.

MP4 files

By contrast with AVI files, MP4 file store the times of the frames. The only problem with this is when a camera puts time stamps on frames that are incorrect. We have seen cameras that time stamp the frames to show that they occur at exactly 30 FPS regardless of the actual rate (which could be much less in low light conditions). There is an option in the Configuration to ignore the camera time stamps and set our own to combat this problem. It is worth setting this option if you are not convinced that the camera times match the Spike2 data as this has always given reasonable results in our tests. When reviewing data, you can use the `MMFrame()` script command to get the actual frame times of MP4 files.

View menu

The view menu contains four items:

Toolbar

The toolbar holds short-cuts to preview the video image, to reduce the frame rate to the value set in the Settings menu Set Slow Frame Rate and to open the configuration menu. You can choose to show or hide it with this command.

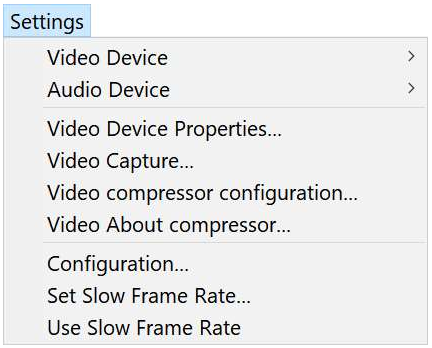
Status bar
You can choose to show or hide the status bar at the bottom of the window. The status bar gives you feedback about your menu selections and information about recording.

Always on top
Tick this menu item to keep the `s2video` application on top of other windows.

Preview
When a video camera is selected this shows or hides the video preview window.

Settings menu

This menu has three sections: Device selection, Device configuration and Program Configuration.



The first section allows you to select the data capture device to use for video and or audio.

The second section may be empty. It holds links to various property and configuration menus that are exposed by the video and audio devices and by any video compressor (codec) that you have selected. In this example, we have not selected any audio device, so only video configuration items appear. We choose the names of the menu items; what they link to depends on what your system makes available.

The final section holds links to the Configuration menu, where you choose how (or if) the video and audio streams are compressed and the codecs used. It also holds the Slow Frame Rate controls (used to reduce the quantity of video information during uninteresting data capture).

Device selection
To record you must select a capture device. In theory, we can capture from any device that provides DirectShow drivers. Capture devices can generate a wide variety of data stream types. We are only interested in Video and Audio and we ignore other streams. Further, to be useful, the format of the data stream must be compatible with (or convertible to compatible with) any codec and the output file.

Video Device
A new menu will pop-up with a list of the possible video sources plus No Video Device. If a device that you want to use does not appear in the list, make sure that it is switched on and connected. You do not have to select a video device; recording audio only is allowed. Once you select a video device, you will see a preview of the image in the video display area of the window (possible after a delay of a few seconds). You can enable and disable the preview with the View menu Preview command or with the first button in the toolbar. If you select No Video Device, the s2video application window will reduce in size to hide the video area.

Audio Device
A new menu will pop-up with a list of the possible audio sources, plus No Audio Device. If you have selected a video device that generates audio as an integral part of the video stream, the From Video Device item is enabled and can be selected. When you select an audio device, an audio level meter appears in the dialog. You do not have to select an audio device; recording video only is allowed.

In some cases, the use of an audio device may disable the system we use to lock the video timing to the Spike2 sampling clock. This will usually not be significant over a period of a few minutes.

Video Device Properties

This menu item appears in the Settings menu when you have selected a video input. What you see in this menu depends on the selected camera. It will probably include controls for image brightness, saturation, colour balance and saturation. It may also have controls for the frame rate (this will set the maximum frame rate that

you can record at), monochrome or colour output and various settings to compensate for background illumination and to cope with fluorescent lighting.

If you can control the frame rate here, set the lowest rate that provides enough detail for your sampling tasks. If you do not need colour and you can request monochrome here; you will save disk space (in compressed images).

If your video device includes audio support, there may also be audio set up controls here. These may replace or extend the controls in the Audio Device Properties dialog.

You will not normally find controls for the image size or the image data format here; these are usually in the Video Capture page.

If the Properties dialog shows disabled items that you would expect to be able to change, you may be able to enable these by checking the *Disable video during Properties dialog* button in the *Configuration* dialog.

Video Capture

This menu item usually appears when you set a video device. It opens a dialog in which you can set the frame rate. It may also allow you to control the image size and format. This may duplicate controls in the Video Device Property page.

Important

The first time you open this dialog in a S2Video session, the dialog displays the default settings for the camera. If you do not use the OK or Apply button, these will remain as the default settings, but may not be used. To be sure that these settings are used you must click the OK (or Apply) button.

For example, we have seen a camera that offers MJPG output at 30 FPS 1280x720 (and other resolutions) that also offers YUV2 output at the same resolutions, but is limited to 10 FPS at 1280x720. The default is MJPG output, however this requires a more complex data processing pathway (the Preview output requires a MJPEG decompressor) than changing the format to YUV2, which does not. As the default state is not mandatory, the system chooses the simpler pathway and runs at 10 FPS, not 30 FPS.

We may be able to make these settings be remembered in a future version, but for now it is advisable to always set the Video Capture settings.

Frame Rate

The Frame Rate sets the frames per second produced by the camera. You will get fewer frames per second than this if the system cannot keep up or if you limit the rate with the Set Slow Frame Rate dialog or with the Spike2 `MMRate()` script command. The lower the frame rate, the smaller the data file. Use the lowest frame rate that gives you enough information for your task. A higher frame rate uses up processor time and disk space. It is likely that a rate of 10 or even 5 frames per second is all you need for many applications. The range of allowed frame rates depends on the camera. Some cameras have a single, fixed frame rate.

Many web cams advertise a frame rate, but the actual rate depends on the light intensity, dropping as the illumination level falls. If the frame rate does not match the advertised rate, you will see a dropped frame count during capture. You can also (more rarely) get this if you select a video compressor that slows things down so much that the capture cannot keep up.

If your data capture source is already compressed (for example it is advertised as MPEG or H264 or similar), we have to be careful about the points where we stop or restart the stream of frames. A compressed video stream holds'key' frames, that hold all the information required to generate the image, and then other frames that hold differences. We always wait for a key frame before we change between dropping frames and not dropping frames.

Output Size

The Output Size is the image resolution expressed as horizontal x vertical pixels. Please use the smallest image size and frame rate that is suitable for your needs. The uncompressed size of the output file is proportional to the product of the horizontal and vertical sizes and the frame rate. Doubling the resolution can create a file that is four times the size.

The remaining settings depend on the device. Unless you have a good reason to change them we suggest you accept the initial values.

Image format

The image format specifies how the camera arranges and compresses the video stream from the camera. There are many different formats possible, all with differing benefits. It is possible that s2video may not work with all the formats your camera can produce. This is because the DirectShow system has to connect the camera source to the (optional) video compressor you have selected, and then to the output data file. To do this, it has to find a set of DirectShow filters that can do any format translation required. If a filter for a particular format is missing or the DirectShow software cannot figure out how to make the connection, the format cannot be used and you will get an error if you select it.

There is a wide variety of image format possibilities. It is worth experimenting with codecs and formats to see which combination gives you the smallest files. If your camera supports a compressed output format, for example MPEG or H264 you should try using this without a codec and compare it with using an uncompressed format with a codec.

If your camera has a choice of MPEG or YUV output, worthwhile combinations to test are:

- MPEG and do not set a video codec
- YUV and a video codec

We have seen an instance where selecting MPEG and the Microsoft MJPEG codec resulted in an output that was several times larger than the camera MPEG output with no codec. If the stream is already compressed, to set your own codec requires the input to be decompressed first, then compressed, which is at best more work and can generate a data stream that is larger than the one you started with.

Troubleshooting video

If you cannot get a video camera that has a DirectShow driver to work with s2video there may not be a lot we can do about it. Please remember that a very large number of devices do work; we have to deal with a generic interface to a huge range of possible video devices and what is possible depends largely on the quality of the DirectShow device driver and any auxiliary DirectShow filters provided with the camera.

Not all video hardware is very good. I have a web-cam that works when my room is brightly lit, but that refuses to connect when it is dim and does not recover until I reboot my machine. It took me a couple of days to figure this one out...

There is a very useful free program, GraphStudioNext (latest version in November 2019 is 0.7.1.59), which you can find on the internet. If it is possible to link up your camera with this program, we should be able to make it work in s2video. The simplest test is to use the Graph menu->Insert Video Source command. Make sure the Graph menu->Intelligent Connect item is checked. Then right-click the Capture pin and select Video Renderers->Enhanced Video Renderer. If this does not connect, there may be nothing that can be done. If it does connect, you should be able to play the graph and see an image.

Video compressor configuration/codec properties

If you select a video codec, the *Video compressor configuration...* menu item appears if the codec uses the old-style Video for Windows compressor configuration interface. Both the XVid and the x264vfw codecs we mention support this dialog. If the codec supports DirectShow directly and has a configuration dialog, the *Video codec properties...* menu item appears.

Both of these options lead to a dialog in which you can modify the behaviour of the codec to suit your requirements. The contents of this dialog are determined by the video compressor. Describing the available options is beyond the scope of this documentation. Usually, the default settings are a reasonable starting point. If you are serious about changing the settings, you should read any codec specific documentation you can find.

The following describes some of the things that you might find in the dialogs:

Number of passes

For real time use you will want to use a single pass.

Data rate/image quality

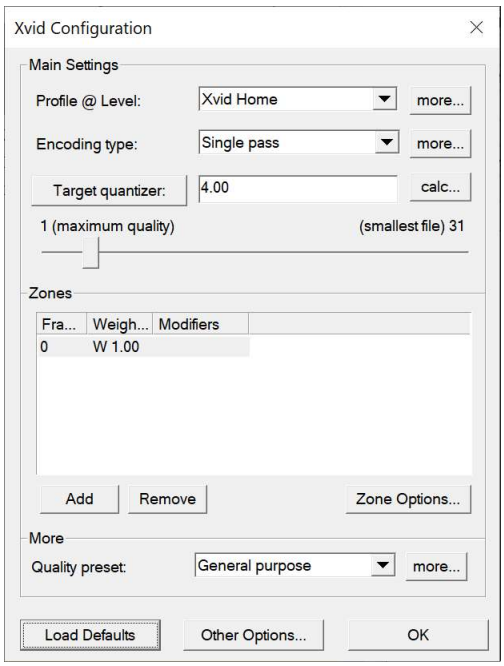
Some devices let you suggest a data rate, others suggest an image quality. In general, the higher the data rate, the better the image quality. We want the lowest data rate for an acceptable quality.

Key/I/B/P frames

Some devices let you specify how often key frames (frames with all the picture information) occur, and how often other types of frames are generated. You will need to read up on video compression to make sensible choices here. The default settings are likely to be reasonable. However, if you intent to 'scrub' the output (move backwards and forwards randomly), the larger the gap between key frames, the longer it takes to move to a new position because a random move requires a seek to the key frame before the desired time, then decoding all the frames between that key frame and the desired position. Usually key frames are emitted using a heuristic based on the quantity of 'change' in the image and an absolute maximum of frames between key frames.

Xvid Configuration

This codec has a large set of configuration options. Fortunately, for our purposes the default settings will likely work, but you can choose to tweak them. I found a description of the options (but aimed at two pass encoding, which we cannot use) at: http://www.divx-digest.com/articles/xvid_setup_page1.html (in August 2020, but refers to a much older version with some differences). This is well worth a read in addition to the following.



The very first time you open this dialog, the contents are likely to look like the picture. This will work, but is not ideal. The minimum change you should make is to change the Quality preset from General purpose to Real-time. The dialog fields are:

Profile @ Level

This sets the general characteristics of the codec and each choice has different combinations of resolution and bit rate limits. For our purposes, you can probably use one of Xvid Home or if you know the number of lines generated by your camera is 720 or 1080 you can choose Xvid HD720 or Xvid HD1080. These will preset some fields to expect to process 1200x720 images at 30 frames per second or 1920x1080 images at 30 frames per second.

If you choose XVid Home and you know your camera resolution and likely frame rate you can use the more... button and in the new dialog Level tab, set the Suggested field to the camera resolution and frame rate. I am not sure how much difference this makes...

Encoding type

You must set Single pass as we are processing on-the-fly. The other options are for off-line use and write a file of information on the first pass and generate the compressed output on the second pass with the benefit of knowledge about future and past frames.

Target quantizer/target bitrate (kbps)

The next two lines are two different ways of looking at the same thing and control how much information is saved into the file. Click the button to set either of these. The Target quantizer is a number in the range 1 to 31 with 1 setting the best quality and 31 setting the smallest file. The starting value is 4.00. You can experiment with increasing this until the quality of the output becomes unacceptable.

You can also set the Target bitrate in the range 16 to 20480 kbps. Larger values create larger files. The starting value is 700, you can experiment with decreasing (or increasing) this. This setting does not guarantee any particular file size; it is more a suggestion to the codec.

Zones

This is not relevant to us. If you are compressing a movie, you might want a fast action sequence to have higher quality than the end credits and you can set zones that start at particular frame numbers where different parameters are to apply. We want a single zone starting at frame 0 with weight 1.0.

Quality preset

The minimum to do here is to set Real-time (and possibly set the Turbo option, see below). If you want to limit the gaps between key frames, do the following:

1. Select Real-time.
2. Click more... and respond Yes to the question to copy the Real-time fixed quality presets to the (User defined) preset.
3. Edit the Maximum I-frame interval field to your chosen value (the default is 300).
4. Do not change other fields (except maybe the Turbo :-) field, which might encode faster at the cost of a bit of quality).

Only reduce the key frame value if this is a real problem for you. If you are dealing with images that are mainly unchanging, the key frame interval is one of the major factors in reducing file size.

Load Defaults

If you think you have messed things up, click this button and start the settings again.

Other Options

This leads to some stuff you do not need to use that is present for diagnostic purposes. In particular to the Display encoding status option, which is positively evil if checked in S2Video as it causes the program to hang.

OK

Click this when you are happy with your settings. If you want to compare recordings, you need to record the same images, as much as you can, for a reasonable time and evaluate the file size and image quality. It is a good idea to change only one thing at a time.

Audio Device Properties

This menu item appears in the Settings menu when you set an audio device. It opens a Properties dialog for your selected device. If the audio device has multiple inputs you can use this dialog to select the one to record. You can also set the volume level of the input. There are likely to be fields for stereo panning, treble and bass, loudness (bass boost) and to force monophonic sound from a stereo source.

With most common sound devices the majority of fields are grey and cannot be used. If the Enable field is not grey, it is important that you use the Pin Line field to display the input to record from and then check the Enable field to select that input. You should also make sure that the volume control (the vertical slider under the Pin Line Input Mix label) is not set to the bottom position, which is zero volume.

You can check the signal level with the Volume control display at the bottom of the window.

If the Properties dialog shows disabled items that you would expect to be able to change, you may be able to enable these by checking the *Disable video during Properties dialog* button in the *Configuration* dialog.

Set Slow Frame Rate

This command opens a dialog in which you can set the maximum frame rate to capture and a slower rate to use when you do not need the rate set in the Video Capture dialog. The slower rate is also set by the Spike2 `MMRate()` script command and can be changed during sampling if the Spike2 Sampling configuration Mode tab option to set the Video FPS is activated. We achieve these rates by throwing away frames captured by the camera that are at intervals shorter than the reciprocal of the rates you set.

If the source video from the camera is compressed, we cannot start the stream after a gap until a "key-frame" comes along (that is a frame that contains enough information to generate the entire image). Some devices generate separate outputs for Preview (uncompressed) and Capture (compressed); it can happen that the preview display shows exactly the frame rate you request (1 frame per second, for example), but the frames written to the file may be at different intervals.

DO NOT set the slow frame rate or the fast rate to the actual rate that the camera produces. For example, if the camera generates 30 FPS and you set a slow rate of 30 FPS, you may reject every other frame due to slight timing irregularities. You will normally set the maximum frame rate to a rate much higher than the camera can generate (200 Hz, for example) and the slow rate to 1 Hz or lower.

Use Slow Frame Rate

This command can be found in the Settings menu, but is more conveniently activated and cancelled with the Use Slow Frame rate button on the toolbar.

Use this command to swap between the slow frame rate and the maximum frame rate set in the Set Slow Frame Rate dialog.

Use of the slow rate can be automated from Spike2 with the `MMRate()` script command and by activating the Spike2 Sampling configuration Mode tab option to set the Video FPS.

Key frames

Compressed video usually is formed from two types of frames: key frames and non-key frames.

Key frames — A key frame (sometimes called an I-frame) holds all the information necessary to display it. It does not rely on any knowledge about previous frames. Any compression of the image in the frame is based solely on the frame itself.

Non-key frames — These frames hold enough information to construct the image given that we already have the previous frame. These typically hold much less data than a key frame. If an image is essentially the same as the previous frame, these can be very small compared to a key frame.

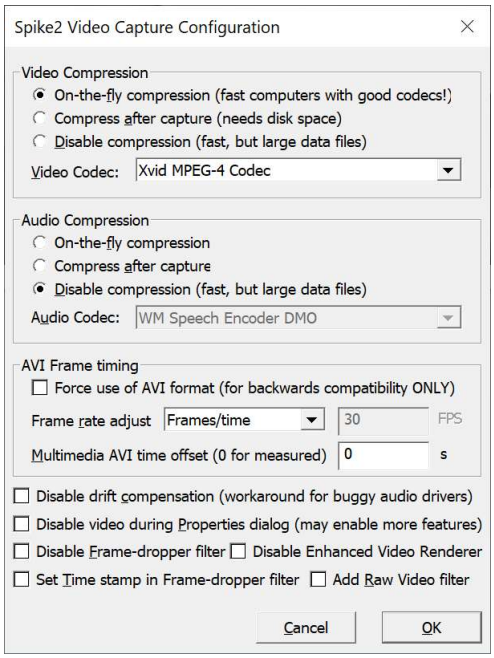
A image that is mainly an unchanging background in constant lighting (which is often the case in research recordings), can be highly compressed by having very few key frames. However, few key frames makes seeking to positions (for example, using the `MMPosition()` script command) in the video very slow. This is because to move to a time, the video must seek to the last key frame before the desired time, then decode forwards to generate the image at the desired time.

If you need random access to the video and your codec allows you to adjust key frame settings, you may wish to experiment with them to prevent excessive key frame intervals. If there is no explicit key frame setting you may need to experiment with the quality/data rate settings.

Some cameras that offer MPEG output compress the data within each frame so that all frames are key frames. If this is the case, and you have sufficient CPU power, you may prefer to use a codec that will compress across frames (for example XVid) to generate a significantly smaller file.

Configuration

The configuration command opens a menu in which you can choose a data compression strategy and, for AVI files, how to adjust the frame rate so the video stream stays synchronised to the times recorded in Spike2. It also has check boxes for working around problems.



Video and Audio compression settings

Compressing your data can reduce file size considerably, especially if the video data is relatively constant. Given that a raw video stream may be producing tens of MB a second, this is well worth having. The down side is that compressing data takes significant processing power. Ideally, you would like to compress the data in real time, but in practice this may not be possible. Compression is achieved by choosing a suitable codec. For both video and audio you can choose from:

On-the-fly compression

Modern computers should be able to cope with the extra processing load of compression data. However, on older hardware, or with multiple cameras this may be too slow to be useful unless you are running at low frame rates or at relatively low resolution. If your computer cannot keep up it will drop frames from the video stream. Real time audio compression is usually possible. However, if you choose to compress the video after sampling, you may prefer to defer audio compression also.

Compress after capture

If you select this option, the data is recorded raw to a temporary file. Make sure you have considered how large this file can get. Recording 640x480 images at 30 fps can easily use more than 1 GB a minute. When sampling has stopped and you choose to save the data file, the temporary file is added to a queue of files to be compressed. The compression task runs in the background, so that the program can continue capturing data. Spike2 will not be able to access the compressed files until the compression task has finished. Compressors may run more slowly than real time.

Disable compression

This option gives you the fastest sampling and access to your files, but at the cost of huge files (unless your source is already compressed - e.g. MPEG). Huge files are very tedious to copy. You can compress files separately using the `mp4comp` and `avicomp` programs.

Codec
This option lets you choose the codec that is used to compress your video or audio data stream. If you get an error message of the form, "*Sorry, a DirectShow error occurred. Action: Render the output stream. Change or disable video/audio compressor. The error encountered was: No combination of intermediate filters could be found to make the connection*", this means that you have chosen a codec that either required an unavailable input format or that generated an output format that was not compatible with the data stream. You must choose a different one.

MPEG (or H264) source
If your camera can generate MPEG or H264 format data, this is already compressed and you may find that your best strategy is to use this stream, as is, with no additional codec. If the camera does not have a separate Preview output the data will need decompressing for the Preview display, but decompressing is a far cheaper operation that compressing. It may be that a camera with MPEG output may compress frames independently of each other, so each frame is a key frame. If this is the case, higher compression is available at the cost of decompressing and recompressing the file.

Selecting a codec
You want to balance the file output size against the time taken to compress your video. If you have a choice of codecs it is worth running a few trials, sampling data for maybe a minute and comparing the size of the output file for each strategy. If you set a codec, all the replay machines need to have access to the same codec (or a compatible one). This is not a problem if you compress to a standard format (e.g. H264).

AVI Frame timing
These options apply to AVI file capture and are only used when you check the first option. We strongly recommend that you DO NOT USE THIS OPTION.

Force use of AVI format (for backwards compatibility ONLY)
If you check this box, we will format the output as an AVI file with the file extension `.avi`. This means that the only timing information available is the frame number. If this frame number is unreliable for any reason the output file will lose track of time. We suggest you should only use this if you have previously used the AVI format successfully and require backwards compatibility.

AVI files hold a list of frames (some of which can be empty of data if the frame was dropped) at a constant time interval. The frame rate is the number of frames per second and this must be set accurately if Spike2 is to be able to scroll a video display to correspond with a time in a data file. During sampling, the DirectShow system writes frames at a constant rate. If a frame is not supplied, an empty frame is written (and the previous frame is repeated on output). If too many frames are supplied, extra ones are dropped. This is all done based on the frame rate claimed by the camera. Ideally, you want to have the rate claimed by the camera to exactly match the rate it claims to generate.

Frame rate adjustment
This field can help you work around timing problems with AVI files. When capture of a video file ends, the frame rate is set in the AVI file header. To give us some flexibility, you can choose to control how the frame timing is calculated and recorded.

AVI video files run at a constant frame rate. By default, this rate is set by the rate that the source claims to generate. If frames are not supplied at this rate, the sequence of frame numbers will not be continuous, or frames will have to be thrown away if the camera rate exceeds the advertised rate. If frames are missing, an empty frame is written to the file for each missing frame. There are four options:

None	Accept the frame rate generated by Windows, which is the rate that the source hardware claims to generate.
Frames/time	When sampling ends, we set the frame rate to the number of frames in the generated AVI file divided by the sample time in seconds. If the video really does run at a constant rate, and no frames were lost, this should generate a video file that lines up in Spike2.
Fixed value	Set the frame rate to a user-defined value. If you are triggering the frames with a clock that is known synchronised to Spike2, such as a clock generated by the output sequence Clock output, then you will know the exact clock period.
Frames/time + fixed	This sets the frame rate used during capture to a user-defined value, then recalculates the actual value after sampling stops. This can be useful to minimise the number of 'missed' frames when you know exactly the camera rate and the rate it claims to run at is higher. This is

an experimental feature added at version 2.00 and can improve timing when used with codecs that cannot cope with missing frames.

Low light levels

Some cameras compensate for low light levels by reducing the Frames per Second (giving a longer exposure time per frame). This can cause the frame numbers in AVI files to be non-sequential, and DirectShow sees this as 'dropped' or 'missing' frames... If you use a video codec, such missing frames may cause timing problems. You may be able to turn off this exposure feature in the video settings (and live with a dim image), or you must light your subjects brightly.

A variable frame rate should not be a problem for `.mp4` files, which is one of the reasons that we prefer that format to AVI.

Multimedia AVI time offset (0 for measured)

There will almost always be a time offset between the time that Spike2 starts running and the time that the camera records its first frame. This can be positive (if the first frame is late) or negative (if the system buffers up a few frames, ready for you to say go).

If your device tends to have the same offset each time, you can set it here, and we save it in the AVI file. Spike2 will apply this offset when you review the file. The time is saved to millisecond precision.

We store the information in a supposedly unused section of the AVI file header. AVI files not generated by s2video will always have this value set to 0. You can also adjust this value in Spike2 to line up the video exactly, assuming that the frame rate is correct. We do not save this information in .mp4 files.

From S2Video 2.03 we measure the time offset of the first frame. If you set an offset of 0 (recommended), the AVI file offset will be set to the measured value.

How to check video timing

The simplest way to check the timing is to point the video camera at your keyboard and record yourself typing the alphabet during data capture. You can then compare the keyboard markers with the video.

If this is not practical, a more complicated (but more accurate) method is to use the following output sequencer script:

```
        set 1,1,0
go:     digout [00000000]
        delay s(5)-2
        digout [......11]
        delay s(5)-2,go
```

This causes the lower two digital output bits to change state every 5 seconds. This causes the front panel LEDs on Micro1401 and Power1401 devices to switch on and off. If you connect digital output 0 to event 0 input on the front panel, then sample the input as a Level event channel, you can record the precise timing of the digital output changing state. If you point your video camera at the front panel, you can now record a video that should show the front panel LEDs changing at exactly the times when the Level event channel changes. You are likely to see that the changes are late/early by a more or less constant time (remember that no times can be more accurate than the frame rate). If this delay stays constant over a range of video files, you can set this delay value for AVI files as the *First frame delay*. You can also set this delay in the Multimedia review window in Spike2.

Working around problems

Windows allows manufacturers to generate drivers for a huge range of devices. The interface specifications are complex and not all devices implement all aspects of the specifications perfectly. Because of this, we have added workarounds for problems that can occur. Do not set these options unless you need to.

Disable drift compensation

This is a workaround for some DirectShow audio device drivers that crash or generate errors when we try to match the multimedia time base to the Spike2 time. If you have problems of this nature, try checking this box; the multimedia time and Spike2 time will drift apart. You can read more about drift compensation here. If you set Frames/time in the AVI *Frame rate adjustment* there can sometimes be an advantage to checking this box as you are going to rescale the timebase using the frame rate. Most problems of this type are caused by recording audio. If you need accurately-timed audio you may do better to record an additional Spike2 waveform channel as audio.

Disable video (and audio) during Properties dialog
We normally leave the multimedia capture running when you open the device properties dialogs. If you have Preview enabled, this allows you to see the effect of any changes you make. However, some devices disable aspects of their Properties dialog if the device is being used. Check this box to stop running the device while the Properties dialogs are open. This option makes no difference to data capture.

Disable Frame-dropper filter
We may ask you to check this (at the cost of not supporting the Slow frame rate option, not time stamping frames, not measuring the frame time offset and not reporting on frame rates) when attempting to diagnose problems generating the framework to capture video data. **Do not set this option unless we specifically ask you to do it**.

Disable Enhanced Video Renderer
There are two Video Renderers we can use to display the Preview data stream. The Enhanced Video Renderer (EVR), which can take advantage of hardware acceleration (the Graphical Processing Unit or GPU in your display adaptor), is the default. However, you can disable it and use the default renderer that should be compatible with all systems. Prior to Windows 10, this one may have problems with multiple screens. However, the EVR is known to have problems with Windows Remote Desktop and from S2Video 2.03 onwards we do not use it (regardless of this setting) if we detect that S2Video is run as part of a Remote Desktop session.

April 2021: Microsoft seem to have improved the default Video Renderer in Windows 10. On my Windows 10 system it has no problems with multiple screens and reduces the CPU load compared to the EVR; you can measure this on your system by using the Task Manager (`Ctrl+Alt+Del` and select Task Manager). I now usually run with the EVR disabled and I may make this the default in future releases. On my machine, with a webcam and not connected to Spike2, S2Video with the default Video Renderer uses maybe 0.4% CPU and no GPU, the EVR uses around 1.2% CPU and some 5% GPU. You may get a different result on your machine.

Set Time stamp in Frame-dropper filter
Normally, frames are given a time stamp when they reach the PC, and this is the preferred method as it should have the least timing uncertainty. If you check this box, and have not disabled the Frame-dropper filter, we set time stamps on each frame based on the current DirectShow time. This relies on the video processing keeping up with the data, and will add some timing jitter to the frame times, but should fix timing problems in MP4 files and may do so for AVI files.

We have come across cameras that apply time stamps to the data in the camera. If the camera sets time stamps, DirectShow does not, so any drift between camera time and Spike2 time is not corrected. This may not matter for relatively short recordings. However, some cameras lie about the frame times, especially in low-light situations where the camera frame rate becomes less than the advertised rate. We know of one that reported frames times as if it was still generating frames at 30 FPS when it was actually generating at 25 FPS and this completely messed up the file timings.

Add Raw Video filter
We have had a couple of instances where cameras fail to connect and adding the ffdshow Raw Video filter has allowed the connection to work. The Raw Video filter accepts as input the following Video media types: `YV12`, `IYUV`, `YUY2`, `YUYV`, `UYVY`, `VYUY`, `RGB32`, `RGB24`, `RGB555` and `RGB565`. If your camera generates one of these formats and it will not connect, it is worth trying this filter to convert the input to a format acceptable to downstream filters (typically `YUY2` or `RGB24` or `RGB32`).

This filter is not part of Windows; you must download and install it. As I write this in April 2021 you can locate an installer for it by searching the web for "ffdshow download", which should point you at `SourceForge.net` (though other sources are available) from where you can use the Download button to get the installation program. You can accept the default installation settings, or just select the Raw Video filter. Once installed, check the Add Raw Video box to run the camera input through the Raw Video filter before splitting it into Capture and Preview streams. This feature did not exist before S2Video 2.08 (April 2021).

Storage of settings
S2Video saves its configuration settings in the registry in the key:

`HKEY_CURRENT_USER\Software\CED\Spike2 Video`

Please note that Video and Audio device Property Page settings and Video Capture and codec settings are saved by the devices themselves, not by our code. Any settings that your device does not choose to persist will require manual adjustment for each sampling session.

The Edit menu Copy Graph option copies the options as text to the clipboard.

Codecs

You can choose to compress the video and audio data so that it occupies less disk space. This is done using plug in modules called codecs. A codec is in two parts: a **co**der and a **dec**oder, hence codec. The coder changes the input signal in some way and the decoder reverses the process. Not all codecs are compressors; some may just change the format of the data to suit a particular purpose or to make the content secure. We are only interested in codecs that can make the data significantly smaller and that can do it fast enough to be useful.

Some codecs are lossless, that is, the decoded data exactly matches the encoded data; these tend to produce only small amounts of compression in real world data. Most codecs are lossy; the decoded result is not exactly the same as the original. However, they are designed to lose information in parts of the signal that we are less sensitive to, so often the result may look and sound close enough to the original not to matter.

Different codecs are also designed for different purposes. For example, the Microsoft RLE video codec is designed to compress images with areas of identical colour, so it does a very good job on cartoons but makes a terrible mess of real-world images. Some codecs rely on being able to do two (or more passes) through the data, so cannot be used in real-time where only one pass through the data is allowed. Some codecs need tuning for the task; the fact that a codec is not recommended may just mean that we failed to set it up correctly or it is not intended for this use.

When you installed your operating system a set of standard codecs was added to your system. When you installed your video camera or web cam, the installation software may have added further codecs. In the Configuration dialog, the Audio codec and Video codec fields let you select codecs that advertise themselves on your system as supporting audio or video. Not all of these are useful, or will even work with `s2video`.

Does your camera generate compressed data?

If your camera has the option of a compressed data stream (for example MJPG), you should experiment with using that first as it may generate a similar compression to a software codec, and saves the time in the computer of compressing the data. However, most cameras we have seen that generate MPEG output compress on a frame by frame basic and do not take advantage of the similarities between one frame and the next, resulting is quite high data rates.

Video codecs

To give you some idea of which video codecs you could consider, the following table gives number of kB per second required to record of an image of my hands typing for 30 seconds. The camera was set to produce 640x480 pixel images in colour at 30 frames per second in YUY2 format.

Unfortunately, none of the codecs that come with Windows 10 were of much use for this purpose. This was mainly because they generated very little compression.

We have located two open-source codecs that achieve worthwhile compression and of these, XVid seemed to be the better (however the other has many property page adjustments, and we may have missed the best settings). The table gives the performance I obtained on my quite old desktop machine:

Codec	kB/sec	Factor	Comments
None	17720	1	Uncompressed video. This is more than 1 GB a minute, which is unlikely to be useful except for very short recordings.
XVid	829	21	Available from https://www.xvid.com/. See the additional information, below
x264vfw	1572	11	Available from https://sourceforge.net/projects/x264vfw. Use the Start menu x264vfw Configure option and check the Zero Latency box to reduce time offsets. You may find this useful off-line.

With 320x240 images, the data rate could be 4 times lower. If you want HD images (1920x1080), the data rates may be 6 times higher. It is certainly worth spending a little time finding a combination of settings and camera resolution that give you acceptable performance and data rates.

XVid codec information

To get this codec, go to the XVid web site and click *Get the XVid Codec*. This leads to a page where you select between Windows and Linux. You want the Windows codec. When I did this in January 2020 I downloaded the file `Xvid-1.3.7-20191228.exe`, which holds version 1.3.7 of the codec. You must run this file and if you are offered a choice of formats, check all three. The XVid codec can be further configured by registry settings in `HKEY_CURRENT_USER\Software\GNU\XviD`. There is one value that is set by the installer and one that we attempt to set, but if this fails, you must add it manually.

Name	Default	Use
`Supported_4CC`	7	Bits set for supported formats (DIVX=1, 3IVX=2, MP4V=4). Set by the XVid installer.
`display_status`	1	0=suppress the status display, 1=show it. Not set by the XVid installer.

As installed, the codec displays a 'status' page when it is in use. This is useful for advanced users who wish to optimize the settings, but it causes s2video to hang up. It can be suppressed by setting the `display_status` value in the registry to 0. We attempt to set this value for you when we install Spike2 and each time s2video runs. If the value is not set correctly, the s2video Help menu About Spike2 Video... dialog will contain a warning message. To set this value manually, or to check the state, do the following:

1. Run RegEdit. In Windows 10 type `regedit` into the search window and you should be offered the Registry Editor Desktop application. Run it.
2. In the left-hand pane under Computer you will find `HKEY_CURRENT_USER`. Open this to find `Software`, then find `GNU` and finally `XviD`.
3. In the right-hand pane you will see at least two items `(Default)` and `Supported_4CC`. (If you have used the s2video properties dialog to adjust the codec settings there may be many other settings). If `display_status` is already present with a value of 0 you have nothing to do. Otherwise we need to add it and set it to 0.
4. Right-click in the right-hand pane and select `New->DWORD (32-bit) value`. A new item `New Value #1` will appear with the value 0. We need to change the name of this item.
5. Right-click the new item and select `Rename`. Set the name to `display_status` and you are done.
6. Close the registry editor. The change will take effect the next time you run s2video.

You can configure the codec from the Settings menu, see the Video compressor configuration... option for details.

Under Other Options you will find a check box for Display encoding status. If you check this, this will set the `display_status` field to 1, causing compression status to be displayed. Do NOT do this. This extra display will prevent you from opening this dialog again to cancel the option and you will need to reapply the option manually, as described above.

x264vfw codec information

You can get the codec by downloading it from the web site and run the installer. We use the x86 version for encoding. You can configure the codec from the Settings menu, Video compressor configuration... option. There are many suggestions about suitable settings to apply that I have found by searching the web. For real-time use I have seen the suggestion to change the following from the installed state:

Preset=`Ultrafast`, Fast Decode=checked, Zero Latency=checked, Ratefactor=`15.0`, Output mode=`VFW`, VFW FourCC=`H254`, Extra command line (for advanced users)=`--keyint 1`

However, you may find that the default state suits your needs better, particularly if you use this codec off-line. In my tests with my computer and webcam set to 640x480, and the above settings, I found this codec to be slower than the XVid codec; it could not keep up with the data rate and dropped frames.

Audio codecs

There is a wide range of audio codecs to choose from that come with Windows. Some are optimised for a particular task, such as compressing speech. The table shows the result of compressing myself speaking the numbers 1 to 14 in approximately 11 seconds using stereo 16-bit sound at 44.1 kHz (which was the minimum setting allowed by my sound card).

Codec	kB/sec	Factor	Comments

None	180	1	Uncompressed audio.
PCM	180	1	This is uncompressed audio in PCM format.
WM Speech Encoder DMO	8.3	22	Muffled, but understandable. Use only for speech!
GSM 6.10	15	12	OK
WMAudio Encoder DMO	47	3.8	OK. Artefacts (possibly caused by clock time adjustment)
IMA ADPCM	55	3.3	OK
Microsoft ADPCM	55	3.3	OK
CCITT A-Law	97	1.9	OK
CCITT u-Law	99	1.8	OK

The uncompressed file rate of 180 kB/s is the 44.1 kHz rate times 4 bytes per sample plus overhead. You can reduce the audio data rate significantly by using 8-bit mono sound input (if your sound hardware allows this), which gets you down to 45 kB/s or so without a codec. If you can tolerate a lower sampling rate, this will also reduce the rate. Remember that you are trading off disk space, which mainly means the time taken to write to disk, against the time taken to encode the data. For example, if your sound hardware allows you to sample mono 8-bit data at 20 kHz and you are also sampling video, you may decide that the audio is the least of your problems and it is not worth the time to compress it.

If the exact synchronization of sound with other spike channels is important, record the sound as a Spike2 Waveform channel (you will have to provide a suitable audio amplifier to generate an input signal of a reasonable amplitude). It is possible to switch recording of specific channels on and off to save disk space. From Spike2 [10.13] you can use the SoundCard Talker to pipe sound from a sound card directly to a data file (the synchronisation will not be as exact as recording with a microphone and amplifier through the 1401)..

Note that selecting multimedia audio may remove the ability of s2video to track the 1401 time. This is because DirectShow will likely use the clock from the audio source regardless of any clock source setting we make.

Disable drift compensation

Time in a Spike2 data file is measured based on a crystal-controlled clock inside your 1401 interface. Time in a multimedia file is based on either the computer system clock or on a clock in some multimedia hardware. The 1401 interfaces use crystals that are accurate to 50 parts per million over a 0 to 70 degrees centigrade range, this is a worst case gain or loss of 4.3 seconds a day; most units will be better than that. Assuming that the clock in your computer is of similar accuracy to the 1401 (and it could be much worse), there is a possibility of the multimedia recording and the 1401 disagreeing about time by as much as 8.6 seconds per 24 hours.

To avoid problems with time drift, Spike2 keeps track of the time differences between 1401 time and computer time and notifies the `s2video` application of any drift. This drift can then be compensated for to make sure that the times in the multimedia file remain locked to those in the Spike2 data file. The drift compensation is to the nearest millisecond. The compensation value is shown in the s2video text bar at the bottom of the window as the Drift value in milliseconds (omitted if it is 0).

If you record multimedia audio, DirectShow may decide to use the audio source clock as the timing clock and ignore our attempts to synchronize with Spike2.

We have come across audio device drivers that crash when we try to adjust the time. Sometimes you can fix this by getting the most up-to-date driver for the audio device, but if this is not possible you can work around this problem by disabling the time drift compensation in the Configuration dialog.

Even with the time drift compensation in place, there will be a fixed time shift between the Spike2 data and the multimedia file due to the time it takes the system to process the video and audio data. This fixed shift depends on the hardware in use and is usually less than 300 milliseconds.

Do not confuse the problem of time drift (which is usually small) with gross effects due to video frame rate or codec problems. I usually test time synchronisation by pointing the camera at my keyboard while I type and then check that the keypresses recorded in Spike2 match the video.

Help menu

This menu has 2 entries:

Index
This searches for the Spike2 online Help file and if it finds it, opens the Multimedia Recording page.

About Spike2 Video
This opens a dialog that reports the current version of the program.

It may also hold information about detected set up problems. At the time of writing, the only problem it will report is:

XVid codec Registry setup
If you have installed the XVid codec (recommended), we require a system Registry value to be set. We check that the value is correctly set during s2video start up and if it is not set, we attempt to set it. If this fails, we generate a warning message in the dialog. If you see this message and want to use the codec, you should look at the XVid codec documentation for instructions to set the required registry values manually.

Recording data

The `s2video` application acts as a slave to Spike2. If Spike2 is not running, you can configure the program, but it will not record data. Once a copy of Spike2 is located, `s2video` registers itself as a "listener" application. Spike2 will now tell it about all recording activities: opening a new temporary data file ready to record, starting recording, aborting or resetting sampling, stopping sampling, saving the temporary data file as a permanent file or throwing it away.

The `s2video` application does exactly the same tasks with its own multimedia file. There is one extra step required if you have chosen the Compress after Capture option in the configuration dialog. Instead of moving the temporary file to its final destination, it must be compressed. Compression can take a long time, and Spike2 might want to sample another file immediately. To avoid holding up Spike2, the temporary file is added to a list of files that need compression and the compression is run as a background task.

Compressing a file can take a long time, so while it happens `s2video` displays a dialog showing how far through the task the compressor has got for the current file. If there are multiple files waiting in the queue, the number of files left to process is also displayed.

In long recording sessions, most of the time you may need one video frame every few seconds, but there are short periods where you need the full frame rate. There is a Spike2 script command `MMRate()` that you can use to request a new frame rate. This command can only generate rates up to the maximum frame rate set for the camera.

When recording data under script control, you must use `FileSaveAs(name$,-1)` to save the data file and multimedia files. `FileSaveAs(name$,0)` exports the data file and does not save any multimedia files.

If you get video stream timing problems, check that any frame rates set in the Video Device Properties match those in the Video Capture dialog. Beware of Auto options that may set unexpectedly high rates; if this is the case, it may be better to set the rates manually. If the problems persist, set a lower frame rate or a smaller image size; the most common problem is that the system cannot process data fast enough.

Remote Desktop problem

We have customers who run Spike2 and S2Video remotely using the Microsoft Remote Desktop. This can be convenient when experiments may run for long periods or overnight. Note that when the remote desktop connects, it logs onto the remote machine and creates a desktop window (display context) that is independent of the screen of the remote machine.

Remote Desktop and S2Video

The original version of S2Video placed video on the screen using the Microsoft DirectShow Video Renderer. This is a software only component and came with the original version of DirectShow. It has some deficiencies, particularly when multiple monitors are used and does not take advantage of graphics cards with hardware acceleration. S2Video 2.00 (from March 2019) used the Enhanced Video Renderer, which works with multiple screens and takes advantage of hardware acceleration and is usually much faster; there is still the option to use the original renderer.

It turns out that there are problems using the Enhanced Video Renderer (EVR) with the remote desktop. When a Remote Desktop session terminates, it destroys the display context used to render the image and the EVR response to this is to abort, which ends the video replay and/or capture (you get a general purpose error code 0x80004005 in S2Video).

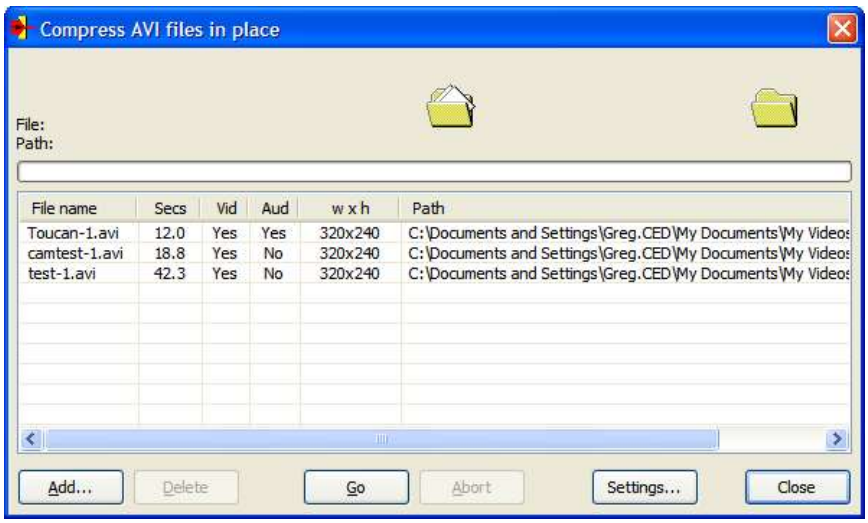
There is a workaround for this in S2Video, which is to disable the Enhanced Video Renderer. From S2Video version 2.03 we do this for you automatically (regardless of the Settings menu values). For versions 2.00 to 2.02 you must use the Settings menu Configuration... option and check the Disable Enhanced Video Renderer box. When the EVR is disabled, S2Video uses the software-only Video Renderer, which is slower, but which seems to survive the change between the remote desktop and the screen.

Remote Desktop and Spike2

Currently, there is no way to disable the EVR from Spike2, but this is less serious as the only effect is to stop multimedia replay working when you disconnect the remote desktop and this can be worked around by closing and opening the multimedia window when you reconnect the remote desktop.

The mp4comp and avicomp applications

The `mp4comp` and `avicomp` applications compress `.mp4` and `.avi` files generated by s2video that were not compressed during data capture. To run the application, double-click the file `mp4comp.exe` or `avicomp.exe` that is in the `video` folder in the same place as the Spike2 application.



avicomp main window

The application main window is in three sections. The upper section holds the name and path of a file that is being compressed, with a progress bar that indicates how much of the file has been processed. The middle section holds a list of files that are to be processed and also displays the length of the file in seconds, if it holds video and audio and the size of any video image. The bottom section holds a list of buttons that you can use to control the program. The buttons are:

Add

This button opens a standard file selection dialog in which you can select multiple `.mp4` or `.avi` files. To be added to the list, each file must contain a file with the correct format (the name need not have the `.mp4` or `.avi` extension, but all files created by `s2video` do have the correct extension for the internal type). The files must also be readable and writeable. You can always add a file, even when compressing. You cannot add a file that is

already in the list. If there is a problem adding a file a message box opens to give you a clue as to what went wrong.

Delete
This button is enabled when there is a selection in the list. The command removes the selected files from the list.

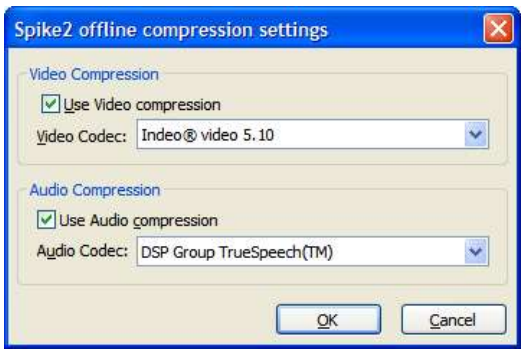
Go
Start compressing the files. Each file is removed from the list as it is compressed and the file name, path and compression progress are displayed in the upper part of the display. Files are compressed to a temporary file in the same folder, then if the compression is successful, the original file is deleted and the temporary file is renamed to the original name. If you want to preserve the original file, you must save a copy first.

Abort
Click this button to stop compressing the current file. Any temporary file is deleted and the file will be put back into the list. The compression process will stop.

Settings...
This command opens a dialog where you can choose the video and audio compression filters to apply to your file. You can change settings, even while processing files. The settings used will be those in effect when each file starts to be processed.

Close
This button is enabled when you are not compressing any data. It saves the list of file names in the window (up to a maximum of 100 files) in the registry, then closes the application. The list of files will be reloaded the next time you run the `avicomp` application.

Compression settings



Offline compression settings

From this dialog you can select the video and audio codec to apply to your data stream. The dialog will not let you exit unless you enable at least one compressor.

Errors on file add

Adding files to the list can fail for the following reasons:

Already open or a disk error
The file is open in another application or there was another problem that cause the attempt to open the file to fail.

Already in the list

This is self-explanatory. It is a waste of time to attempt to compress the same file twice and would (at best) result in a loss of quality as the file would be uncompressed, then compressed.

Not an AVI/MP4 file

Again, self-explanatory. The system failed to recognise the file as an AVI/MP4 container - probably because you have selected the wrong file.

An AVI/MP4 file but ran out of memory

The system recognised the file format, but ran out of memory trying to extract information about the contents. This is most unusual on a modern system and may mean that the file is corrupted.

An AVI/MP4 file but no driver

You may also see: (REGDB_E_CLASSNOTREG: missing registry entries?) This means that the file was recognised, but when Windows looked up the type of a data stream in the file it could not locate it in the system registry. You can get this error legitimately if the codecs needed for a file do not exist on the system. You can also get this when the registry entries in `[HKEY_CLASSES_ROOT\AVIFile]` (for example) are missing.

22: Technical support

Technical support

How to contact CED

Technical support

There is a bulletin board at https://www.ced.co.uk/phpBB3/ where you can post questions and get responses from CED and also from other Spike2 users. It also enables others to learn from your questions and the responses you get.

We also have software and hardware technical support desks that are available during UK working hours (and somewhat beyond). You can call us by telephone or send us a message by FAX, but our preferred medium is email as you can send example files and scripts as enclosures (even from within Spike2 with the File menu Send Mail option). If your question is not absolutely specific to your own circumstances, you might care to pose it on our Bulletin Board system so that others can benefit from the answer (you may even get a response from other CED users).

USA and Canada: 1 800 345 7794
World-wide: +44 1223 420186
email (software help): softhelp@ced.co.uk
email (hardware help): hardhelp@ced.co.uk
email (sales-related): sales@ced.co.uk
email (general): info@ced.co.uk
Bulletin Board: https://www.ced.co.uk/phpBB3/

You can also find up to date information, example scripts, and Spike2 down-loadable updates on our **Web site** at https://ced.co.uk

There are video tutorials for various aspects of Spike2 on our web site: https://ced.co.uk/tutorials/introduction

If you send us an enclosure, please have mercy… don't send us gigabytes. We have a medium speed link so a few megabytes is fine, but please contact us first if you need to send more than this. Most problems that need a file can be illustrated with a file section (use the File menu Export option to chop out a small piece). If you need to send us a large file, see here, or consider using a DVD and the mail for truly huge files.

Before you contact us please make sure you have checked that the information you need is not in the manuals or the on-line help. The on-line help is more up-to-date than the manuals and the Index is always worth a try; remember to try a few synonyms when searching for information.

Please tell us the serial number of your copy of Spike2 (this can be found in About Spike2 in the Help menu; to make it easy the Copy button will place the dialog information on the clipboard as text), the Spike2 version number, the operating system version, and any hardware-related information that might be relevant to the problem. If you have a crashing problem, please read the information about crash dumps (below).

If your problem is sampling related, please send us information about your 1401. The easiest way to do this is to run the Try1401 test program and use the File menu 1401 info... command and email us the results. A copy of the sampling configuration (.s2c file) will allow us to reproduce your sampling set up. Please remember that we will also need any other files that make up your sampling configuration (output sequences, arbitrary waveforms, scripts). If you have a complex arrangement of folders, scripts and sequences, please consider simplifying things - you will get a fix much more quickly if you send us the simplest arrangement that causes a problem.

Crash dump files

If Spike2 crashed out with a system error (General Protection fault, or the like), please send us all the information that the system gives you as often we can pinpoint the offending line in the code given the register dump. If the problem is easy to reproduce, send us all the steps needed to reproduce it and this may be all we need. However, if it only happens rarely and for no obvious cause having a crash dump file is essential if we are to have any chance of fixing it. We give a very high priority to finding and fixing crashing problems; we will generate a fix for the next release and may even be able to send you a fix or a work-around.

Enabling crash dumps from Vista onwards

We automatically enable crash dumps by setting the following registry keys within the registry at:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error Reporting\LocalDumps\sonview.exe
```

```
DumpCount REG_DWORD 10
DumpFolder REG_EXPAND_SZ C:\ProgramData\CED\Spike11\CrashDumps
DumpType REG_DWORD 1
```

In the event of a crash, the operating system will write a file to the path set by `DumpFolder`. We set things up so that up to 10 dump files can be saved, after which the oldest file will be overwritten. If you send us the dump file and tell us the version of Spike2 we should be able to identify where the problem happened (as long as it was within Spike2).

Crash on start up

If Spike2 crashes during start up, there is a section of the start up code beyond our control where it reads in control bar positions, and if the registry becomes corrupt for any reason, this can stop the program starting. You can delete some registry entries to fix this, see the Script language `Profile()` command to read more.

Sales related enquiries

Write, telephone, FAX or email us at:

Cambridge Electronic Design Ltd
Technical Centre
139 Cambridge Road
Milton
Cambridge
CB24 6AZ
UK

Telephone: +44 1223 420186
FAX: +44 1223 420488
email: sales@ced.co.uk

If you want price list or product information, visit our World Wide Web page: ced.co.uk

Spike2 revision history

The following changes, additions and fixes have been made to Spike2 in version 11. You can also get update information, downloads and script examples from our web site. The revision history lists items under the following headings:

New	Extra features in the program or improvements in operation.
Fixes	Repairs of broken features (bugs).
Changes	An alteration to Spike2 (which may also be listed in New or Fixes) that may require you to change existing scripts or program operations or changes how something works. We try to minimise these.

If there is a Known issues section, it lists problems for which no fix is yet available.

Revision 11.00

These are the new features, fixes and changes since version 10.21

New

1. The Sampling configuration can now hold a list of pre-set TextMark data items that can be applied during sampling.
2. The `SampleTextMark()` script command has new variants that can set and read pre-set TextMark data items and links to the sequencer and arbitrary waveform output. You can also lock these settings against user changes during sampling.

3. TextMark data created during sampling either manually or from a serial line input can trigger the output sequencer and arbitrary waveform output.
4. When sampling serial line data as a TextMark, you can now set how the code associated with each TextMark item is located in the data and converted.
5. There is a new `SampleTextMarkLink()` script command to control the link between on-line TextMark codes and the output sequencer and arbitrary waveform output.
6. In Overdraw 3D mode you can now use click and drag to change the screen arrangement and changes can be undone.
7. The Overdraw list dialog has been extended to allow user-defined trigger times in addition to Event times and changes can be undone.
8. The `ViewTrigger()` command with `mode%` set to 4 returns the current trigger time (position displayed as 0.0 on the x axis).
9. Triggered data sweeps that would partially extend before the start or after the end of a file are now drawn. Previously they were ignored.
10. Dragging the y axis now hides horizontal cursors during the drag operation. This is slightly faster and fixes a problem where a horizontal cursor with labels using a filled background did not redraw properly until the end of the drag.
11. Using the Cursor menu to Fetch a horizontal cursor now makes it visible if it was hidden (which matches the behaviour for vertical cursors).
12. The Text importer can now read files holding UTF-16LE format files in addition to files with 8-bit characters.
13. The Graphical Sequence editor displays a vertical blue line over the Digital and DAC traces for each Wait command as a visual indication that the time base is suspended at that point.
14. The Graphical Sequence editor action to wait for a digital input change now allows an optional time out.
15. The tooltip generated when hovering over a track in the graphical sequence editor now lists the number of items in the track.
16. The graphical sequence editor will not accept a blank track name.
17. The output sequencer supports new expression functions `Ceil()`, `Floor()`, `Round()` and `Trunc()`.
18. The output sequencer allows relative branches by replacing a label with +n or -n, where `n` is the number of steps to branch forwards or backwards.
19. The system toolbar now has a button that activates a TextMark dialog. When sampling, this opens the Create TextMark dialog, when reviewing data it opens the TextMark List/Edit dialog.
20. If there are multiple TextMark channels, the TextMark List menu command now opens the lowest-numbered selected channel or the lowest-numbered channel if none are selected.
21. The Pen Width dialog is enhanced with a spinner control that changes the pen width to the next value that makes a visible difference.
22. In a time view, the command `MaxTime(-1)` resets the maximum file time (stored in the file header) to the maximum time of any item stored in the disk file.
23. When drawing RealMark data as a waveform in Dots mode, the channel Primary colour now overrides the default of drawing using Marker colours.
24. Info windows can now display recent TextMark text or Marker channel codes.
25. The Analysis menu Process command Gate Settings dialog now remembers the last values applied in a session when opened for a new Process. Previously the values were always reset.
26. The File menu Load Configuration command has a new option to load the default configuration.
27. Active Horizontal cursors have two new modes: Median and Median + factor * size.
28. Cursor Regions and the `ChanMeasure(chan%, 22, ...)` script command can measure a channel amplitude using medians of differences from the channel median.
29. The first argument of the `ArrSort()` script command is now allowed to be an array of Objects if the `opt%` argument is -1 (randomize order).

30. In the Sampling configuration dialog Channels tab, the Copy as Text and Log options now include the Script tab settings.
31. The `SampleText()` script command can now control the output sequencer and trigger a PlayWave area.
32. With recording enabled, the Create TextMark dialog now records user commands.
33. The Sampling Configuration channel setup for TextMark data now includes the 'friendly name' of each COM port to make port selection easier.

Fixes (since version 10.21)

1. The Edit menu Copy command of a screen image to a Metafile or a bitmap did not show cursor labels with a filled background.
2. The graphical sequencer Wait for time command was drawn as if it was 1 sequencer tick in duration, but caused subsequent items in the generated sequence to be incorrectly timed. It now works as intended, effectively pausing the sequence for a user-defined time.
3. When viewing output sequencer code, it was possible for incorrect colourisation of command arguments; the command `"WAVEGO S,W"` would display the `'S'` in the colour set for sequencer functions as it was confused with the `s(expr)` sequencer function.
4. Opening the Signal conditioner settings dialog off-line after accessing them while sampling could show the wrong port number.
5. The TextMark List dialog did not update the displayed channel number if there were multiple TextMark channels and the user changed channel.
6. Measuring the Median or Size of a result view channel with the Cursor Regions dialog or the `ChanMeasure()` script command changed the channel data and did not return a result value.
7. When recording the `SampleTextMark()` script command, the serial line input `term$` string was truncated to a single character.
8. When recording the `SampleChannels()` command, the special channels position was always left at the default value.
9. If you sampled with a Micro4 with a Monitor revision less than 7, the output sequencer did not step.
10. In a user-defined dialog, clicking a spinner to change a large real number by a small increment could cause no change to the value.
11. When calculating the power in a band in a virtual channel, the low band edge was set incorrectly. The error was typically less than a quarter of the requested frequency resolution.

Changes

1. The channel list generated by the `DlgChan()` script function is now in the order of the displayed channels on screen. Previously, the list was sorted alphabetically, but as the entries started with the channel number, the order was a little strange (1-10, 100-109, 11, ...).
2. Marker and Marker-derived channels drawn in dots mode or WaveMark mode can override the marker code colour with the channel Secondary colour. Previously it was overridden with the channel Primary colour.

Frequently asked questions

The following questions have been asked several times over the history of Spike2:

I've set a sampling configuration. How do I sample data with Spike2?

Make sure you have a 1401 interface connected to your computer and switched on. You must also have installed the appropriate 1401 device driver (the Spike2 installation will have done this unless you told it not to).

Use the File menu New command and select a data file. This opens a new, empty data file. You can start sampling either by using the Sample menu Start sampling command, or clicking the Sample button in the Sampling control dialog box.

You can also start sampling from the Sampling configuration dialog by clicking the Run now button.

If you have several sampling configurations that you use frequently, load them into the Sample Bar. You can then load the sampling configurations with one mouse click and start sampling with a second or you can set the sample bar to start the configuration running immediately by right-clicking on the sample bar button and using the Immediate Start option.1401

Help! I accidentally failed to save a sampled data file.

Because your sampled data files (`.smr` or `.smrx`) may be very valuable and difficult to replace, we don't delete them at the end of sampling if you don't save them. Instead, we move them to the Recycle Bin; this is a quick operation, being just a change to the disk directory with no moving of file data. To recover from this situation do the following:

1. Open the recycle bin and locate the file. The easiest way to do this is to display the files in Details mode and list them in Date Deleted order.
2. Make a note of the Original Location field, then right-click the file and select Restore.
3. Go to the Original Location folder where you will find your file.
4. The file will not have any extension (`.smrx` or `.smr`). You must now rename the file to give it the correct extension, `.smrx` for a 64-bit file, `.smr` for an old 32-bit format file.
5. Open the recovered file.

If you suspect that there is any problem with the recovered file you may also wish to run S64Fix (for a `.smrx` file) or SonFix (for a `.smr` file) on it.

Abort and Reset

We do not move data files to the Recycle Bin if you use the Reset or Abort commands during sampling. Both these commands warn you that you are about to lose data and require confirmation, so it is quite difficult to lose data accidentally with them.

Spike2 beeps when I try to open a new file for sampling or exit

If you have a new data file, either sampling or about to sample, you cannot open another or exit from Spike2. If you try to open a new file or exit from the Spike2 menu, the program beeps to remind you that the data file is open. A script will give an error if you try to open a new file.

This problem usually happens when you have been working with a script that samples data. Such a script may open a data file “hidden”, so it is not obvious that there is a data file. Use the Window menu Show command to find any hidden windows or the Window menu Windows command to list all windows..

When I sample data, where is it saved during sampling?

Unless you have set a file name in the Automation Tab of the Sampling configuration dialog, Spike2 saves new data in a file called `Data` plus a sequence number in the path (folder) set by the Edit menu Preferences->Sampling command. If no path is set in the Preferences, the data goes to the folder set for data. Where this is exactly depends on the options you chose when installing the system. If you chose to install Spike2 into the system Program Files folder, the file will be typically found in:

```
c:\ProgramData\CED\Spike11\
```

One way to discover exactly where your files are being saved is to:

1. Sample a new file for a second or so, stop sampling but do NOT save the file.

2. Go to the Windows Start menu and type `regedit` in the search box. Windows should offer you regedit as a runnable program, so run it (despite the dire warnings from the operating system).
3. Open: HKEY_CURRENT_USER\Software\CED\Spike2\Recover
4. You should see that the key Last is set to the path to your temporary data file and Type is set to smr or smrx, depending on the file type.

Once you save the file, the registry entry is wiped. If Spike2 is interrupted during sampling by a power or system failure, the next time you run Spike2 it will detect that the last sampling session did not terminate as expected. You will be asked if you wish to attempt to recover the data.

We recommend that you set a path in the preferences so that you can control where temporary data is saved. The data is stored as a Spike2 data file without the file extension. When you close the file or use the Save command, you are prompted to supply a file name and the original file is renamed if the new name is on the same drive (volume), or copied if the new name is on a different drive. As files can be large and take a long time to copy, you should try to make sure the same drive is used for recording as for saving. On the other hand, if your saving drive is across a network you really do not want to record to it!

Do I have to use scripts to use Spike2?

No, you do not.

Many users of Spike2 use the standard data capture and built-in analysis routines to capture data, and then export data, pictures and values using the clipboard or data files and make measurements using the cursors.

You can perform quite complex automated analyses without any scripting if you use Active Cursors. This feature allows you to search for features and plot and tabulate measurements. With judicious use of Channel Processes, Memory Buffers and virtual channels, complex data can be extracted without any use of scripting.

However, Spike2 can do a great deal more than is exposed by the menu-driven interface and often a few lines of script can automate a tedious manual procedure. If you suspect that a script could help your work, but you do not have the time or inclination to learn how to write one, you might consider using our script writing service. Contact CED for more information.

How do I get started writing scripts?

Start by reading the Script manual as far as the alphabetical list of commands section. This should give you an overview of the script language. Next, look at the scripts supplied as part of Spike2. These will give you a flavour of how the language is used. You can look up any key word in a script by placing the text cursor in the keyword and then pressing the `F1` key.

We supply a copy of the Spike2 Training Course manual as a PDF with each copy of Spike2. This contains many script examples and explanations. It also has tutorials covering a range of common scripting tasks.

There are also scripts available from our web site (click here). These range from tutorial examples to full applications.

We also run training coursed in the UK and around the world. Contact CED for more information.

Which version of Windows is best for Spike2?

We strongly recommend that you use a version of Windows that it supported by Microsoft and keep up to date so that you are protected against attacks based on known system problems. In July 2024 this means using Windows 10 or 11.

Spike2 version 11 runs on Windows 10 onwards (we do not test on earlier versions; it very likely works on them, but we do not support this).

Spike2 version 10 runs on Windows 7 or later. We test in Windows 7 and Windows 10. We develop in Windows 10 and 11.

Spike2 version 9 runs on Windows 7 or later. We do all our development work in Windows 10.

Spike2 version 8 runs on Windows XP service pack 3 or later. It is available as a 64-bit program for 64-bit operating systems or a 32-bit build.

Spike2 version 7 runs under Windows XP service pack 2 or 3, Vista and Windows 7 and 8.

Spike2 version 6 will run on earlier versions of Windows but we would strongly advise you to upgrade to at least Windows XP.

Spike2 may work on Virtual machines and machines that emulate a Windows environment. However, we do not support this use, so if you have problems you must deal with them yourselves.

What is your support policy?

We actively develop the latest major version (currently 10 in February 2020), and fix bugs (where this is possible) in the latest major version and the previous major release. We may choose to make changes to older versions for reasons of backwards and forwards compatibility.

If you have an older major version and find a bug you should still report it as we may be able to suggest a workaround, and in some cases we may even fix it (especially if the bug persists into later versions).

We will always answer questions for licensed users of old versions of Spike2, and try to help you, regardless of the version. However, in some cases, the answer may be, "You need to upgrade".

I get error -544 when I try to sample. Why?

The most usual cause for error -544 is that you are logged on as a User with insufficient privilege to increase your Working Set. This error code usually means that the system has been unable to lock down memory needed for data acquisition. Spike2 tries to avoid this problem by requesting an increase in the *Minimum Working Set* of the program, but this request can fail if:

- The size requested is too large for the system to operate safely.
- The user does not have the right to change the allocation (see below).

The *About Spike2…* command in the Help menu shows the current Minimum and Maximum Working Set sizes. Spike2 attempts to set the Minimum Working Set to 1000kB and the Maximum Working Set to 8000kB, but you can override these values in the registry. As this is an advanced option with system-wide implications, we do not provide an easy way to do this from inside Spike2.

This error is unlikely to occur unless you are running Windows XP or your system administrator has decided to be super cautious.

What is the Working Set Size

You can think of the Working Set Size as the amount of physical memory allocated to a process (in this case the Spike2 program). A program may use many megabytes of memory space (logical memory), but only the memory needed for immediate use has to be present in physical memory, the rest can be saved on disk and loaded when needed. In general, the more physical memory a program can use, the faster it runs as it does not need to wait for information to be loaded from disk.

However, Windows is a co-operative environment, and if any process (program) grabs a lot of physical memory, there is less for others to use, and in extreme cases, the system may grind to a halt. As far as the operating system is concerned, it likes all processes to use the minimum possible physical memory. Each process (like the Spike2 program), is allocated a minimum working set size and a maximum working set size which the operating system uses as a guide to how much physical memory a process uses.

These settings used to be more important when systems did not have much physical memory. Nowadays (2013), most systems have 4 GB of memory (if you have a 64-bit operating systems you can have much more), so you may want to greatly increase the (conservative) values we set by default.

Minimum Working Set Size

This is the amount of physical memory that the operating system will always try to give a process, even when memory is running very low or your process is not running. If there are many processes running, the sum of the minimum working set sizes must always be less than the amount of physical memory in the system. The value

given a process by the system by default depends on the amount of physical memory in the system and can be as low as 120 kB, and lower in some circumstances. Spike2 needs to lock down around 300 kB of memory when it samples data, plus needing memory for some code and data. If the minimum working set size is too small, the operating system will refuse to lock the memory for sampling, resulting in error -544.

Maximum Working Set Size

The maximum working set size is the maximum amount of physical memory that the operating system normally lets your program have if memory is in short supply. If there is no other demand for memory, your process can have (much) more memory than this. The default values for this can be quite small, values of around 1 MB are common. If you run other applications at the same time as Spike2, you may be able to improve performance by setting a larger value.

What Spike2 tries to do

When Spike2 starts up, it tries to set the minimum working set size to 1000 kB (enough to sample data and run), and the maximum size to 8 MB. You can override these sizes in the system registry. You must create and set the registry keys as `DWORD` values:

```
HKEY_CURRENT_USER\Software\CED\Spike2\Win32\Minimum working set
HKEY_CURRENT_USER\Software\CED\Spike2\Win32\Maximum working set
```

to the required sizes in kB that you need. If you do not supply a key, the default size will be used. If the size you request is too large for the system to grant, the request will be ignored for both parameters. Spike2 will never set the sizes less than the sizes the system gives you by default. If you need to find the default system sizes, set the sizes in the registry to 0, run Spike2 and open the About Spike2… window.

You can set these keys with `RegEdit` which is part of the Windows operating system. Spike2 does not create these keys, but will use their values if they exist. From version 4.08 of Spike2 you can set them with the following script:

```
Profile("Win32","Minimum working set", 1200);  'set new minimum size as 1200 kB
Profile("Win32","Maximum working set", 32000); 'set new maximum size to around 32 MB
```

Any change you make will not take effect until you exit Spike and restart it.

If you run large scripts that use a lot of memory, you can improve performance by increasing the maximum size, but if you overdo it, you will reduce system performance by starving other processes (including system processes) of memory.

If Spike2 fails to change the sizes it writes a message to the Log view to warn you. A typical message is:

```
Working Set size change failed
Error 3: A required privilege is not held by the client.
```

What to do if you cannot set the size

If you get the log view error message and the *Help* menu *About Spike2* box does not report 1000, 8000 kB (or the sizes you have requested in the registry), then you probably do not have sufficient "rights" to change the allocation. The following describes how we altered the rights on our system. Your system administrator will know how to do this, but they might find the following will save them some time checking the system manuals.

We describe how to do this for the local machine: it may be that your machine is administered remotely in which case the general description is correct, but the details may be different.

Prior to Windows Vista, the right you need is *Increase scheduling priority* (called *SeIncreaseBasePriorityPrivilege* in low-level system documentation). This enabled a lot of other actions and from Vista onwards the less general *Increase a process working set* right was added (*SeIncreaseWorkingSetPrivilege*).

Rights are assigned to user Groups, so you must either add this right to the rights enjoyed by a Group you are a member of, or you must create a new Group with this right and become a member of it. In the descriptions below, steps to create a new group are labelled X. To extend the rights of an existing group, skip the steps labelled X and substitute the name of the existing Group for all mentions of *1401 Users*. The details vary with the operating system. Note that Spike2 version 11 is supported on Windows 10 onwards; the information about older operating systems is for interest only:

Windows XP

1. When logged on as Administrator, open the *Control panel* and then the *Administrative tools* folder.

X1. Run *Computer management*, select the entry *System tools:Local users and groups*.

X2. Select the *Groups* folder, use the *Action* menu to add a new local group called *1401 Users*.

X3. Use *Add…* to make the appropriate users into members of the new group.

X4. Exit from *Computer management*

2. Run *Local Security Policy*, select the entry *Local policies*, sub-entry *User rights assignment*.
3. Select Increase scheduling priority, use *Action* : Security... in NT 2000, use *Action* : Properties in XP.
4. Add *1401 Users* to the list of groups with that user right.
5. Exit from Local Security Policy.

Vista, 7, 8, 10

A new right was added at Vista, "Increase a process working set", and this is normally set for all users, so from Vista onwards you should not need to make any changes for this. The instructions below are ONLY needed if your system administrator has (most unusually) decided that no users should have this right and has specifically disabled it. If you follow these instructions you will only enable it for members of the *1401 Users* group.

1. When logged on as Administrator, open the *Control panel*, (switch to *Classic View* in Vista) and then open the *Administrative tools* folder. In Windows 10 type *Administrative Tools* into the Search panel and then run the tools (you can also use this to find and run *Computer management* and *Local Security Policy*).

X1. Run *Computer management*, select the entry *System tools:Local users and groups*.

X2. Select the *Groups* folder, use the *Action* menu to add a new local group called *1401 Users*.

X3. Use *Add...* to make the appropriate users into members of the new group.

X4. Exit from *Computer management*

2. Run *Local Security Policy*, select the entry *Local policies*, sub-entry *User rights assignment*.
3. Select *Increase a process working set*, use Action : *Properties.*
4. If *Add User or Group* is present, click it, else step 5.
5. If *1401 Users* is not visible in the list, click on *Object types...*, check the *Groups* box then click *OK*.
6. Add *1401 Users* to the list of groups with that user right.
7. Exit from *Local Security Policy*.

Next time you log on as a member of *1401 Users*, you should find that *Help:About Spike2* shows the expected memory available and that the error -544 no longer occurs.

What to do if it still doesn't work

Don't panic! It works for everyone else, so there is a logical reason for failure. Please check that you have the latest service pack for your OS installed as out-of-date system software has been the cause of this in the past. If you could not follow the above instructions, make sure that you are logged in with Administrator rights when you attempt to do it. If you are not, you will need to find your system administrator and ask them to do this for you. To run Spike2 version 11 you should be using Windows 19 or later, so someone has removed your *Increase a process working set* right (probably in an attempt to protect you from malicious programs). If you do not get this right restored, there is nothing we can do to work around this.

Why can't I make Process in Gated mode work?

The Gated by events radio button in the on-line Process dialog has **nothing at all** to do with a stimulus channel used for PSTH analysis or event correlations or waveform averages. Many users who select the Gated by events radio button should have selected the Automatic button.

The purpose of the Process dialog is to select an area of data for analysis, defined by a start and end time. The fact that the analysis might use a stimulus channel, so only a small part of the data in the time range is used, is not a consideration for the Process dialog. If your analysis uses a stimulus channel (e.g. PSTH or Waveform average), then only the stimuli that lie in the time range set by Process are used.

Normally, you set a process region explicitly as a start and stop time. However, for on-line use we allow result views to update in three ways: Manually, Automatically, and Gated by events. In Gated by events mode,

the data area to be processed is not defined in terms of a start and stop time. Instead, the data area is defined with respect to a Gate signal. Each time a gate is detected, the analysis set for the result view is performed on the data starting at the pre-gate time before the trigger for a time set by the Sweep length field. If your analysis also uses a stimulus channel, then the stimuli that lie in this time range are used.

A common mistake is to assume that the gate signal is the same as the stimulus. At best this makes the analysis rather inefficient, at worst it can stop any analysis happening at all (for example if the pre-gate time is less than the pre-stimulus time set for the analysis).

Sample rate limits and interface type

The maximum possible data transfer rate is determined by your 1401 type. However, this can be further limited by the way that your 1401 is connected to the host computer. There are four different possible connection methods, of which only three are supported for sampling in Spike2:

1. ISA interface. This is now obsolete and you are only likely to find it with a very old computer. If you have a Power1 or Micro2 with an ISA interface you will not be able to sample with it in Spike2. However, both of these devices can use USB, so you should use that instead. For interest only, the maximum transfer rate from the 1401 to the host with this interface was around 250 kB per second.
2. PCI interface. This is increasingly rare, but your Power1 or Micro2 might be using it. Modern computers do not have PCI slots any more so you should use the USB interface instead. The maximum transfer rate with the PCI interface was around 1 MB per second.
3. USB1 interface. Some of the very early Power1 units had a USB 1 interface. This limits the maximum transfer rate to around 1 MB per second.
4. USB2 interface. All the supported 1401s (Power1-3 and Micro2-4) apart from a few very early Power1 devices have a USB 2 interface. The data transfer rate is much higher in the range 10 to 48 MB per second, depending on the 1401 type. However, if you plug your 1401 with USB 2 into a USB 1 port on your computer (older computers often have both types of USB port), your transfer rate will fall back to USB 1 speeds.

If your 1401 supports USB 2.0 and you have a USB 2.0 port available on your PC, then USB is the fastest interface and you should use that.

If your 1401 supports USB 1, you have a choice of interface as all 1401s with USB 1 can also be used with the CED PCI card. The PCI card is a little faster than the USB 1 interface, but this is unlikely to make much difference unless you are getting towards the limits of what can be achieved on your 1401.

Ctrl+Shift+0 does not work

Microsoft have grabbed the `Ctrl+Shift+0` key combination for IME language control in Vista and Windows 7 onwards. We make use of this combination to scroll the x axis to locate a vertical cursor in a data view and to set all visible codes to 00 in the clustering dialog. If you have multiple languages installed, `Ctrl+Shift+1` etc. may also fail.

Windows Vista and Windows 7

You can get instructions to defeat this at this Microsoft link: http://support.microsoft.com/kb/967893.

The Microsoft instructions are (in January 2012):

1. Click Start, and then click Control Panel.
2. Double-click Regional and Language Options.
3. Click Keyboards and Languages, and then click Change keyboards.
4. Click Advanced Key Settings, and select Between input languages.
5. Click Change Key Sequence.
6. For Switch Keyboard Layout, select Not Assigned.
7. Click OK to close each dialog box.

Windows 8

In Windows 8 the `Ctrl+Shift` combination is used by default to switch the keyboard layout. To disable this:

1. Go to Settings->Control panel->Language->Advanced settings.
2. Click Change Language bar hot keys.
3. In Text Services and Input Languages in the Advanced Key Settings tab select Between Input Languages, then click the Change Key Sequence button and disable the key sequence. It should display as (None).

Windows 10

The following instructions worked for me in Windows 10 Professional 20 H2 English UK. I'm afraid there is no guarantee that these instructions will stay the same in future Windows releases.

1. Open the Start menu Settings (the gear cog icon).
2. Chose Time & Languge.
3. Choose Language in the left-hand panel.
4. Choose Keyboard to open the Keyboard settings page.
5. In Switching input methods check Use the desktop language bar when it is available.
6. Click on Input language hot keys.
7. Choose the Advanced Key Settings tab and select Between Input Languages and then click the Change Key Sequence button.
8. Change Switch Input Language and Switch Keyboard Layout to Not Assigned. You can leave one of these on Left Alt+Shift if you wish.

I'm having trouble getting FilePathSet to work in Windows. Why?

The most common reason for his problem is that you have forgotten that the \ character is treated specially by the script compiler when it is part of a literal string. Backslash is an escape character and is considered in combination with the following character. Not all character pairs generate acceptable “escape” sequences. You must use \\ to produce a single \ in the output. For example, to set the path to `c:\Spike2\data` you would need: `FilePathSet("c:\\Spike2\\data")`. A simple way to avoid this problem is to use the slash character / in place of the backslash, thus: `FilePathSet("c:/Spike2/data");`

Spike2 marked as Not Responding

If a top level window (such as the Spike2 application) does not respond to Windows messages for about 5 seconds, it may be marked as Not Responding. This can happen when Spike2 is processing a lot of data, if a script is running without releasing any idle time, or if a drawing operation takes a very long time.

Since Windows XP, the operating system replaces the Not Responding window with a *ghost window* that copies the attributes of the real window and adds "Not Responding" to the title bar. This is done so that you can move the window and attempt to close it (if you think it is really dead). If the window subsequently comes back to life, the original window is restored and all is well.

Never ending drawing

In Windows 7, a ghost window created in the middle of a screen update seems capable of causing the system to draw forever.

Unfortunately, it seems that if the application is in the middle of a Paint operation (updating an area of the application window that is no longer valid) when this occurs, the application is disconnected from the DC (Device Context - a logical representation of the screen that is being updated) it is painting to, and when the paint operation completes, the original DC does not think it has been painted. If the application then starts to respond to messages, Windows restores the original window, but immediately asks for another Paint operation for the same screen area, which puts us into a loop.

In Spike2 from version 7.03 onwards, if we detect that a time view drawing operation is taking more than a couple of seconds, we stop it early and mark the as yet unpainted area of the channel as invalid so that it gets painted in a subsequent Paint operation. However, this cannot be done if the channel is part of a group of overdrawn channels or if the window is in triggered overdraw mode or for a channel in OverDraw WM mode.

If your very slow screen update starts to loop approximately every 5 seconds, the only way out is to use the `Ctrl+Break` key combination to break out of the drawing operation.

How to make underlines requested with & appear

Since Windows 7, Microsoft have made the underlines requested in prompts and button text invisible unless you press `Alt` to cause them to appear. This change has also caused the underlines that used to work in `Toolbar()` and `Interact()` script commands stop working in Windows 7. You can fix both of these features by permanently enabling the underlines, as follows:

Windows 7
Go to the Control Panel and select Ease of Access Centre. From this, select Make the keyboard easier to use. Finally, check the box labelled: Underline keyboard shortcuts and access keys.

Windows 10
Open the windows Settings dialog (in the Start menu select the cog wheel icon). Click on Ease of Access. In the Interaction section click on Keyboard. Scroll down to Change how keyboard shortcuts work and turn the Underline access keys when available setting On.

Index

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- K -

- L -

- M -

- N -

- O -

- P -

- Q -

- R -

- S -

- T -

- U -

- V -

- W -

- X -

- Y -

- Z -