CAMBRIDGE ELECTRONIC DESIGN LIMITED

eNEWSLETTER

#6
September 2020

Contents

Welcome

Thank you for downloading our September newsletter. With the beginning of the new academic year, we would like to welcome everyone back from their holidays; we hope you had an enjoyable time. In this issue we continue to bring you more helpful scripts, answer questions for Spike2 and Signal, and continue our Scripters Corner feature for newer script writers.

Many of you may already be aware of the cancellation of Neuroscience 2020, and we are sad to be missing the event for the first time in 35 years. It is not surprising with the ongoing measures to contain the pandemic, and most other events of 2020 had already been cancelled or postponed. Whilst we will not be able to see you in person, CED is still here for all your data acquisition and analysis needs. Should you require any help or wish to discuss your system, email us at info@ced.co.uk and we can arrange for a video call via Skype/Zoom/Teams. As always, should you have any suggestions, questions or queries for the newsletter please get in touch Marjorie@ced.co.uk.

Training Days

Due to the ongoing COVID-19 pandemic, all current training events have been put on hold. We do however offer remote training sessions by Skype/Zoom/Teams either one-to-one or with groups.

Join us and learn how to make the best use of Spike2 and Signal to save hours of repetitive analysis. Our remote sessions are free to arrange and are suitable for both existing and prospective users of our data acquisition and analysis systems. If you would like to schedule a session, please get in touch.

If you are interested in hosting a training event in your local area once social distancing measures have been eased, please get in touch: Marjorie@ced.co.uk.

If you see these buttons in our newsletters, it means a file or script relating to the section is available to download:



Latest versions of Spike2 and Signal

Spike2	Released	Signal	Released
Version 10.06	07/2020	Version 7.05a	02/2020
Version 9.11	06/2020	Version 6.05b	10/2019
Version 8.19a	11/2019	Version 5.12a	02/2018
Demo	07/2020	Demo	02/2020

Back to contents

Future meetings and events

Brainbox Initiative 2020
Virtual Conference
September 22nd – 25th 2020

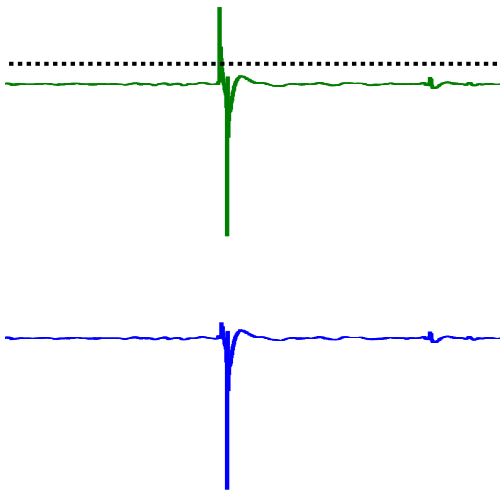
See Signal and the CED1401 in action at Brainbox Initiative 2020. Join Brainbox on the 23rd September for a full day of TMS talks, and a short demonstration of the DuoMAG TMS system. Our meeting calendar will be updated each time we receive news of postponements or measures implemented for safe meetings; the full calendar is located on our website.

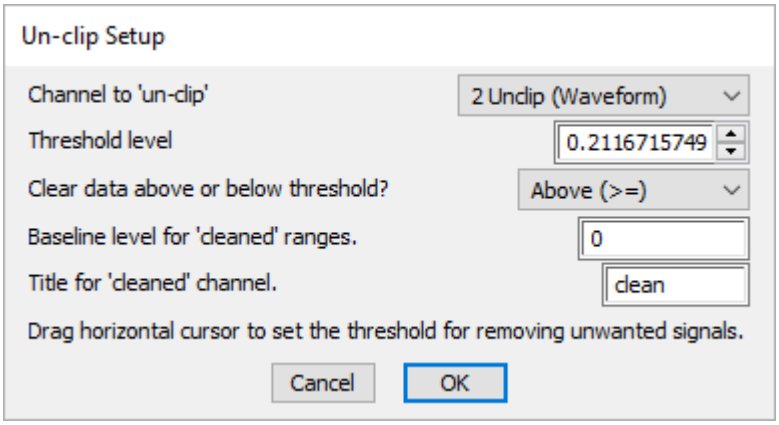
Back to contents

Script Spotlight – Unclip.s2s

Many types of analysis involve detecting events that cross a threshold level. This threshold is often based on the mean level plus a multiple of the standard deviation. Measuring the mean and standard deviation is tricky in cases where the data is affected by artifacts that cause the waveform repeatedly to hit the limits of the recording range, i.e., 'clipping'. This is a problem that is often encountered when using telemetry systems to record EEG or ECG. The unclip.s2s script, available on our website, provides a method to identify artifacts that exceed or fall below a user-defined threshold and makes a RealWave copy of the affected channel with the artifacts replaced by a user-defined level. It should reduce the distortion of mean and standard deviation measurements and make the data easier on the eye by removing glitches.

To operate the script, close all time views except the one you want to clean up and then run the script. A dialog opens to choose the channel for amending, set a threshold level for artifact detection (either by entering a level or dragging the horizontal cursor), and specify whether to replace waveforms that are above or below the threshold. When you click on *OK*, a copy of the channel is generated with a title of your choice and the artifacts replaced with zeroes. If the data has artifacts of both polarities the cleaned channel may be processed a second time with a new threshold and the remaining polarity.





Back to contents

How do I use the spectral functions in virtual channels?

Virtual channels include spectral functions which are used to show how a feature of the power spectrum of an input channel changes over time. The available functions include:

- Power in frequency bands – Calculate the power in a defined band and optionally divide the power by the power in a second band.
- Spectral edge – Calculate the frequency at which a percentage of the power in a band lies below.
- Mean frequency – Calculate the mean frequency, optionally within a defined band.
- Dominant frequency – Calculate the frequency region with the maximum power, optionally within a defined band.

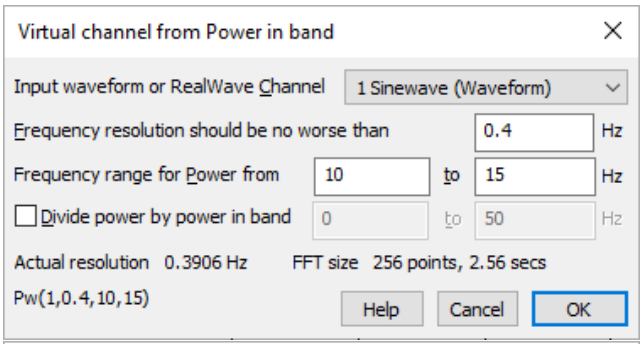
Each function calculates the power spectrum of an input channel and are particularly useful in EEG and EMG analysis. The power spectrum is calculated using the Fast Fourier Transform (FFT), applied many times to span the data. These functions take more time to calculate than other virtual expressions; saving the virtual channel as a real channel will ensure the calculation is only performed once.

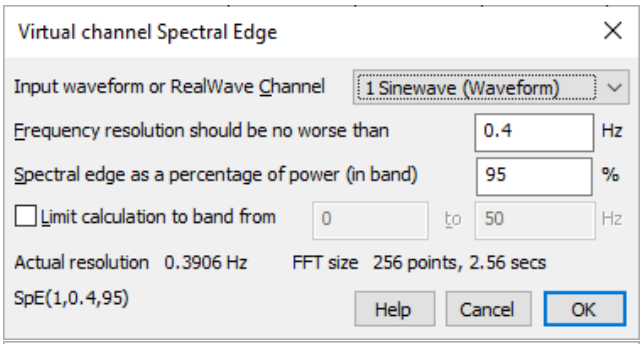
Please note that spectral analysis is a compromise between temporal and frequency resolution. If dF is the desired frequency resolution in Hz and dT is the desired time resolution in seconds, these are related by:

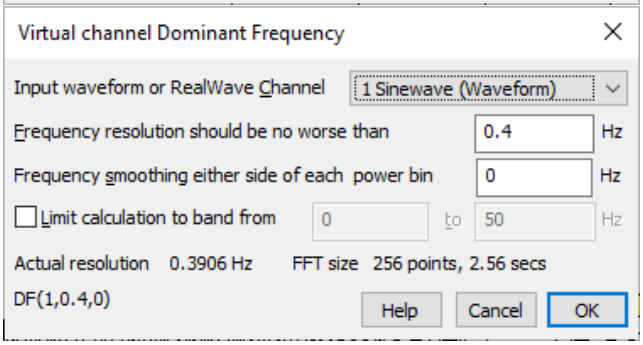
dF * dT = 1

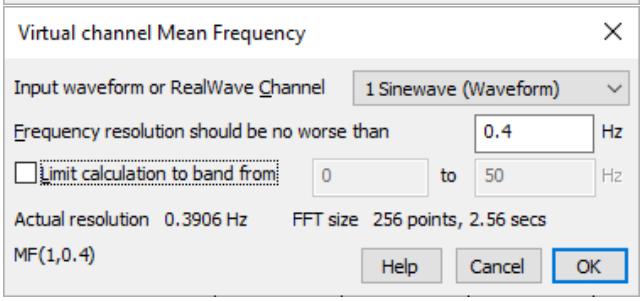
That is, if you want a 0.1 Hz frequency resolution, the result is based on at least 10 seconds of data. The FFT we use is limited to powers of 2 points, which constrains the available resolutions.

All the above functions are applied from the *Spectral functions* option in the *Expressions* menu of the *Virtual channel expression dialog*. Selecting a spectral function opens a new dialog. These dialogs help you build the power spectrum, with the resulting expression displayed in the bottom left corner of the dialog. There are several common inputs between the dialogs:









- **Input waveform or RealWave channel**

  The maximum frequency for use in bands is half the sample rate of this channel. Missing data values in the input channel are treated as zero.

- **Frequency resolution** of the output

  Set the resolution to 0 for an FFT size of 256 points. The actual resolution and FFT size are displayed at the bottom of the dialog.

- **Frequency range**

  The low and high edge of a frequency range may be defined in Hz. This is optional for the spectral edge, mean frequency, and dominant frequency functions, enabled by ticking the relevant box.

In addition to the common inputs, there are specific commands for three of the spectral functions:

- **Power in frequency bands**

  There is an option to divide the power in the frequency band (f to l Hz) by the power in another defined frequency band (r to s Hz). If the high edge s is omitted, the band extends from the low edge r to half the sample rate of the input channel.

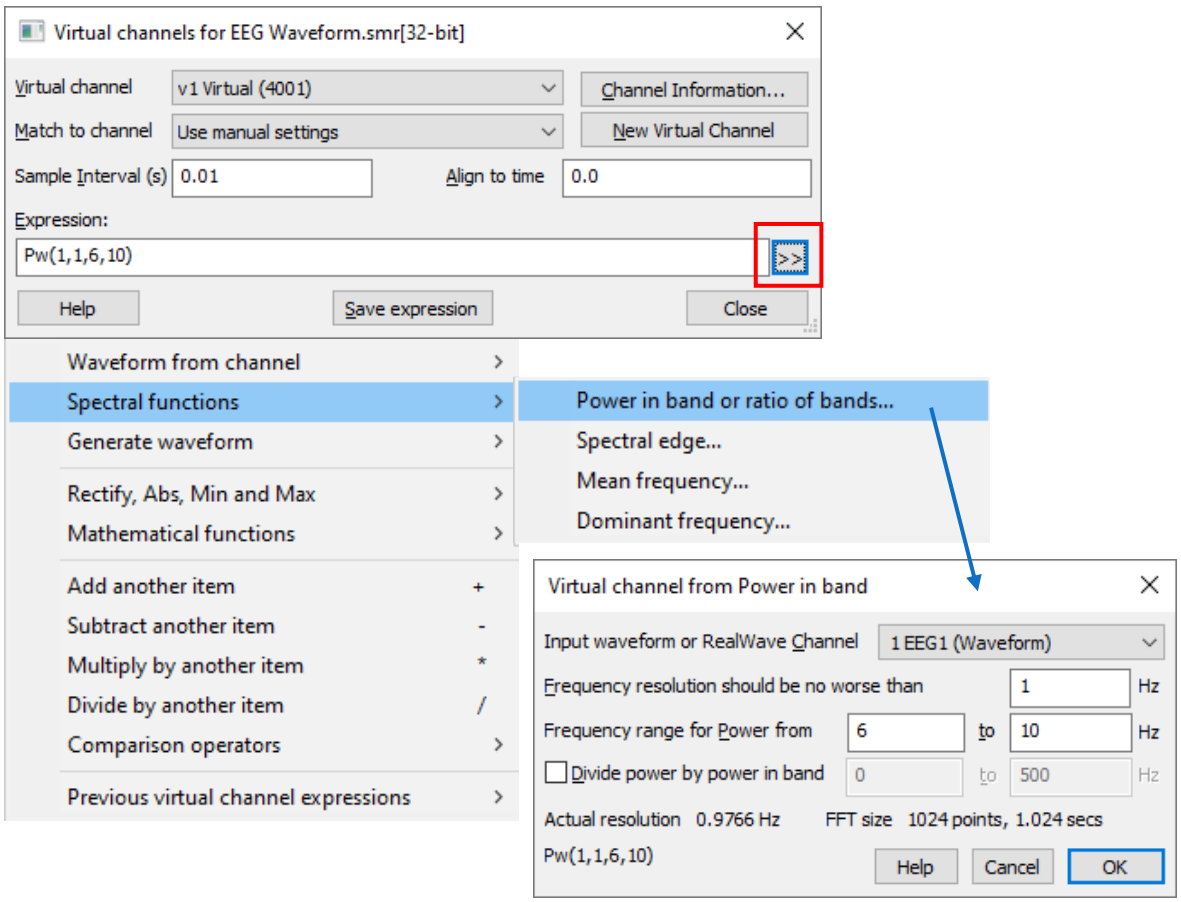
- **Spectral Edge**

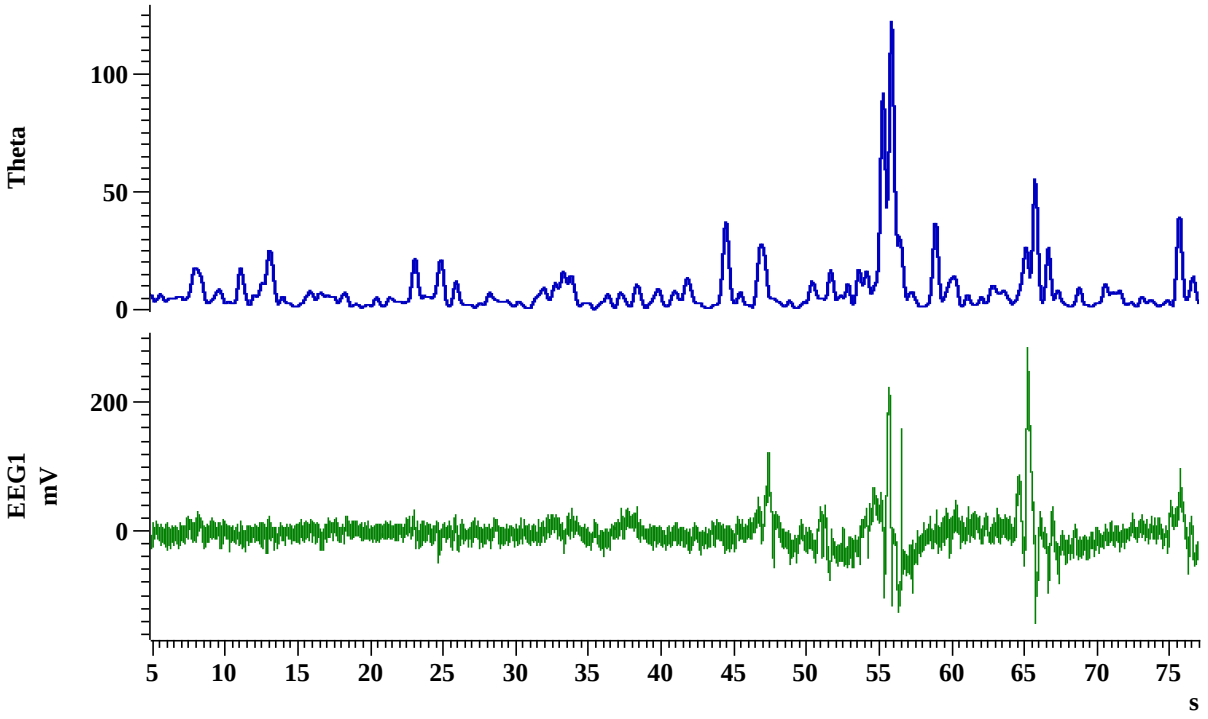
  The spectral edge as a percentage of power must also be provided.

- **Dominant frequency**

  Define the distance in Hz to smooth the data either side of each point. Enter 0 for no smoothing.

The example below calculates the spectral power in the theta band of an EEG input channel.





Use the *Analysis menu > Save channel* command to save virtual channel as a permanent channel.

Back to contents

Scripts: Spike2

We have recently improved our Auditory Stimulation and recording script package, available from our website. The download contains two scripts: AudioStimRecSU v705b.s2s and SoundCalSU v702a.s2s. These scripts allow you to generate acoustic stimuli via one or two speakers, or to generate vibratory stimuli using a mini shaker. Single unit responses to these stimuli can then be recorded and analysed on-line. The scripts operate with the CED 3505 devices, which are our are single-channel programmable attenuators offering main attenuations from 0dB to over 100dB in 2.5, 3 or 5dB steps. You can read more about the CED 3505 here.

Since the output of speakers/mini shakers are not flat over their entire frequency range, calibration curves are needed to derive the stimulus intensity for each frequency, given an input signal of constant amplitude. The SoundCalSUx script measures the response intensities over a wide frequency range and generate calibration tables for acoustic and/or vibrational stimulators. The AudioStimRecSU script then uses these calibration tables to generate stimuli of known intensity.

The package offers you up to two channels of auditory stimulation, with generation of tone-pips, tone-bursts and noise-bursts generated internally by the CED1401 interface. Like the Create wave for output.s2s script mentioned in our last newsletter, these stimuli have a trapezoid envelope, a linear ramp rising phase followed by a plateau and then a linear fall. The rising and falling ramp duration is set independently to allow generation of asymmetric envelopes. You can select a wide range of ramp and plateau durations, stimulus repetition rates and number of stimuli to present. The envelope may be filled with noise (pass band DC – 1kHz) or sinusoids in the frequency range from 1Hz to 10kHz.

The package has further functions to trigger an external stimulus generator and record externally generated TTL pulses for synchronizing on-line phase –response analysis, as well as play back pre-recorded waveforms as auditory stimuli. The stimulus intensity is adjustable by means of the CED3505 programmable attenuator. CED1401 generated stimuli are adjusted so that equal sound intensities can be achieved over the full range of carrier frequencies by compensating for the frequency characteristic of the speaker.

For analyses, intra- or extracellularly recorded auditory responses are displayed on-screen along with a stimulus monitor. Spikes are detected on-line by means of a virtual window discriminator. Unit activity is displayed as an event channel and optionally as instantaneous spike frequency. The following analyses may be performed on-line:

- Post-stimulus time histogram
- Inter-spike interval histogram
- Phase locking, i.e., a phase histogram shows the phase lag of each spike with respect to internally or externally generated synchronization pulses. Results are shown on a polar plot. Mean vector, vector length (r) and the Z statistic indicating significance of the distribution (relative to uniformity) are displayed on-line.
- Automatic auditory threshold determination using an automated sequence of stimuli.
- Threshold tuning curve. Specify the frequency range and intensity range of stimuli to use and the order of presentation, the responses are displayed as Intensity-Response functions for each frequency and are used to generate the threshold tuning curve.

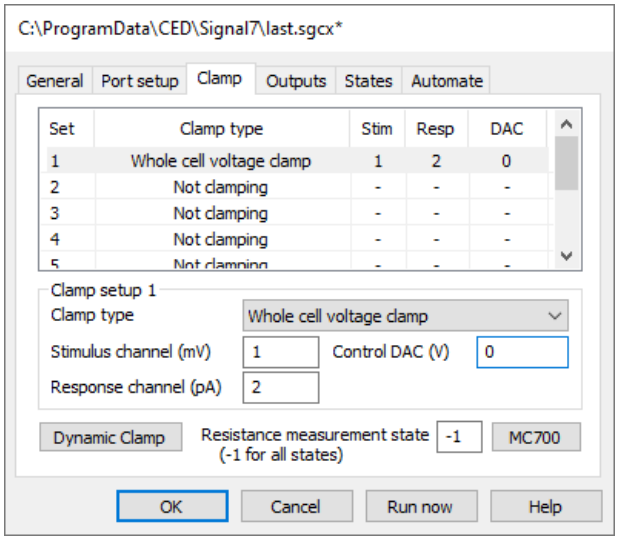
Full instructions on how to operate the scripts, as well as connect and calibrate the equipment is included with the download. We trust these scripts prove useful to you, and feel free to email us with any comments or questions.

Back to contents

Signal

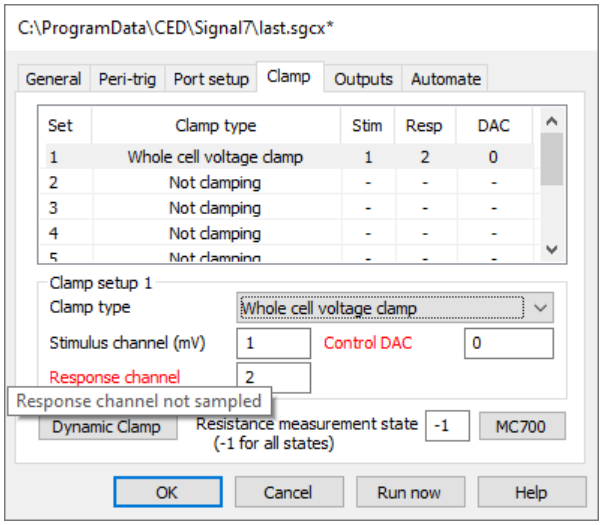
How do I set up Signal for patch clamp recording?

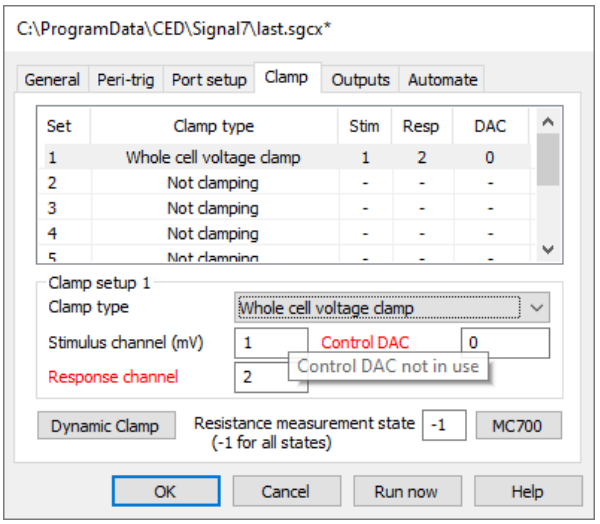
In our previous newsletter, we mentioned the necessity of defining your clamping sets when preparing Signal for patch clamping. Here we provide more detail about these sets and how they function. You define your clamping sets in the Clamp tab of the sampling configuration:



A clamping set is a pair of interlinked data file channels that hold the current and voltage data across a cell membrane, plus a control DAC that is used to control the membrane potential (for voltage clamp experiments) or current (for current clamp). It is necessary for the clamping sets to be defined in the sampling configuration so that the stimulus and response channels are known and the membrane analysis can find the correct data, this also allows Signal to check the units for these channels so that current and voltage values can be scaled correctly to amps and volts. Defining the DAC used to control the membrane current or voltage allows Signal to manipulate the holding potential and stimulus pulses and again to check that the DAC units are correct for the clamping mode.

The top of the Clamp tab shows a list of all eight clamp sets. Click on a set to select it, the lower area allows you to view and edit the parameters of the selected set. Any issues with your clamping set will be highlighted in red, if you hover the mouse pointer over this prompt a tooltip will show that gives more information about what is wrong with your selection:





- The Clamp type for a set can be Not clamping, to disable use of that clamp set, or Whole-cell or Single channel for either voltage clamp or current clamp. If all the sets are Not clamping, then the online clamping support will be disabled for this sampling configuration.

- Specify the Stimulus channel and Response channel, this defines the ADC port and corresponding data file channels to hold the data. For voltage-clamp experiments the stimulus channel is voltage and the response channel is current, for current-clamp it is the other way around. If the channel specified is not enabled in the sampling configuration then the text for the incorrect item will be shown in red. The units for a voltage channel must be one of V, mV, uV, nV, pV or fV. Similarly, the units for a current channel must be one of mA, uA, nA, pA or fA. Writing the units as ‘mVolts’ or ‘mAmps’ is also acceptable.

- The Control DAC item sets the DAC number that is used to control the stimulus, 0 to 7. This DAC must be enabled in the pulse outputs and the units set for the DAC must be appropriate for voltage or current (according to the clamp type) or the DAC prompt text will be shown in red.

- The Resistance measurement state specifies a state (a set of pulse outputs) to be used for measurement of resistance and other membrane analyses. This value must either be -1 or zero if multiple states are not being used, or from -1 to the maximum state number in use if you are using multiple frame states. Whenever this state is used during sampling Signal will automatically recalculate and display the membrane resistance. In addition, this state will be automatically selected (if it is not -1) when the membrane analysis dialog is used. If this value is set to -1 then the membrane analysis is always carried out, regardless of the state.

- A suitable stimulus pulse for resistance measurements needs to be defined for the control DAC in the specified state (or all states), this pulse is added via the Outputs tab -> Configure pulses, or may be modified during sampling if necessary. If there is more than one pulse for the state the pulse to be used for measurements can be labelled by setting the pulse ID to ‘RM’, otherwise the first suitable pulse found in the outputs will be used. Currently, only a square pulse (constant or varying amplitude) or a square pulse train can be analysed.

Back to contents

Scripts: Signal  

We have developed the APtresholdchopandtrack.sgs script for use with current clamping in Signal. This script allows you to track and discover the lowest stimulus threshold for a cell to produce action potentials. The download comprises two parts: the script file and the sampling configuration cclamp.sgcx.

The script itself essentially contains two functions, tracking the amplitude and a binary chop:

- The binary chop function allows you to input a known pulse amplitude which causes the cell to respond. The binary chop sets a new amplitude of half this value. Should the cell fire, the script records the amplitude of the pulse that caused the threshold crossing and denotes it as the new upper limit. If the cell does not fire, then the amplitude is recorded as the new lower limit. The script continues homing in until the lowest stimulus threshold is found.

- The tracking function increases the amplitude of a pulse by a specified amount. If the cell fails to fire it will continue increasing the amplitude until it does. If the cell does fire, the pulse amplitude is reduced to the specified amount. This process continues until either "chop" is selected or the script or sampling terminates.

To operate the script, first load the configuration and ensure the clamping set and configurations are correct for your equipment. Your cell response is also defined in the configuration file: when sampling, cursor 1 will search for a threshold crossing in the voltage trace that is above the level of horizontal cursor 1. Open the script and a new data file, and then run the script. The script creates a toolbar for you to choose either chop or track. If you chose chop, the script displays a dialog to enter the maximum pulse height and begins to binary chop based on that height. If you chose track, the script displays a dialog to enter the amount to change the pulse height each time. This value is set to 2.5mV by default, and is also used as the limit for the binary chop to stop homing (i.e. when the size of the pulse change is smaller in chop than it would be in track then the chopping stops). You would normally use chop first to home in as quickly as possible on the threshold then use track to monitor the threshold over time.

Back to contents

Scripters Corner – Variables

Welcome to our second scripters corner! In this issue we are discussing variables, what they are and how to use them. Our last feature covered view handles, which if you missed it can be downloaded here.

Variables are used in a script to hold and calculate values. They can be thought of as ‘boxes’ whose contents can be modified but whose name remains the same. Before a variable is used it must be ‘declared’ using the var keyword. Declaring the variable determines the type of data the variable can hold. This is followed by a list of variable names and terminated by a semicolon:

```
var myInt%, myReal, myString$;
```

This is the same as declaring the variables individually:

```
var myInt%;
var myReal;
var myString$;
```

Variables can be declared as one of four types: integer, real, string, and objects.

- Integer variables can only hold whole numbers and always have ‘%’ appended to their names, e.g. `myInt%`
- Real variables hold a real number which can have a fractional part and have no suffix, e.g. `myReal`
- String variables hold a string of text and have ‘$’ added, e.g. `myString$`

Object variables were added with Spike2 version 10 and are a special case in that they group data types, including other previously defined objects together into a single item that can be passed into user-defined functions and procedures. We will not be covering these further in this issue as they are for more advanced scripters, but it is useful to know they exist.

Once a variable has been declared then it can be accessed or 'called' within a script. For example:

```
View(LogHandle());      'Makes the Log view the current view
WindowVisible(1); 'Makes the current view visible
Window(30,30,70,70);    'Resizes the window
PrintLog("Start:-\n");  'Print "Start:=" to the log view and insert a new line
var i%;                 'Declare variable i%
i% := 3; PrintLog(i%);  'i% becomes 3, then print value of i% to the log view
i% := 4; PrintLog(i%);  'i% becomes 4, then print value of i% to the log view
i% := i% + 1; PrintLog(i%);   'Add 1 to i%, then print value of i% to the log view
var j%;                 'Declare variable j%
j% := 4; PrintLog(i% * j%);   'j% becomes 4, print the result of i% * j% to the log
view
j% := i% + j%; PrintLog(j%);  'j% becomes the value of i% + j%, then print the value of
j% to the log view
Halt;                 'Halt the script
```

This script shows how variables are defined and used; the values printed out when this script is run are 3,4,5,20 and 9. The `PrintLog()` function prints a value to the log file. The '`:=`' symbol should be read as 'becomes' not 'equals'. For instance `i% := i% + 1` should be read as 'the value of i% becomes the old value of i% plus 1'. The '`=`' symbol on its own tests the values for either size or equality and should be read as 'equals'. Whilst the variables above were declared and assigned values individually, it is also possible to assign values at the time of declaration. For example:

```
var myInt% := 4;
var myRea l:= 10.123456;
var myString$ := "String Variable";
```

When assigning a value to a string variable, the string of text must be encompassed by double quotation marks: `"This is a string of text"`. You may have also seen the special character \n in the previous script example, this inserts a new line into the printed string. There is also \t to inset a tab.

Now we have introduced to the different types of variables, there are two categories of variables you also need to be aware of:

- Global Variables are declared outside any function or object. Typically, these will be declared at the beginning of the script, and they can be called on from anywhere in the script after they are defined. All the variables you have seen declared as examples here are of the global category.

- Local Variables are declared inside a function (`Func`) or procedure (`Proc`) and can be accessed only from within the function and after the declaration. The `Func` and `Proc` keywords are used to begin a contained function/procedure within a script, but we will cover these in more detail in a later issue. The important point you need to take away is that local variables can 'hide' global variables with the same name. For example, if you have declared a global variable `Var Int%:=10;` and later on a local variable `Var Int%:=5;` within a function, then when `Int%` is called in the function it will return the number 5 and not 10.

If you tried to declare two variables with the same name in the same category (global, or locally within one function), you will be met with an error. You can name your own variables almost anything you like; however, it is usually best practice to keep the names short, logical, and descriptive. You do not want to get part way through writing a script and forget what the variable i% is used for.

Whilst declaring a variable determines the type of data held, it is possible to convert between data types. This may be necessary for some script functions to work. For example:

```
View(LogHandle());
WindowVisible(1);
Window(30,30,70,70);
var myInt%;
var myReal := 4.789;
var myString$;
myInt% := myReal;
myString$ := Print$(myReal);
PrintLog(myInt%);
PrintLog(myString$);
PrintLog(Val("642 apples"));
PrintLog(Val("I have 642 apples"));
```

This script uses the ':=' symbol to transfer the contents of the `myReal` variable to the `myInt%` variable. However, because `myInt%` holds integers, the actual value is truncated to the whole number 4 and printed to the log view. If you wished to round the real variable up or down we would first need to use the script function `Round()`.

Converting a real or integer variable to a string requires the `Print$()` or `Str$()` script commands. The `Print%()` function will convert the real variable enclosed in brackets into the string format, which is transferred to the `myString$` variable.

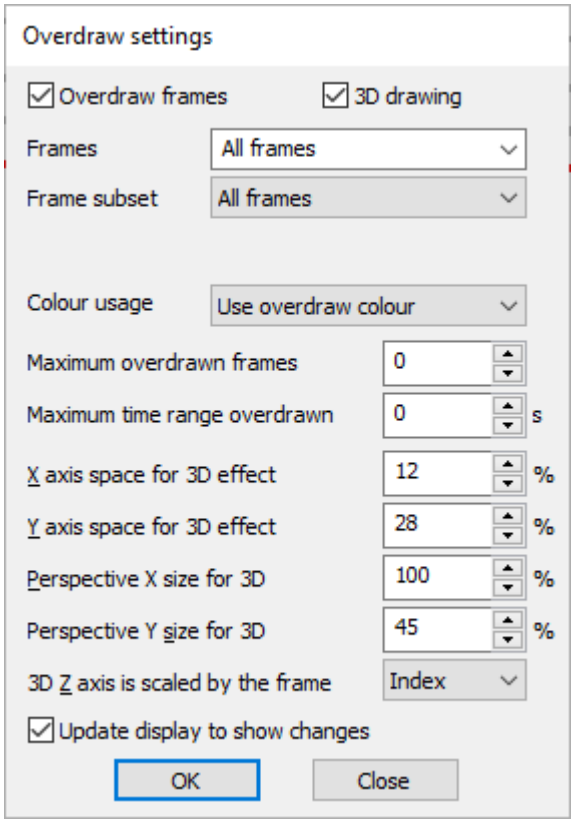
To convert from a string to a numerical value we could use the `Val()` or `ReadStr()` script commands. For example `myInt% := Val(myString$);` however this command will only convert up to the first non-numerical character in a string. In the above example `Val("642 apples");` returns 642, whereas `Val("I have 642 apples");` returns 0 as the first character of the string is not numerical.

Back to contents

Recent Questions

How can I display all my frames of data at once?

It is possible to overdraw all or selected frames of data within Signal. The overdraw feature is accessed by right clicking and navigating to *Overdraw Settings*, or with the shortcut *Ctrl+Shift+D*. The dialog allows you to set the frame display list and controls the way overdrawn frames are displayed. The upper part of the dialog is used to enable frame list overdrawing and to define the frames to be overdrawn, the lower part controls the optional 3D overdraw mode.



At the top of the dialog there are check boxes to turn on frame overdrawing and to enable 3D overdrawing. There is another check box at the bottom of the dialog which causes immediate display updates as you change fields in the dialog; very useful for getting a display just right.

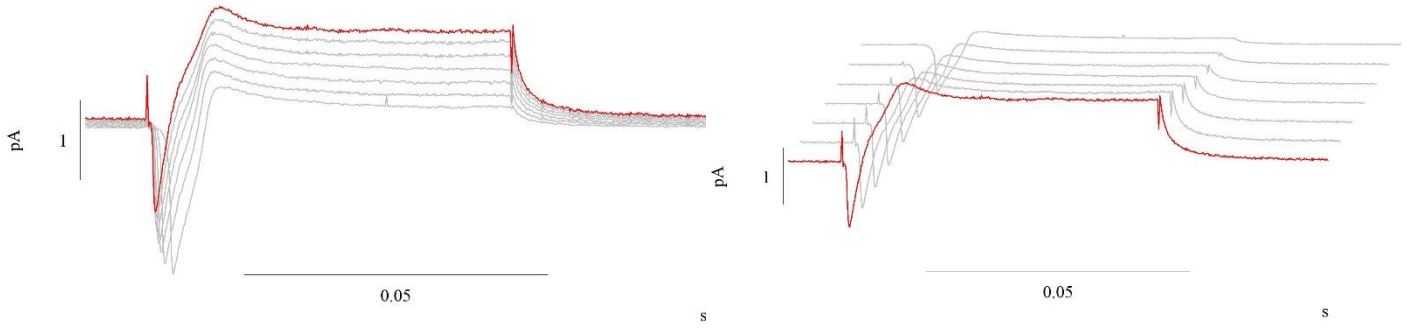
The *Frames* and *Frame subset* items select the data frames to be overdrawn. Set the frames either by entering a list of frame numbers directly or by using the drop-down list to select a category of frames: *All frames, Current frame, Buffer, Tagged frames, Un-tagged frames, Frame state = xxx* and *Last n frames*. A subsidiary field below allows a subset of these frames to be defined.

If you select *Frame state = xxx* the *Selected frame state* subsidiary field appears, use this to select the state number to be displayed. Similarly if you select *Last n frames* a subsidiary field appears for the number of frames desired; this selects the number of frames

before the current frame when used after sampling, and the number of frames most recently filed to disk when used on-line.

When the overdraw feature is used during sampling an extra *Sampled frames* option is available; this enables a minimum-delay overdrawing mode where the previously sampled frame data is not erased and new data is simply drawn on top of it.

The frame list is dynamic so that, for example, if you request all tagged frames and subsequently tag a frame it will be added to the overdrawn frames.



Back to contents

CED User forums

Try the CED Forums bulletin board for software and hardware support.

If you have any comments about the newsletter format and content, please get in touch: Marjorie@ced.co.uk.

To adjust your subscription preferences, please visit our website: www.ced.co.uk/upgrades/subscribeenews.

Back to contents

Contact us:

In the UK:
Technical Centre, 139 Cambridge Road,
Milton, Cambridge, CB24 6AZ, UK
Telephone: (01223) 420186
Fax: (01223) 420488

Email: info@ced.co.uk
International Tel: [44] 1223 420186
International Fax: [44] 1223 420488
USA and Canada Toll Free: 1 800 345 7794
Website: www.ced.co.uk