NEURON

Acquiring & Analyzing Neurophysiological Data

Wytse Wadman

2011-11-15

Content:

Main purpose of the program 3
DATABASE 3
STARTING NEURON 5
ON-LINE OPERATION 5
OFF-LINE OPERATION 7
Displaying data 8
ORGANIZATION OF THE DISPLAY SCREEN 8
MAIN DISPLAY INSTRUCTIONS 11
Recording data 25
HOOKING UP YOUR COMPUTER TO THE HARDWARE 25
RECORDING DATA 25
PROTOCOL SYNTAX: 27
PROTOCOL SPECIFICATION 30
Analysis using macros (scripts). 57
DESCRIBE GENERAL SET-UP OF MACROS, LOADING, SAVING, RUNNING. 57
MACRO COMMANDS 61
FITTING XY DATA TO A FUNCTION 77
THE FORMULA MECHANISM 81
ARTEFACT SUPPRESSION ROUTINES 90
SPIKE/EVENT ANALYSIS ROUTINES 92
PCA 102
DATASETS 103
SEGMENTS 106
COVERSION ROUTINES FOR NEURON 107
Internal organization (sweeps, traces, channels) *107*
EXPORTING RESULTS AND RAW FROM NEURON 108
Logger *108*
Exporting raw data *111*
RUNNING A SUPERSET TO TEST NEURON MACROS 113
Explaining Specialized procedures in Neuron 115
LEAK CORRECTION PROCEDURE 115
PCA ANALYSIS 116
SPIKE/MINI DETECTION AND ANALYSIS 117
Appendix A: Hardware documetation 118
INTERFACE EEG OPSTELLING "WIRELESS" 118
CONNECTORS ON THE FRONT (INPUT FROM 8 HEADSTAGES): 119
SUB ADC SYSTEM WITH NI6009 120
SUB ADC SYSTEM WITH NI6259 / NI6229 122
SUB ADC SYSTEM WITH NI6251 / NI6221 123
Index to protocol, macro, display commands: 124

Main purpose of the program

The main aim of NEURON is to provide you with a measurement tool that allows easy interfacing with DATA acquisition (sample any number of signals from an experiments for determined or undetermined time), easy production of control signals for experiments (voltage clamp protocol, stimulation patterns, shutter control) and control other equipment (Amplifiers, Camera, Stepper motor).
The basic philosophy of the program is that you try to define an experiment as complete as possible, so that you can run it very systematically. Data is stored in a data file that contains all info about the experiment and in such a way that it is easy and unambiguously accessible for analysis.
NEURON consists of a rather flexible (MATLAB) GUI interface that transforms a "text"-defined protocol into an experiment, where you have on-line view of how it proceeds and where you have optimal control over the essential parameters. In addition you get quite a bit of visual and computational feedback on the preliminary results so that you can continuously monitor the success of the experiment. Data are stored with complete description of the experiment to ease further analysis. The program is open ended. It can be hooked up to 1 - 3 (NATIONAL INSTRUMENTS or MCC) hardware devices and it is easy to add specific tools that control user defined experiments or specific hardware. Quite a few extrernal data formats can be read for analysis or converted to internal NEURON data formats (EDF, MEAtools, HARMONY, Raw text, WU-VSD).

Database

The whole systems serves only one purpose, collect data, store data and analyse data. As most of the data is collected in a physiological setting, the vast majority consists of traces that record some signal as a function of time.
Experiments can record from one to many channels, depending on the hardware in function. Channels have as a fundamental property that they are always recorded over the same time period. If you enounter a limitation it could be because we never used more than 32 channels. This should not be a principal limitation. We tested NEURON also on 60 chanel MEAs and 464 channel VSD data traces

Tooltips
Holding your mouse over a variable or pushbutton, will display a specific HELP text that explains the meaning of that button/variable. I try to keep them updated! Shout when some old meanings appear!

Data collection in NEURON can be divided in two fundamentally different types which are treated separately in the program.

The easiest one is called **Tracemode:**
It implies that at the start of the experiment (manually or by an external trigger) we do not know how long the sampling will continue. It is mostly stopped manually (but could also be done by an external trigger). The duration can last from ms to weeks (e.g. EEG in epilepsy recordings) only limited by the size of your data medium. In **Tracemode** you can only start the recording and stop it, there is not much more advanced to do. The data is then stored in a *.NRN datafile with a series of data segments of the type *0000.SEG …to *0nnn.SEG datafiles in order to keep the individual files a reasonable length, that can for example be backed-up and analysed. The results is one **Meting** (see below).

In **Sweepmode**, the length of the data to be recorded is known in advance and that implies that we can precisely schedule the start of the sweep and also record many of those sweeps in a predefined way. This implies that we can make very complicated protocols in **sweepmode** that can do a lot of

different measurements as a whole. Because you can exactly predict the flow of your experiment, it is also possible (even necessay to impose precise timung on your measurement. There are only a few limitations:
Each recording consists of a predefined number of data points simultaneously sampled form all input channels. These measurements can be exactly repeated in order to average or construct a time course etc.
The result is a 3-dimensional array with dimensions: time points, channels, sweeps (the last two can be equal to 1). Such a block of data is called a **Meting** and is the elementary unit on which quite a few analysis routines operate.
We can combine the input channels of a Meting with output trace (dependent on the hardware you can produce up to 4 analog output signals and sometimes up to 32 digital signals for a typical hardware device). Also most of these signals have a sampling rate (which is equal for all analog channels but maybe different from the input sampling rate, depending on the resolution needed). The signal may also have a different duration than the input sampling (but that can sometimes be difficult in the analysis). During repeated measures for an average, the output signal is onle generated once and then it is re-used during successive sweeps, so it is only present once, to reduce file size. Some forms of the output channels (FP stimulation, step-wise voltage clamp protocols for Voltage Clamp, are also automatically compressed to reduce file size by a factor of 100.

Almost at the same level of the Meting we can also define a Set of metingen, which are repetitions of the Meting with only small variations, such as an input-output curve for FP stimulation or activation/inactivation curves in Voltage Clamp experiments. Such a group of **Metingen** is often considered as a functional measurementand is therefore grouped into a **SET**. Your specific measurement can consist of any (finite) number of **SETs** that are put into one and the same *.NRN datafile.

Because your datafile reflects in a clever way the planning and the organization of a whole experiment it is very rewarding to think about the organization a little bit before you plan your experiments. If you can combine measurements that are related (e.g. controils and responses) you will have them also together for the analysis. Careful organization of this data structure might save you a lot of time later (e.g combine activation, inactivation, binding in one experiment), because those data is always loaded together in NEURON and can be analyzed in one run.

NEURON will now be described in 4 logically but partly unrelated steps:

1) Starting up and getting an overview
2) Displaying your data

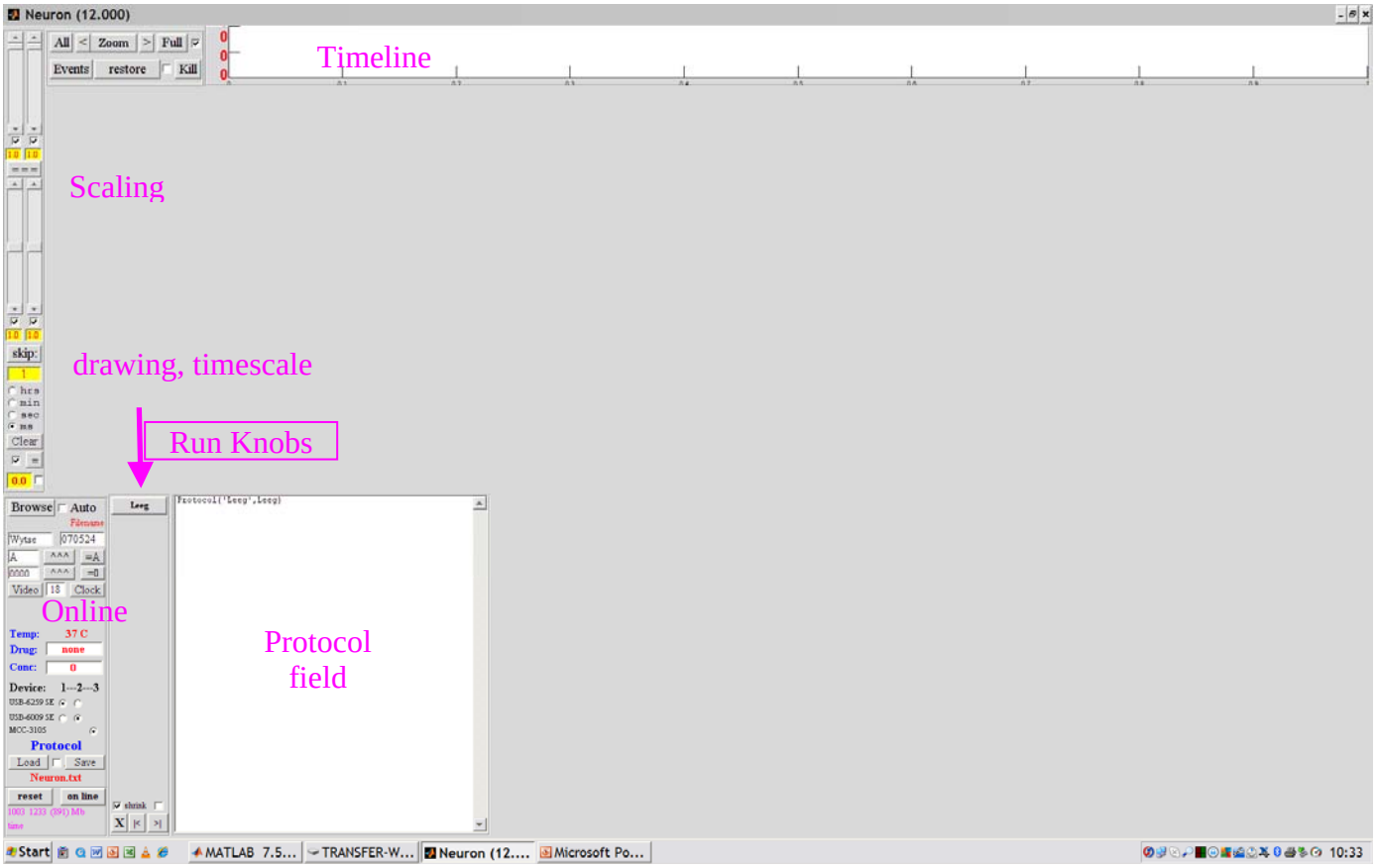
3) Planning and running an experiment, followed by a detailed list of commands needed to define a protocol
4) Examples of how to run specific experiments

5) Running analysis, followed by a detailed list of commands needed to define an analysis macro
6) Examples of how to run specific analysis forms

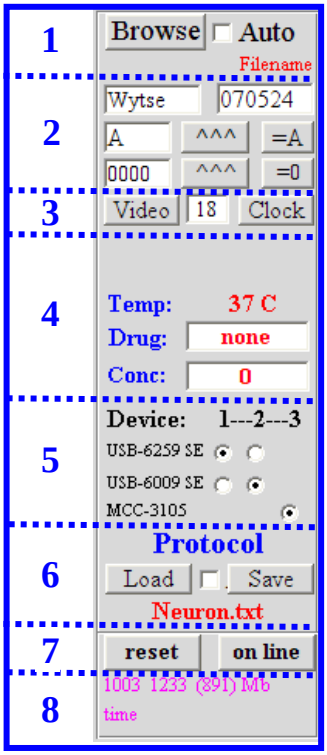
There may be a backlog in updating the description of the different programming parts, but the description of the various commands is mostly kept up to date and can always be used for reference.

Starting NEURON

Start NEURON by typing "**Neuron**" in the main Neuron mfiles directory. Normally this directory is the default one that you get after starting Matlab from your desktop. You get the following frame:



If you have hardware connected the program will start up in **Online** mode, otherwise it will startup in **Offline** mode.



The **scalebars**, the **online/offline,** the **Run Knobs** and the **Data Time line** mode panel will always be there. All the other objects you see are context dependent and will be configured dependent on the situation you create. Be in particularly aware about the Protocol and the Macro panel.

We will first give you a short overview of the basic elements in the window, although quite a few of the functions will become clear in the subsequent paragraphs.

The Online *(remember to use ToolTips for extra information)*

On-line operation

This panel only appears when you have data acquisition hardware connected (if not you will always be in the Offline mode and can not switch to Online.

--- section 1

Datafile saving. Use the Browse knob to define the directory where data will be saved (you can always change it). The first time you write a

datafile, the directory will be asked for. The directory name will then be remembered and used for successive savings. Do not put to much data in one directory, Clever use of subdirectories can be very helpfull.

If you hgave data available, the save knob will appear. With the save knob you can save the last recorded data from memory to disk.
If you activate the Auto marker, Save disappears and every file will automatically be saved after you record data.
If there is unsaved data in memory the Save knob will turn Bluey. The “filenumber” will then be automatically increased. If you manually increase the “fileletter”, that will reset the file number back to zero. See the following section.

--- section 2
This section is meant to define file names that are used to store the data. The file is a composition of several elements, that are aimed to produce well recognizable files with generation date and owner:

-- prefix usually contains the name of the owner. (This element can be defined in the protocol, so it will automativcally link the files to the owner of the protocols that were used to generate them.

-- data is information on the generation dat (year-month-day). It is automatically generated.

-- a letter from the alphabet (A—Z, or actually anything!) to indicate a subsection of your experiment (in our cases, take a new cell or a new slice or a new rat)

-- a number ranging from 0000 to 9999 to number the files that have some relation together.
Numbers and letters can be increased by one step (arrow knobs) and they can be set to their starting value (A and 0)

--- section 3
This section is controlling the video/ photo image acquisition.

-- The video knob will start the video The number behind it gives the format (will be upodated soon).

-- It also allwos you the scheduler. (a small timer module that let's you follow the protocol, and also allows you to warn for certain (programmable actions, such as drug application etc)

--- section 4
This section is used to record additional information from your experiment.

In the current example we measure the temperatue (Using the monitoring ADC and a temp sensor in the set-up (Qiluan/Pascal).
You can also enter the drug name that is used and the concentration (numeric!).

This information will be linked to each Meting node will appear in the data structure (temperature is measured before each Meting).
The other two you have to provide. Because some of this useful information could not be enetered in older versions of the datfiles, there is also a neat trick to extract them from the directory nems (see offline section).

--- section 5

The next section on the way to the top of the panel is the hardware section, which explains you which ADC devices the program has detected. At the moment you can work with three different ADC devices simultaneously. Their names show up at the left, you can determine here, which one will be use as the primary device(1), the secondary (2) or the tertiary device(3). I have not used these options to their full extend, but extensions can be easily made (ask if needed)

--- section 6

The section above the reset is devoted to loading Protocols. The load knob will allow you to load a Protocol text file that describes all your experimental set-ups. The Save knob allows you to save an edited protocol. Once you have edited a protocol the knob will turn Bluish to remind you that saving is necessary. It will stay Bluish until you save! If you activate the marker next to the save knob, you can invoke automatic changing of the protocol file after each edit. (Realize that this overrides your file! So it is unwise under those circumstances to clear your text!) The name of the current Protocol file is below the knobs in red, name and directory of the protocol file are conserved in the program.

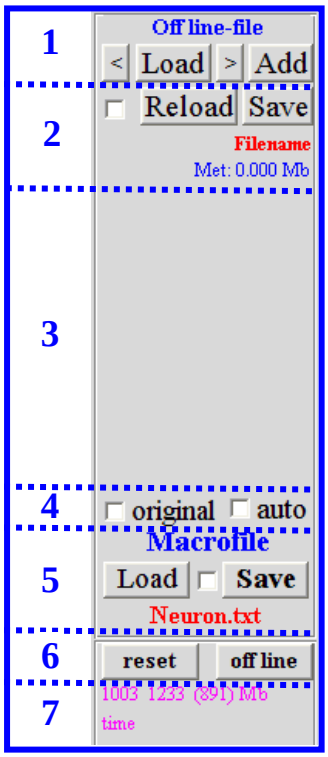
--- section 7

At the bottom of the panel we have a rest-knob, use it when the hardware does not respond any longer. it will through out of memory any data that was loaded and configure (almost) all elements back to their default conditions. You should never need to use under normal circumstances, so make a note if it starts to happen!
Next to the reset knob you have a switch knob that can move you between online and offline mode. Several options in the online mode are restricted as for example drawing might take so much time that it could interfere with your experimental timing.

--- section 8

Below the knobs some information on memory and time use is presented (this is again context dependent).

Off-line operation



With the online/offline knob in section 6 you can switch to Offline mode, which changes the appearance of this panel

--section 1

This section allows you to read an offline file (file.mat files only!). The arrow knobs (left and right) allow you to read the next file in the alphabetically sorted directory. The number indicates which number in the file list we are loading.

--section 2

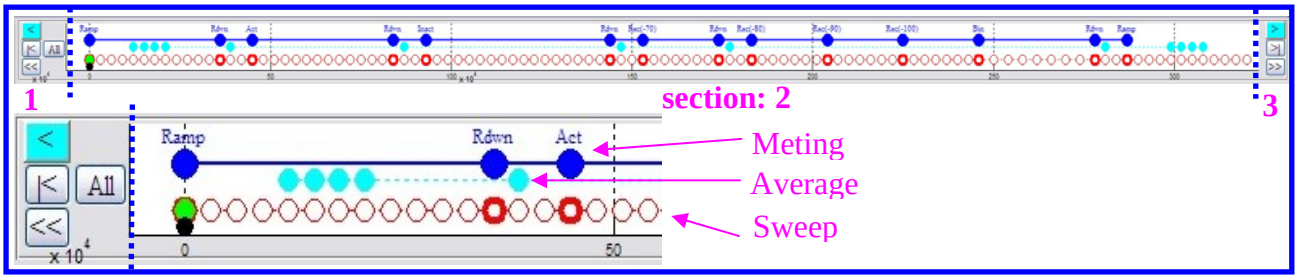
The knobs in this section allow you to save a datafile **[SAVE]** again (including nodes that were added during the analysis process). you can also remove a Meting **[KILL]**. Once you hit the knob, your cursor will change shape and the knob will trun RED indicating that the function is active, and stay that way until you have clicked on any **Meting** (see display description) which will then be removed. If you want to cancel the Kill option, hit the kill knob again, the cursor will return to its original shape and the red knob will turn grey.

--section 3

In this section you can define/find back information that is very useful when converting old data types to new ones, or to reconfigure them in Trace mode (will shift to macros in a later stage)

--- section 4
Similar to the description of how the protocol files are managed (Load, save, filename).

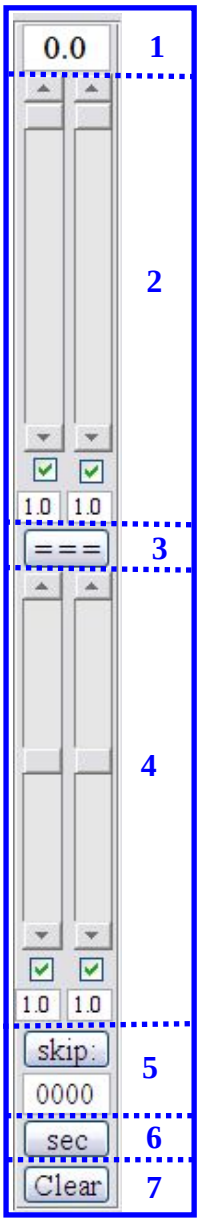
Now there are two markers more
-- *original*
when this marker is on, an offline file when loaded will be displayed in the same way as when it was measured. The advantage of this mode is that it will almost always work, while in other situations the display could easily crash if it is not at all suited for the specific data.
Under special circumstances it is possible that parameters that were defined at the moment of generating the plotter specifications are no longer known during replay. In that case you will be asked to specify that specific parameter, please use that chance, you only get it once!.
If you do not use original, you need to define the display options using the plotter definitions above the protocol (they appear as soon as you unselect original).
-- *auto*
With the auto marker you can force the system to go back to original display mode as soon as a new datafile is loaded offline.
--- section 6
Identical to the one described under the online window



Displaying data

Organization of the display screen

Assume you have loaded an offline datafile and want to display it. We will first treat the **Sweepmode** situation and then explain **Tracemode.** Once the data is loaded the Timeline panel could look like the following.
The Timeline window shows the content and organization of the data in the file.
--- section 2
The lower trace is the one that gives a red ball for each sweep in the Meting. If you click on it, it also has an output trace that is displayed, because you asked for it, (using dac; see below), an additional smaller and lower black ball will be showed. One level higher you find larger dark blue balls that indicate the Meting (set of sweeps). They have a name (given during the measurement, and this name is indicated above the ball.
If a measurement is organized in identical sweeps that were intended to be averaged, these averages will appear as light blue balls between the Metingen and the Sweeps. All these balls are intended to be pointed add and will display their content. You can do it in a few different ways:
lower red ball
-- left mouse pointer: show this Sweep
-- right mouse pointer: show this Sweep in original mode.
middle light blue ball
-- left mouse pointer: show this Average
-- right mouse pointer: show this Average in original mode.
top dark blue ball



-- left mouse pointer: show this Meting (= all sweeps)
-- right mouse pointer: show this Meting (= all sweeps) in original mode.
--- section 1

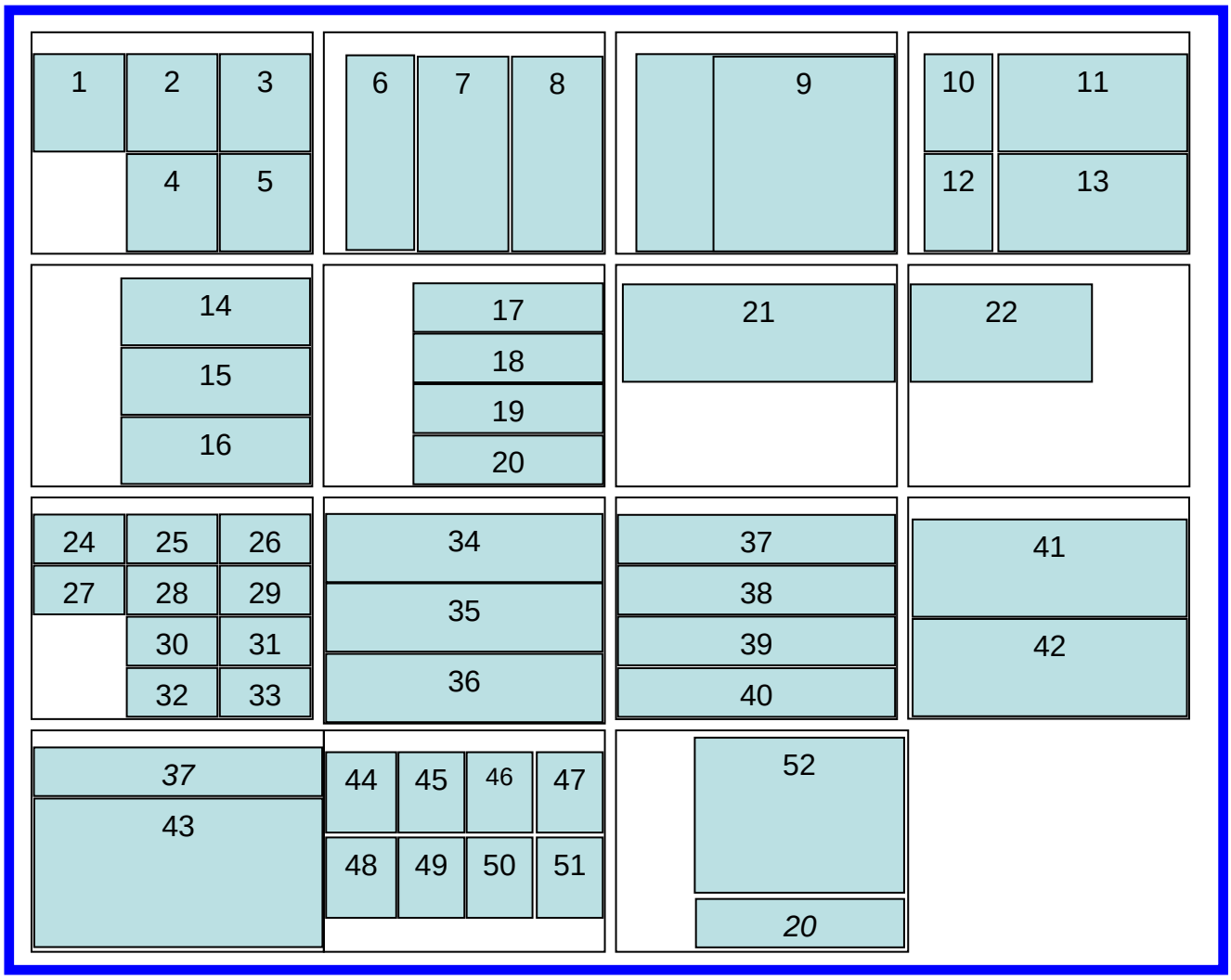
--- section 3

Gain and Stack sliders (work on all windows at the same time (except XY):
-- section 1
Topslider will define how to spread the traces over the plotframe (0 = superimposed, 1= maximal spread, non overlap, use a number as alternative, mark sets exactly to 1.
-- section 2
Lower sliders: amplitude gain of for channels, same as above.
-- section 3
The sliders are separate for ADC and DAC display, use the knob ‘adc dac/adc=dac’ to connect the two. Under normal conditions use markers to set gain to 1, then full scale will match with full ADC scale, the gain range is 0.1 to 10

Display your data:
If NEURON does not work in original mode, it will get the instructions on how to display from the

0	avg(1)/sem(2)
3	adc
4	dac
0	dac

command window. In this panel you can define 4 different views for the data you select to be displayed at the same time how a node will be shown. Each command has a number to the left, which defines a specific plot window. The meaning of the Window-Numbers are defined below. To the right is an instruction line that can be used to define how the display should take place.



The instruction line is identical to one command that can also be given in a maco command that will be defined later. The 40 different window nummers show a predefined display shape. In Macros any number of displays can be given. In the GUI mode we can define four different plots at the same time for each node you click.
Later you will learn how you can also redefine the window shape if necessary.

The textline is intended to define a large set of modifiers of the form:
adc(1:4)/legend/color('rgrgr')/csd

The general form demands a command and any number of optional modifiers separated by backslashes:

maincommand (parameters)/ modifier(parameters)/ modifier(parameters)
It is possible to add an exclamation mark as the first element of a modifier, which will eliminate that specific modifier (usefull for testing purposes avoiding a lot of typing). Some qualifiers exclude each other. If that happens, or you define one twice, the last one will be used. Case matters!

Main display Instructions

At least one of these needs to be present for anything to happen!

Parameters enclosed between curly brackets {} are optional.

==========

/none()

Empty command

Nothing to display

==========

/any (context dependent (see below))

Any is a useful command if you do not know what you want to display. In that case you click on any of the data-balls in the content-bar in the top of your window and depending on what type of data you point, any will be displaced by the command that fits the best. Ideal for scanning through data. Options only work if they in line with what you want to show.

==========

/adc (numset1, {numset2})

Plot multiple channels of a single sweep or trace of a measurement

numset1 **channel-list = any MATLAB type valid list of adc channels to plot, Remember that you now need the file number definitions (1 to max channel number), not the original ADC channel numbers**

numset2 **gainlist (optional), the display gain according to the following rules:**
-- default: all gains set to 1
-- single number: gain for all channels
-- set of numbers defining gains for selected channels (more gains than channels allowed)

example: *adc(1:5)*
adc([1 3 4])
adc([2:2:16])
or with gain specification
adc(1:5,10)
adc([1 3 4],[1 10 1])

==========

/avg (numset1, {numset2})

Plot multiple channels of a single average trace of a measurement (recalculate if necessary). If recalculation takes a lot of time you can prevent it with a knob in the zoom window (default is "on")

numset1 **channel-list as above**
numset2 **gain specification as for the adc's**

==========

/dac (numset1, {numset2})
Plot multiple channels of a single DAC trace of a measurement

numset1	**channel-list for dac as above**
numset2	**gain specification as for the adc's**

==========

/catch (numset1, {numset2})
Very special plot mode used only during continuous recording in order to display current data.

numset1	**channel-list = any MATLAB type valid list of adc channels to plot, Remember that you now need the file number definitions (1 to max channel number), not the original ADC channel numbers!**
numset2	**gainlist (optional), the display gain according to the following rules:** **-- default: all gains set to 25 (for some unknown reason!)** **-- single number: gain for all channels** **-- set of numbers defining gains for defined channels**

==========

/mets (mets-list)
Define which mets to use in an display (if there are more than one).

default is current met

==========

/swps (sweep-list)
– Define which swps to use in an display (if there are more than one).
default is current swp

==========

/fit ({num})
Plot the fit result

num select which fit to show if the node contains multiple fits, default = 1

==========

/hist (num1, {num2})
Draw a histogram of the column indicated by num1.

num1	Column in the events.xydata set to draw the histogram from.
num2	number of bins to use in the histogram (over the full datarange!)

Use the setx and sety commands to scale the histogram.

histogram is sensitive to the mask command.

==========

/foto ({num})
Display a measurement associated photo in the data window.

num	**which photo to display (default = 1)**

==========

/event ({num1,{num2}})
is basically the same as xy, with the difference that now the data are collected from the last xy list that you saved (use in combination with mask, if you want to be sure about which data to display

num1	which column in the data set to use for the x-axis
num2	which column(s) in the data set to display on the y-axis.

==========

/xy ({num1,{num2,{num3,{num4,{num5,{num6}}}}}})

Construct a xy graph from the column data in this node (mostly the result of a Formula Run or Spectra command analysis)

Realize that each result consists of a rectangular datablock with nn columns of mm realizations. In the xy graph you can display any column (x-axis) against any other set of columns (y-axis). If the dataset was generated in Formula, the columns are always the result of one of the formulas you defined with Formula_Set.
Depending on how you generated the data set the rows can mean measurements, sweeps or channels or any combination (not advised!).

The general format of the command is as follows:

num1	**which result data column in the 2e dimension to display as x-axis if you fill in 0 for this index you can use the inherent axis for this dimension.**
num2	**which result data column in the 2e dimension to display on y-axis.**
num3	**which result data column in the 3e dimension to display on y-axis.**
num4	**which columnblok in the data set to display on the y-axis. 1 = power 1 2 = power 2 (if spectra was run with 2 or more channels) 3 = coherence (if spectra was run with 2 or more channels) 4 = phase difference (if spectra was run with 2 or more channels)**
num5	**which xy column blok to use (formulas always one, spectra more complex 1 = raw data (or formula result) 2 = mean data 3 = PCA vectors 4 = PCA result vektors 5 = PCA coefs 6 = reconstructed vektors using nn components.**
num6	**which column(s) in the data set to display on the y-axis.**

This means for the general for: *xy(3,7,8)*
xy(2,1,[3 4 5])

Depending on the data several simplified formats are possible:
-- If you leave out number1, you will get results from the first dataset only. so any of the formats xy(**number2**, **numbers3**) works only on the first set.

-- **Number2** may be set to zero, in which case the x-axis will just be datapoint number (no meaning), but you can illustrate a good overview of datapoints that way. If you leave out **number2** (and also **number1**!), its value is assumed to be 0.
-- So any command of the form xy(**numbers3**) will assume datapoints as x axis, and illustrate column(s) **numbers3** from the first dataset.
-- If you set **numbers3** to 0 you can generate datapoints on the x axis, all with amplitudes zero, Can be useful in case of interesting moments in time (spiketrain) etc.

Use the /noline switch to prevent connecting the points by lines
Use the /legend switch to add almost complete description of the data (if it was generated with formula).

There is a long set of shortcuts that makes access easier by using less indices

---
/xyd ({num1, num2{, num3})
is equivalent to:
xy(**0, num1, num2, num3, 1**)
in other words, will show raw data using intrinsic axis

/xym ({num1, num2)
is equivalent to:
xy(**0, 1, num1, num2, 2**)
in other words, will show mean data using intrinsic axis

/xyms ({num1, num2)
is equivalent to:
xy((**0, 1, num1, num2, 2**), (**0, 2, num1, num2, 2**), (**0, 3, num1, num2, 2**))
in other words, will show mean data and sem using intrinsic axis

---
/xyv ({num1,num2)
is equivalent to:
xy(**0, num1, 1, num2, 3**)
in other words, will show PCA vector using intrinsic axis

/xyr ({num1,num2)
is equivalent to:
xy(**0, num1, 1, num2, 4**)
in other words, will show result of PCA analysis using intrinsic axis

---
/xyc ({num1,num2)
is equivalent to:
xy(**0, num1, 1, num2, 5**)
in other words, will show reconstructed vektor using intrinsic axis

---
/mxy (numset1{, numset2{, numset3}})

similar command as any of the xyL commands, but you can now compose the display of a series of indexed units

---

/mxyd (numset1{, numset2{, numset3}})
/mxyr (numset1{, numset2{, numset3}})
/mxyv (numset1{, numset2{, numset3}})
/mxyc (numset1{, numset2{, numset3}})
/mxym (numset1{, numset2{, numset3}})

All other options are in essence modifiers that can be added. They are there fore context dependent and most of them only have a meaning in combination with a specific instruction

==========

/gain (num1, {num2})
Set the gaim of channels (will be multiplied by the slider position of the gain slider)
There are two modes in which this will work:

gain (**num**)
num -- set the gain of all channels to number.

gain (**num1,num2**)
Overwrite the gain setting of these channels to specific channels, you can have more gain statements in one command line!

num1	-- channels for which to change the gains (these are the channel numbers in the file).
num2	-- values to which the channel gain will be set

==========

/blank (numset)
Blank a set of channels

numset	-- channels that will be nulled e.g for artifactual ones). this statement is equal to **gain** (**num**, 0* **num**);

==========

/map ({num1 {, num2 {, num3}}})
Draws 2D colormap only of the channel versus time data.

num1	number of colorlevels to use (default is 16)
num2	resampling the singnal (default is 1)
num3	gain settings for color (single number or one gain for each)

==========

/line ([numset
Draw a vertical line (or many vertical lines) at the time points indicated by number(s). Time in the same units as the graph (ms, s, min, hrs)

numset	time points at which to draw the vertical lines in the current window(s).

==========

/cross (num1 {, num2 {, numset}})

Draw one or multple cross sections in a separate window and use the cursor to specify the position. The cursor is indicated in the trace window, you can pick it up and move it, to show define the crosss sections.

num1	window in which to show the cross sections(s).
num2	time point at which to draw the cross section. If you omit the parameter, or use [] the cursor will start centered on the x-axis, once you move it, the current value is kept.
numset	give a list of numbers the MATLAB way in order to create multiple cross-sections and draw them simultaneously in the cross section window e.g. [0:2:10] or [10 20 50 150].

optional:

'ms'	Overrule the time setting that would otherwise be taken from the main time
'sec'	setting of the window.
'min'	In particularly useful if you switch time settings from time to time
'hrs'	(e.g. between sec and ms)
'points'	define in sample points

==========

/colormap ('name')

Define the colormap for the color graph to make

'name' -- any valid colormap

Redefine the colormap used in drawing the "color" option, will use the "map" as the new default, valid names:

jet	spring	gray
cool	summer	bone
hot	autumn	flag
HSV	winter	copper
prism	colorcube	pink
lines	white	dutchflag (number of levels)

==========

/sd ({num})
draws mean plus/minus sd (in red).

num how many sd signal to draw.

==========

/sem ({num})
draws mean plus/minus sem (in red).

num **how** many sem signal to draw.

==========

/vsd ({num1{, num2}})
Will draw the spatial representation for the 464 channel VSD array

num1 howindow in which to duisplay the VSD spatial pattern.
num2 time point for which to display the pattern

==========

/diff
Will calculate spatial differences between successive channels (First channel set to zero/original)

==========

/tdiff ({num})
Calculates timedifference using number of points separation.(estimate of time derivative)

num number of points width for time derivative.

==========

/csd
Will calculate second order spatial difference: csd (-1 2 -1) (First and last channel set to zero)

==========

/nonull
Do not draw the horizontalzero line

==========

/resample (num)
Resample the time axis with this factor (2D-color picture!)

num resampling level (same as resample command).

==========

/zero
/Zero
mean trace will be set to zero

/zero (num) **num** time in the trace trace will be set to zero

/zero (num1, num2)
num1 -- left timepoint of zero-range
num2 -- right timepoint of zero-range
one number: use that one for zeroing
no numbers means set the mean value to zero

/Zero As above, but now the time/region is indicated by blue vertical line.
/Zero (num1)
/Zero (num1, num2)

When Averages are calculated (with sd/sem!), traces are first zero corrected, then the new sd is calculated

==========

/submean
– subtract mean from individuals, while plotting

==========

Filtering the individual traces
/lpass (order,f1)
low pass filter over the plotted data.

order Order of the butterworth that will be used, (default = 2);
f1 f1 – cutoff frequency = 3db point of the butterworth filter

==========

/hpass (order,f1)
high pass filter over the plotted data.

order	Order of the butterworth that will be used, (default = 2);
f1	f1 – cutoff frequency = 3db point of the butterworth filter

==========

/bpass (order,[f1 f2])
band stop over the plotted data!

order	Order of the butterworth that will be used, (default = 2);
f1	f1 – cutoff frequency = 3db point of the butterworth filter
f2	f2 – other cut-off frequency = 3db point of the butterworth filter

==========

/bstop (order, f1, f2)
band stop over the plotted data!

order	Order of the butterworth that will be used, (default = 2);
f1	f1 – cutoff frequency = 3db point of the butterworth filter
f2	f2 – other cut-off frequency = 3db point of the butterworth filter

==========

/mpass (num)
median pass filter (non-linear)

num	the width of the median filter

==========

/mstop (num)
median stop filter (non-linear)

num	the width of the median filter

==========

/zoomlines
Set zoomlines in this graf, zoomlines are in the centre and are the Run-step apart. They indicate the range that is selected when the mousewheel is used (in combination with zoom, see below.

==========

/zoom
This window is a zoom of the current window (uze start & end of the zoom window, center it and then use Run-step, Together with zoomlines (above) you can nicely scroll through the display.

==========

/grid
-- will add a grid to the display 9if not already there0

==========

/noline
-- only draw symbols for this x-y line at the points

==========

/legend
-- will display an extensive list of specifications in each figure.

==========

/markers ({num})
Use markers to indicate individual datapoints (use this one or the one below)

num Mark this specific point on the trace (try for formulas etc)to some extend you can also use the formulas from formula here number indicates time point to show.

==========

/marker ({num})
Use markers to indicate individual datapoints (use this one or the one above)

num Mark this specific point on the trace (try for formulas etc)to some extend you can also use the formulas from formula here number indicates time point to show.

==========

/color ('chars')
Give the successive Colors of lines in xy / adc mode values. Use only the letters given below. Different modes have different defaults.

==========

/symbol ('chars')
Give the successive Symbols of lines in xy / adc mode values. Use the only the letters that are given below. Different modes have different defaults.

==========

/nosymbol
Suppress symbols in xy graph.

==========

/type ('chars')
Give the successive Types of lines in xy / adc mode values. Use the only these four letters:

s: solid
d: dotted
i: dot-strike
p: strike strike

==========

/width (num)
Define linewidth of the trace that you want to display.

num line width

==========

/logx
Use a log scale as x-axis

==========

/logy
Use a log scale as y-axis

==========

/logxy
Use a log scale as x- and y-axis

==========

/setx (num1,num2),
Scale the x-axis of the current graph
num1 -- fix left timepoint of zoom-range (default = -inf, use if you want automatic)
num2 -- fix right timepoint of zoom-range (default = inf, use if you want automatic)

==========

/sety (num1,num2)
Scale the y-axis of the current graph
num1 -- set y scaling bottom (default = -inf, use if you want automatic)
num2 -- set y scaling top (default = inf, use if you want automatic)

==========

/setxy, (num1,num2,num3,num4),
Scale the x-axis and the y-axis of the current graph
num1 -- fix left timepoint of zoom-range (default = -inf, use if you want automatic)
num2 -- fix right timepoint of zoom-range (default = inf, use if you want automatic)
num3 -- fix left timepoint of zoom-range (default = -inf, use if you want automatic)
num4 -- fix right timepoint of zoom-range (default = inf, use if you want automatic)

==========

/always
Always clear window before drawing (default)

==========

/first
Only clear window first time of drawing

==========

/never
Never clear window before drawing

==========

/group (number)

==========

/mask (number)

==========

/template

==========

/xyevent (number1, number2)

==========

/metdac (number1)

==========

/shift

==========

==========

/femke ({number})

number dd

==========

/spectrum (numset1,numset2,{*options*})

Calculate and display a power spectrum of one or two channels (by definition the first two channels of your channels selection are shown (if you select 2 traces, 2 spectra will be calculated, first one in blue, second in red just as first channel is shown in blue and second one in red).
Power spectrum is calculated over the piece of data in your display window. Skipping points fast plot option is ignored, all data points are used filter actions are first performed on the data.

numset1 **define the windows for the results**
num1 **window in which to plot the spectra**
num2 **window in which to plot the coherence (if you have 2 channels)**
num3 **window in which to plot the phase difference (if you have 2 channels)**
numset1 **defines which data to use for the spectrum, you have many options**

example 1
{signalwidth}, frame/fullframe/range
signalwidth **length of the elementary signal to use in the fft (1/t determines spectral resolution)**
if you do not give it, or set it equal to 0, it will be the same as the signal length that you analyse.
frame/fullframe/range
have the usual meaning as explained when they are options. This way of definition is in particularly useful when you are visually selecting parts of the signal that you also use for other purposes (e.g. spectrogram etc)

example 1 (applies if you do not use one of the options givven in example 1)
[start, span, signalwidth]
start **time in the trace that indicates which moment is the start point of the relevant signal.**
span **time length of the relevant signal.**
signalwidth **length of the elementary signal to use in the fft (1/t determines spectral resolution)**
remark all these times use the definition you set for time scaling in the left box of the GUI, you can overrule this by using one of the options 9see below) to redefine the time units in your command.
example 2
[start, span, signalwidth]
start **time in the trace that indicates which moment is the startpoint of the relevant signal.**
span **time length of the relevant signal.**
signalwidth **length of the elementary signal to use in the fft (1/t determines spectral resolution)**

example 1
signalwidth, frame/fullframe/range

signalwidth **length of the elementary signal to use in the fft (1/t determines spectral resolution)**

frame/fullframe/range
have the usual meaning as explained when they are options. This way of definition is in particularly useful when you are visually selecting parts of the signal that you also use for other purposes (e.g. spectrogram etc)

remark all these times use the definition you set for time scaling in the left box of the GUI, you can overrule this by using one of the options 9see below) to redefine the time units in your command.

number2 **number of realizations to use for the spectrum (splitting up your data pieces in chunks with 50 percent overlap, using a Blackmann-Harris window to prevent leakage). Other windows are possible, but they are not yet implemented. Scream if you need one!**

optional:

chns channel(s) to use. If you ignore it will use the first or the first two channels that you display (ignore all the others. Format as usual.

ms/sec/min/hrs Overrule the time setting that would otherwise be taken from the main time setting of the window. In particularly useful if you switch time settings from time to time (e.g. between sec and ms)

logx, logy, logxy. Can be used to specify log axis in x, y or both directions

setx, sety, setxy *(prefered)* Define axis in x, y, or both directions.

scalex, scaley, scalexy *(will fade out)* Define axis in x, y, or both directions.

==========

/spectrogram (number1, number2, number3,{'log', 'trace', 'scale', [n1, n2, n3, n4] })
Calculate and display the spectrogram of the channel you display 9or of the first channel in the set you are displaying).
a power spectrum of one or two channels (by definition the first two channels of your channels selection are shown (first one in blue, second one in red). Power spectrum is calculated over the piece of data in your display window. Skipping points fast plot option is ignored, all data points are used filter actions are first performed on the data.

number1 **window in which to show the spectrogram.**

number2 **time resolution of the spectrogram in seconds. Determines the quantization of the spectrogram along the x axis. Cannot be larger than the chunk of data you take! Using a Kaiser window with alfa = 0.5.**

number3 **size of a 2-D square filter used to smooth the final spectrogram. If you specify 2 numbers, the first one gives the size of the (square) filter window in the time direction, the second one the (square) filtering in the frequency direction.**

'log' **will draw the frequency scale (y-axis) on a log scale.**

'trace' **will draw the original trace in black superimposed on the spectrogram. Scale is visually the same as it would have been in the original plot window (i.e. reacts to the adc scale bar).**

'scale',[n1..n4] **standard use of scale facility (x-axis scaling makes no sense, will always be equal to the time axis of your original dataset**

optional:

'ms'/'sec'/'min'/'hrs' Overrule the time setting that would otherwise be taken from the main time setting of the window. In particularly useful if you switch time settings from time to time (e.g. between sec and ms)

logx, logy, logxy. Can be used to specify log axis in x, y or both directions

scalex, scaley, scalexy Define axis in x, y, or both directions.

==========

/PCAc (number1, number2, number3,)

Principal Component Analysis.

==========

/PCAv (number1, number2, number3,)

Principal Component Analysis.

==========

Ignore!

Recording data

Hooking up your computer to the hardware

Hardware Options:
The specific hardware determines the limiting properties of the data acquisition. We have tested the program on (and have sufficient NI devices in house):

PCI-6221	16 ADC	2 DAC channels	8-8-8 digital	sampling rate	250 kHz / 500 kHz
PCI-6259	32 ADC	4 DAC channels	32-8-8 digital	sampling rate	1.25 MHz / 2.5 MHz
USB-6009	8 ADC	2 DAC channels	8-4 digital	sampling rate	48 kHz / 150 Hz
USB-6221	16 ADC	2 DAC channels	8-8-8 digital	sampling rate	250 kHz / 500 kHz
USB-6251	16 ADC	2 DAC channels	8-8-8 digital	sampling rate	1.25 MHz / 2.5 MHz
USB-6259	32 ADC	4 DAC channels	32-8-8 digital	sampling rate	1.25 MHz / 2.5 MHz

You should stay within the hardware limitations (numbers of channels, sampling rates). i try to check for wrong parameters but can not guarantee that all are catched, especially the ones that are not wrong, but very unpleasant for later analysis. So please before you generate Gigabytes of not useful data, also check the analysis as soon as possible (in particular non-matching sampling rates, or sampling rates that needed to be rounded are tricky).
(as long as you stay within the hardware limits

Recording data

The minimum experiment you can do is the recording of a “set” of ADC channels, while at the same time (and exactly synchronous) driving a set of DAC channels.
For this experiment you can set a static set of digital output lines.
With slightly less accuracy you can also add additional digital pulses at controlled moments in time (~5 ms).

A set defines a set of ADC sweeps. They all contain the same number of ADC and DAC of equal duration. The ADC channels and DAC channels can have a different duration and an individual sampling rate.

Sets vary in say depolarization levels, stimulus interval, inter pulse intervals etc.

You have multiple ways to display a set (in a maximum of 6 different windows, which you can easy configure); you can average and do several other easy calculations on the data.

Once you understand how to create a “set of sweeps” you can stack different sets on top of each other and build more (even infinite) complex experiments. You have facilities to control the order, the details of the timing and the averaging.

There is a primitive language that allows you to define protocols, in particularly the output. The input is relative trivial.
It is good to realize that variables/parameters can have ntwo different “scopes”. They either are specific to a protocol, or they are general and than the same for all protocols (unless they are redfined, within a protocol. This redefinition only holds within that specific protocol.

Another way to “generalize” parameters is to declare them as a variable. For MATLAB reasons that needs to be done as a “struct withy main name “p”, so legal names are p.duration, p.intensity, p.depolarization etc.. etc. There is no limit.

There are a few predefined variables that have a very special meaning. (see language definition)

Names must be legal MATLAB names.
Once you have variables, you can use them in legal mathematical formulas like:
 2* p.duration/p.amplitue

Also realize that (in principle) there is no detailed order in which the statements are evaluated. If you define a panel twice, than only the first one will be used. I need to be more careful in generating error statements for such conditions!

Once you have loaded a file that defines protocols, you can start action by hitting one of the “run” knobs’
Your protocol textfile can also define special panels that allow you to initiate action. (Starting a measurement or an analysis).

==========

Protocol syntax:

General:

All commands are separated by a comma ","

Special charaters:

Comment indicator:	"**!**"	skip anything from ! until end of line
Comment indicator:	"**$$**"....."**$$**"	anything between $$ and $$ is ignored, nesting will not work!
Percentage indicator	"**%**"	anywhere in a number instructs the program to take this value relative (for stimulus intensities this means between min and max as defined in the EP panel.
Substitution / assignment:	"**=**"	**Vh1 = -65** will give Vh1 the value of -65

Order of evaluation (not essential, but you understand better what is going on!):

1) Remove all comments that follow an exclamation mark: !
2) Concatenate all lines (lines have no meaning!)
3) Remove enclosed comment $$ .... $$
4) Find all "protocols" as the parts between {…} and save them for later

A very few parameters are predefined and have non-changeable meaning, (do not use them in a different context!!):

Tijd	=	absolute time since the start of the specific protocol (Tijd = 0)
Vh1	=	potential 1
Vh2	=	potential 2
Ih1	=	current 1
Ih2	=	current 2
Istim1	=	value of the left slider defined value in the EPpanel
Istim2	=	value of the right slider defined value in the EPpanel
DAC1	=	hardware output channel 1
DAC2	=	hardware output channel 2
DAC3	=	hardware output channel 3
DAC4	=	hardware output channel 4
mouse	=	value of the mouse (click) in the graf panel
events	=	start of the event
evente	=	end of the event

All other variables parameters need to be defined as members of the MATLAB struct p.
Thus of the form:
p.Vm
p.V1
p.anything, etc.

Make it a habit to define a parameter only once, so if you need 2 times the same value in your protocols, then use a variable. Look at the examples I provide. It will prevent a lot of errors. If in doubt please let me have a few looks on your protocol files.

In the sections below (and in the language) the following definitions are used:

number is any (valid MATLAB) expression that will finally result in a number.

It can therefore be of the form:

-- constant	(any real or integer value, inclusive a sign)
-- expression	(any number)
-- number1:number2:number 3	(loop expansion in standard MATLAB);
-- [number1 number 2 number3]	(list expansion in standard MATLAB);

The most important structure to define is the set of sweeps, that can be executed. It has the form:

protocolname {*here we will define the specifics of the set***}**
The curly brackets, uniquely define the set!
Defines a set of a specific number of sweeps of input ADC channels and output DAC channels. All ADC traces have the same length and all DAC traces in the set have the same length (but may be different from the ADC sweep length). All ADC sata is sampled at a specific rate and so is the DAC data (ADC and DAC may again be different.

Once you have defined at least one set, you can make it available to the user by:

Protocol {'Knob', protocolinstructions}

If you know that your protocol is specific for only the VC or the CC mode of the first amplifier you can use

VCProtocol {'Knob', protocolinstructions}
CCProtocol {'Knob', protocolinstructions}

This is equivalent to the Protocol statement, but the knob will only show up in the specified mode (also prevents clutering of the command list).

This command will create a pushbutton in the GUI that you can use to start protocol ***Setname***. The pushbutton gets the name ***Knob***.

Instead of Setname in the Protocol command you may also define a lkist of sets and if you want a set to be executed five times you can even use the construct 5*Setname, thus:

Protocol {Knobname, Setname, Supername, 5 * Setname, must be valid (existing) Setnames **}**

Creates a knob with text Knobname and will execute the list of sets defined after Knobname in the order they are generated (be aware that if you specify absolute times for execution, this order can be adapted. that allows to run this rest of protocols, once started.

There is one important restriction to keep in mind. You will see below for the output sweeps that it is often possible to define in one statement multiple output sweeps for say DAC1 (input-output curve / set of depolarizations etc. In principle you can mix all kind of definitions for the various

output channels as long as the number of sweeps that they define for the same protocol is identical or equal to 1. If it is 1, that trace will simply be repeated all the time.

Look for the description of specific panels if you want more information about other parameters.

Basis {……..}

Many parameters can be defined within a protocol definition or outside. If you define them outside a protocol, then these parameters are valid in all protocols that you define. However you can always redefine them within a protocol, even differently within several protocols.
The following commands can be defined globally and re-defined at the single protocol level

Using parameters

Protocol {'Knob', *protocolname* ({parameters}) , protocolname}

protocolname({**parameters**}){instruction, instruction(**par2**), instruction(**par1,par3**)…..}

parameters = **par1, par2, par3**

There are limited constructions possible:

Repeat (**number,** *any list of protocols*)

Alternate (*any list of protocols*)

Protocol specification

Commands:

Input:

These commands define the pattern that will be output on an analog output channel.
Mostly be used to stimulate, define voltage in Voltage clamp, drive shutters etc.
They can thus define a whole series of sweeps.

==========

ADCmode (num, text)
Defines the mode in which the ADC given by device number num will operate. remember that all channels of an ADC operate in the same mode (Differential or SingleEnded). This only affects the hardware channel numbers and the way you connect the your input to the ADC. Lateron or in the analysis the operation mode is irrelevant.

-- **num:** device to define
-- **text:** text that defines the adc mode for this device. there are only 2 options:
-- ***'SingleEnded'***
-- ***'Differential'***

==========

Amplifier (text1, (specific parameters), adcs);
Define the stimulator that is hooked up to your hardware and if necessary define the properties. You can define multiple stimulators but have to take care that you do not double define DAC channels.

text1 Define which amplifier you hooked up, (several are predefined):

You have to define any channel you want to
-- **EPs**: input channels defined for field potentiels (=external amplifiers).
format: **Amplifier (EPs, adcs, pregain)**
-- **adcs**: channels to reserve for Field Potential recordings

-- **pregain**:
-- if <0 then abs (pregein is overall gain of channel
-- if >0 then pregain is gain of pre-amplifier, main gain from panel (first time)

-- **HEKA**: connected to a HEKA amplifier
Amplifier (HEKA, 'protocol(s)', numset1, {numset2})
we assume adc channel 1 and dac channel 1 are used for connection to HEKA
we also assume that the USB6009 is hooked up as moonitoring device (see connection schemes in the appendix)

protocol(s), protocol to run under Voltage Clamp & Current Clamp test conditions (in the amplifier panel). You need to give valid protocol names! The format is the standard set format:
example: **'VCprotocol;CCprotocol'** defines:
-- **VCprotocol** as the test Vprotocol under voltage clamp
-- **CCprotocol** as the test Vprotocol under voltage clamp
You may leave out the CC protocol!

numset1: [r1 r2 r3], VC scale and transfer parameters to HEKA
-- **r1**: VC hold potential on the HEKA
-- **r2**: VC adc gain, make sure it matches the value on the HEKA amplifier (check in the panel.
-- **r3**: VC dac gain, make sure it matches the value on the HEKA amplifier (check in the panel.

numset2: [r1 r2], CC scale and transfer parameters to HEKA
-- **r1**: adc gain under VC
-- **r2**: VC adc gain, make sure it matches the value on the HEKA amplifier (check in the panel.

-- **Axon200/Axon200A/Axon200B**: Those types
Amplifier (Axon200X, 'protocol(s)', {Default})
we assume adc channel 1 and dac channel 1 are used for connection to HEKA
we also assume that the USB6009 is hooked up as moonitoring device (see connection schemes in the appendix)

-- **protocol(s)**, protocol to run under Voltage Clamp & Current Clamp test conditions (in the amplifier panel). You need to give valid protocol names! The format is the standard set format:
'VCprotocol;CCprotocol'
defines:
-- **VCprotocol** as the test Vprotocol under voltage clamp
-- **CCprotocol** as the test Vprotocol under voltage clamp
You may leave out the CC protocol!

we assume adc channel 1 and dac channel one are used for connection
and we assume an USB 6009 to drive the telegraph contacts
-- **Default**, Startmode up the amplifier (default read it from the Axon switch)
= **'VC'** start under voltage clamp
= **'CC'** start under current clamp

-- **Axon700**: Connect an Axon 700 to channel 1,2,3,4 ADC and 1,2 DAC
Amplifier (Axon700m VC-amp1, CC-amp1,VC-amp2, CC-amp2, {bool}))
-- **VC-amp1**, protocol to run under Voltage Clamp conditions amplifier1
-- **CC-amp1**, protocol to run under Current Clamp conditions amplifier1
-- **VC-amp2**, protocol to run under Voltage Clamp conditions amplifier2
-- **CC-amp2**, protocol to run under Current Clamp conditions amplifier2
-- **0/1:** 1, means coupling the state between amplifier 1 and 2 (default 1).

-- **Own**: define your own (to be implemented)

==========

Modus (txt1, {txt2})
Set up the interface with the Axon700 amplifier

-- **txt1:** Checks whether amplifier 1 is in the right state for this protocol
Valid values are:
VC protocol requires voltage clamp
CC protocol requires current clamp

Any protocol can run in VC and CC mode (internal control using ISVC/ISCC)

-- **txt2:** Checks whether amplifier 2 is in the right state for this protocol
Valid values are:
VC protocol requires voltage clamp
CC protocol requires current clamp
Any protocol can run in VC and CC mode (internal control using ISVC/ISCC)

==========

NoEEG
Checks that there is not an EEG protocol running, because that is not allowed

==========

Seal (num1, val1, { num2, val2})
Define the seal properties of amplifier 1

-- **num1:** Adc channel on which the seal is checked
-- **val1:** Threshold of impedance on which the seal will be performed.
optional the same values for amplifier 2, assuming that it was.
-- **num2:** Adc channel on which the seal is checked
-- **val2:** Threshold of impedance on which the seal will be performed.

==========

SoftClamp (num1, num2, num3)
Define the softclamp parameters for this amplifier

-- **num1:** Sampling rate (Hz) of the softclamp loop for both amplfiers
(do not overdue it, 10 Hz is already fast, 100Hz is a lot! You will create noise with this frequency in your signals).

-- **num2:** value of the gain of the loop

-- **num3:** max jump allowed in the loop correction (in pA) (lower reduces the adjusting speed, but increases stability)

==========

JunctionVC (value, numset1, numset2, numset3)

JunctionCC (value, numset2, numset3)

Option to include junction potential in your measurements.
If you use HEKA or Axon700, use the corrections provided on the amplifier, but fill in the number here, it will be included in the datafile for documentation purposes. But I can not check any errors!

The minimum you need to give is the junction potential value (in mV) value, use external (web) calculation programs to determine it.

This value will in voltage clamp be added to your command voltage channel (always on dac-1) and subtracted from the recorded voltage channels (default = adc-2 on the monitor device and channel 2 on the main sampling device, but you can overrule these values).
In current clamp it will be subtracted from the recorded voltage channel (default = adc-1(scaled) & 2, same as in voltage clamp) and also from the monitoring channels 1 (scaled) & 2.

If you need another configuration you need to specify all channels! (if you have to drive the command voltage from another channel contact Wytse!)

-- **value:** value of the junction potential (in mV) to use for amplifier 1.

The following sets are optional, but if you give one, you have to give them all!

-- **numset1:** the dac channel(s) to be corrected for junction potential (e.g. channels that control the cell voltage, ergo only relevant in Voltage Clamp).

-- **numset2:** the adc channel(s) to be corrected for junction potential on the main recording ADC device (e.g. channels that record voltage).

-- **numset3:** list of adc channels to be corrected for junction potential on the monitoring ADC device (e.g. channels that record voltage on the USB6009).

==========

RecordSweep (numset, num)

RecordSweep can measure a set of traces of defined length. The organization can be quite complex and the total set very large. It is important for the analysis to think about how you recird your sweeps and how and how many you put into one file. As a single file allows you linked analysis that can save you an enormous amount of work in the analysis.
You cannot mix Trace and Sweep mode commands in one file (as the first ones can be of unknown duration).

-- **numset:** Hardware channels that you will use in this recording. The hardware channels are simply the numbers of the channels that are available (always between 1 and max channels, numbered continuously. They map in a special way to the NIDAQ hardware chanels (i.e. the pin chanels that you have to connect your wires to: Depending on the mode your ADC operates in the map as:

SingleEnded:

channel 1 corresponds to NIDAQ input 0.
channel 2 corresponds to NIDAQ input 1.
etc to
channel 32 corresponds to NIDAQ input 31.

Differential:

channel 1 corresponds to NIDAQ input 0 versus 8 (or 16 for a 32 channel ADC).
channel 2 corresponds to NIDAQ input 1 versus 9 (or 17 for a 32 channel ADC).
etc to
channel 8 corresponds to NIDAQ input 8 versus 15 (or 23 for a 32 channel ADC).

and for a 32 channel ADC
channel 9 corresponds to NIDAQ input 9 versus 24.
channel 10 corresponds to NIDAQ input 10 versus 25.
etc to
channel 16 corresponds to NIDAQ input 15 versus 31.

This is the only place where hardware channel numbers matter, so you can record from only channels 8,9,11 if you want. Or put them in a different order, so record from channel 4,3,5,1 if you like that for any (strange) reason. But from now on these channels will be considerd as channel 1 to maxchan imn the program. The original hardware channels are stored for documentation purposes,

but never used again. In differential mode the situation in hardware terms can be even more complicated, but the end result is the same. This has huge advantages for your analysis macros as very often the actual hardware channel is completely irrelevant and you do not want to keep track of whether you originally recorded the voltage on channel 3 or 12.
This already starts as soon as the data is in. So you can record channel 8:10, but in the same routine you plot them as plotter (1:3)!
The hardware scaling is directly coupled to the hardware channel numbers as I define them (so 1:maxkan, in either single ended or differential mode.

-- **num:** duration of the sweep in ms.

==========

RecordTrace (numset, num2, num3, {num4{,num5}})
Record Trace in EEG-style mode (contionuous recording, if necessary of infinite length, piece-wise displayed. You can only measure one such Trace in a Protocol.
You cannot mix Trace and Sweep mode commands in one file (as the first ones can be of unknown duration).

-- **numset:** Hardware channels that you will use in this recording. The hardware channels are simply the numbers of the channels that are available (always between 1 and max channels, numbered continuously. They map in a special way to the NIDAQ hardware chanels (i.e. the pin chanels that you have to connect your wires to: Depending on the mode your ADC operates in the map as:

SingleEnded:
channel 1 corresponds to NIDAQ input 0.
channel 2 corresponds to NIDAQ input 1.
etc to
channel 32 corresponds to NIDAQ input 31.

Differential:
channel 1 corresponds to NIDAQ input 0 versus 8 (or 16 for a 32 channel ADC).
channel 2 corresponds to NIDAQ input 1 versus 9 (or 17 for a 32 channel ADC).
etc to
channel 8 corresponds to NIDAQ input 8 versus 15 (or 23 for a 32 channel ADC).

and for a 32 channel ADC
channel 9 corresponds to NIDAQ input 9 versus 24.
channel 10 corresponds to NIDAQ input 10 versus 25.
etc to
channel 16 corresponds to NIDAQ input 15 versus 31.

This is the only place where hardware channel numbers matter, so you can record from only channels 8,9,11 if you want. Or put them in a different order, so record from channel 4,3,5,1 if you like that for any (strange) reason. But from now on these channels will be considerd as channel 1 to maxchan imn the program. The original hardware channels are stored for documentation purposes, but never used again. In differential mode the situation in hardware terms can be even more complicated, but the end result is the same. This has huge advantages for your analysis macros as very often the actual hardware channel is completely irrelevant and you do not want to keep track of whether you originally recorded the voltage on channel 3 or 12.
This already starts as soon as the data is in. So you can record channel 8:10, but in the same routine you plot them as plotter (1:3)!

The hardware scaling is directly coupled to the hardware channel numbers as I define them (so 1:maxkan, in either single ended or differential mode.

The next set of numbers defines how long you record and how you display and store the data:

-- **num2:** refresh time of the display in second.

-- **num3:** time to view in the display window in seconds

-- **num4:** duration of a segment in the Trace file
If you omit **num4** (and num5) the segment duration will be equal to the view duration (or if too long to the maximum size you defined with **Filesize**)

-- **num5:** number of segments to be recorded (any number between 1 and inf, inclusive).
If you omit **num5** inf is assumed, e.g. measurement will run for ever, or disk full, whatever comes first.

Boundary conditons:
(**num3** must be a multiple of **num2**)
(**num4** must be a multiple of **num3**)

==========

Catch (num1, val1, txt, val2)
Record Trace in EEG-style mode (contionuous recording, if necessary of infinite length, piece-wise displayed. You can only measure one such Trace in a Protocol.
You cannot mix Trace and Sweep mode commands in one file (as the first ones can be of unknown duration).

-- **num1:** ADC channel(s) on which to detect spikes on-line.

-- **val1:** threshold for spike detection (units of the channel).

-- **txt:** 'up' for up-going level crossing
'down' for down-going level crossing

-- **val1:** binning value for the rate plot (and data storage) in (s).

==========

Impedance (txt1, num1, txt2, num2, timeset)
=
Impedance1 (txt1, num1, txt2, num2, timeset)

Impedance2 (txt1, num1, txt2, num2, timeset)
Calculate on-line the impedance from a voltage and a current pulse (in VC or in CC)

-- **txt1:** ADC channel(s) on which to detect spikes on-line.

-- **num1:** ADC channel(s) on which to detect spikes on-line.

-- **txt2:** ADC channel(s) on which to detect spikes on-line.

-- **num2:** ADC channel(s) on which to detect spikes on-line.

-- **timeset:** threshold for spike detection (units of the channel)

==========

Colormap (txt)
Select the colormap for record display

==========

Names (mode,numset,num1,txt)
Name the channels for recording (will affect the names as they appear in the data file.

-- **num1:** ADC channel(s) on which to detect spikes on-line.
-- **num1:** ADC channel(s) on which to detect spikes on-line.

==========

Output:

These commands define the pattern that will be output on an analog output channel.
Mostly be used to stimulate, define voltage in Voltage clamp, drive shutters etc.
They can thus define a whole series of sweeps.

==========

Stimulator (DACx, Type(scale, unit));
Define the stimulator that is hooked up to your hardware and if necessary define the properties. You can define multiple stimulators but have to take care that you do not double define DAC channels.

DACx for all stimulators you have to define to which hardware DAC channels this stimulator(typr) is connected)

Type Define which stimulator you hooked up, several are predefined:
-- **FNWI**: standard dual channel stimulator with a default range of 2000 uA.
to overrule the range, give new one (200)
FNWI(maxvalue,unit), you may overrule the standard

-- **Gent**: Gent Cyriel stimulator 5V pulses (intensity is manual and external)
To run this stimulator, you need to define 2 DACs [DAC1 DAC2]

-- **DS4**: Digitimer stimulator with a default range of 1000 uA
to overrule the range, give new one (200)
DS4(maxvalue,unit), you may overrule the standard

-- **NPI**: multi-channel stimulator from NPI,
without parameters we assume you drive it through the direct input
NPI(maxvalue,unit), you may overrule the standard value of 2000 uA
or you can use the MCC
NPI(MCC), you may overrule the standard value of 2000 uA

-- **Own**: define your own stimulator, you have to add the stimulator value that corresponds to 10 Volt driving input (and the units).

==========

Stimulus (type,[number list1],[number list2]);
This defines the stimulus pattern that will be put at any point indicated in the ***STIM*** command defined above, the meaning of "parameter" depends on type:

"none" Stimulus will be ignored, parameters have no meaning (easy way to switch Stimulus off)

"puls" Generate a puls pattern defined by durations and amplitudes:
-- Parameter1: list of durations (in ms)
-- Parameter2: list of amplitudes (in units of relevant DAC
(exactly the same number as P1)

"Gentpuls" Generate the two 5 Volt "digital" pulses on the two DAC outputs (1-2 or 3-4 that are needed to drive the Cyriel stimulator in Gent.
(Needs to be better tested!)

"sine" Generate a sine wave pattern defined by parameters
-- Parameter1: first par is the period of the sine wave
-- Parameter2: first par is the total period the sine wave will last.
(Easy to expand to a sum of sinewaves with the same phase, if needed?)

'shape" Defines apattern in terms of delta-x (times) and delta-y (amplitude), useful for generating ramps, triangles and elephants.

==========

Stim (DACx, duration, 'group-of-numbers , group-of-numbers');

This command defines a trace on DACx with total duration of duration in ms putting a stimulus pattern at each position defined by **list of numbers**, thus:

DACx DAC output to use (DAC1-DAC2-DAC3-DAC4, if present in this NIDAQ equipment.

duration total duration of sweep in ms
Remember that each trace in the DAC MUST be of same length (contain same number of points, use same samplerate)

group of numbers text string that contains the stimulation pattern

The meaning of **group of numbers** contains from one to four numbers or group of numbers:

moment amplitude {repeat {group repeat}}

The first two numbers are obligatory, the third and fourth are optional e.g. you cannot give a fourth without the third (but that also does not make sense, see below)

The meaning of the first two number(sets) is moment in time of stimulus and amplitude, considering stimulus as a point process.

Moments in time are in ms.

Amplitude is in units as indicated by the scale. If you make amplitude contain a % character than you will get the percentage value taking the left/red slider as zero and the right/blue slider in the EPpanel as maximum. See intructions with this panel if you want tyo know how to hanle it.

At each moment in time a Stimulus as defined above will be given (make sure the sampling rate is sufficient!)

If moment or amplitude contain more than one value, they are values that define multiple traces. In one command series you either have one trace or you have multiple traces. In the last situation all definitions in the protocol need either give one or the same multiple number

With the third number you can repeat the pattern defined by the first two.

And with the fourth you can repeat this whole sequence again.

The values (in units time e.g. ms) of the last two number(sets) are simply added to the already defined times. In this way you can produce theta burst stimulation etc. etc.

You can give multiple '**groups of numbers**' separated by a comma. It is up to you to make sure that these elements are logic and non-interfering. If they override, the last one simply adds to the already existing ones.

The minimal stimulus is defined as a pair of "moment" & "amplitude"

[moment amplitude]

As above but the stimulus will be repeated at moments defined by "repeat" (any valid MATLAB sequence) that will be added to moment. Repeat is ideal to define an identically repeated stimulus pattern)

Moment and amplitude can also define a set, which will generate a set of traces. In that case all sets defined within this sweep need to contain the same number (or 1, which will than be expanded to that number)

==========

Tetanus (**DACx, num1, num2, num3, numset2 [, num]**)
and
Poisson (**DACx, num1, num2, num3, numset2 [, num]**)

Generate a tetanus stimulation train with a fixed frequency or poisson distributed train series using the pattern as defined in **Stimulus**.

[DACx] Hardware dacchannel on which this stimulation will be given (DAC1, DAC2, DAC3, DAC4)

[DAC1 DAC2] If you define two channels the stimulation will be delivered on two channels as "digital" pulses.

num1 startmoment of the stimulation in the trace in seconds.
You should only use as different value than 0 if the last value (num) is 0 or 1, otherwise there will be a stimulation gap in your stimulation. When you give a repeat value, the whole pattern of the stimulus will be repeated.

num2 duration of the DAC stimulus trace in seconds
DAC traces must have the same length!

num3 frequency of the stimulation in Hertz. (Tetanus)
mean frequency of the Poisson distribution (in Hz, defines labda parameter).
On each time point **Stimulus** will be inserted

numset2 **num**: single value: amplitude factor, with which **Stimulus** will be scaled (= multiflied).

[**numa numb**] If you give two values for the intensity the first one is the start of the ramp and the second one is the final value!

MCC If you define the amplitude as **'MCC'** then we will use the MCC Pekel DAC unit for the amplitudes (useful in combination with the NPI stimulator).

Optional:
num4 repeat count for the given pattern, range: 1 to inf.
Default is 1.
Inf inplies repetition until stopt

==========

Puls (DACx, 'group-of-numbers, group-of-numbers');

This command defines a trace on DAC nummer DACX.

DACx = defines which DAC channel is defined

The first string command is obligatory and determines the length of the trace.

It contains an (endless) list of commands that are in the form:

number1 number2 (number3), number1 number2 (number3),…….

groups of n numbers (2 obligatory) are separated by a comma.

The first two numbers form duration/amplitude pairs. The first one is duration in ms and the second one is in amplitude units that were defined for the DAC channel (mV?)

If the third number is also is defined the protocol will have an amplitude that linearly changes from number2 to number3 over the duration defined (ramp, take care of sufficient resolution!)

When any of the numbers 1,2 or 3 is a set of numbers the protocol will define a series of sweeps that must have the same duration (important if the variation is in number1, than the complement of variation needs to be in another number1 in the list.

If you put a symbol % anywhere within the [ ] then amplitude in this part of the stimulus definition is interpreted as a percentage; 0% and 100% are defined in the EPpanel (meant for the Stim.

Optional parameters are not yet defined

==========

Digs (DACx, duration, 'group of numbers, group of numbers, ');
This command defines a trace on DACx with total duration of duration in ms putting a digital pulse (amplitude 5 V) on the output. It also calibrates and scales the channel between stimulus -5 and +5 Volt, make sure you do not intermix your digital and analog outputs, it could destroy the inputs of the equipment you drive.

DACx = DAC output to use (DAC1, DAC2, DAC3, DAC4)

duration = total duration of sweep in ms
DACrate is used for the sample frequency

Instruction string = consists of a quoted string of pairs that define location and duration of the digital puls.

The meaning of peach pair is:

number = **[moment duration]**

all in ms.

==========

Wave(DACx, duration (ms), Offset, Instruction string);
Can be used to create a superimposed series of sines/cosines/alfa functions

duration = total duration of sweep in ms (the same for every sweep!
DACrate is used for the sample frequency

amplitude offset = zero value of amplitude output on this channel.

Instruction string = consists of a quoted string of instruction sets that define a sine, alfa or other function. Format:

'sine (par,par…), sine(par,par,par…), alfa(par,par…)'

with:
(every parameter can define one value or a series, only one or one length of series allowed!
sine (freq, amp, [*start*, [*len*, [*phase*]]])
freq = frequency of the sine wave
amp = amplitude of the sine wave
start = moment in the trace where wave will start (default 0)
len = duration of the wave in ms (default inf = as long as the trace)
phase = start phase of the wave in degree 90-360), default = 0

alfa (freq, amp, tau, [start, [len]])
freq = frequency of the exponential
amp = amplitude of the exponential
tau = decay time of the exponential
len = duration of the wave in ms (default inf = as long as the trace)

chirp (freq1,freq2, amp)
freq = frequency of the exponential
amp = amplitude of the exponential
tau = decay time of the exponential
len = duration of the wave in ms (default inf = as long as the trace)

==========

Barcode(DACx, stepduration in (ms))

Generate a barcode on a DAC channel (to be used to later identify a stimulation o in a continuously running EEG run.

[DACx] Hardware channel to use for the artefact pulses.

[num1] duration of each step in the code
fiurst step: -10Volt
second step 0 Volt
Following steps (asume barcode is 329:
--step to 9
--step to 2

--step to 3

==========

Artefact(DACx,num2)
Generates a puls in a specified channel at the start each stimulation pulse in a tetanus/poisson traina in all other DAC channels. Link to Stim needs to be imp-lemented!)

[DACx] Hardware channel to use for the artefact pulses.

[num2] Duration of the pulses in ms (make sure they are long enough if you sample this channel with the ADC. If your DAC run is a single one, you can get the timing also out of the DAC channel (for repeating DACs that becomes more difficul, not yet implemented).

==========

DACHold({valset,num});

Defines the values to hold ones the DAC is stopped (after continuous mode only!) Generates a puls in a specified channel at the start each stimulation pulse in a tetanus/poisson train in all other DAC channels. (Link to Stim needs to be implemented)

valset Give a (set of) value for each hardware channel (even if not in use!) to be given once the DAC is forced to stop (break in continuous mode). If not specified the DAC will not get values.

num bit on which to generate the safety pulse for DAC stop (1:8 = P2.0--P2.7, exclusive bit8.

So valid forms for a 4 DAC system are:
DACHold([Vh1,0,0,0],5)
Will set DAC1 to Vh1 and the other for to zero once you forcefully stop the trace EEG, stop puls will show up at bit 5

==========

StopStim(DACx, num2);

Activates the Stop Knob to be used in Threshold testing.

[DACx] DACchannel to be used for calculating the threshold value at which the run was stopped (it helps if you programmed a ramp on that one!).

num2 Bit in word 3 (P2.0 --P2.7), indicated as 1..8 to use for generating a relais close signal. Mostly bit 8 is already used to trigger the ADC/DAC.
This bit will go high once you push the stop knob. Then the DAC channels will be set to their Hold values and then the puls will be set back to zero. this is for the time being the best way to safely break the DACs during stimulation (be aware that you do not zero a holding potentiual! use Vh1 or Ih1)

==========

Rampup(num1, num2, num3);

Will program a slow ramp up in a tetanus/poisson trace. Can only work if the intensity is programmed via the MCC device.

num1 Delay to the start of the rampup (s).

num2 Update Interval of the rampup (s). Donot overdo this only as the rampupo is interupt driven and has to run in the background of the running DAQ system. Smaller than 0.05 does not make much sense.

num3 Length of the rampup (s). Delay to the start of the rampup (s).

The Rampup command when Run will activate the Slow-Stop Knob in the MCC panel. You can use it to give the inverse rampdown to switch of the stimulus. It will only stop the DAC, not the ADC!

If you started the Run with a runup and want to stop it immediately, you get the request for a slow stop (Cancel, Immedate stop or slow stop.

==========

Bitdef(num1);

Defines a bit toggle knob in the EEG panel (for test purposes only)

num1 bit to set in word P2.0--P2.7,(specify as 1..8, but bit 8 not allowed = DAQ trigger).

This command will Activate the BitToggle Knob in the EEG panel. If you defined it in the Basis protocol, you can then Toggle the bit with this knob at any moment and test whatever you want.

==========

Bitset(num1, num2);

Will program a slow ramp up in a tetanus/poisson trace. Can only work if the intensity is programmed via the MCC device.

num1 bit to set in word P2.0--P2.7,(specify as 1..8, but bit 8 not allowed = DAQ trigger).

num2 Update Interval of the rampup (s). Donot overdo this only as the rampupo is interupt driven and has to run in the background of the running DAQ system. Smaller than 0.05 does not make much sense.

==========

Plotter(num1, text);

This is a very extensive statement that is used to display data in multiple ways in many windows. It is easy expandable to add options that you like.

num1 where shall we plot it? Use a number that represents any of the locations defined in the above window lay out. At this moment you can maximally define 6 different plotwindows to be used at any moment at the same time (scream if you 9really) need more!

text supplies all the qualifiers needed to determine what to plot. See explanation at Display

==========

ClearNow,(numset)

Will clear the windows directly (so during generation of the protocol

[numset] Set of windows to be cleared (if not specified, all will be cleared)

==========

Clear, (numset)
Will clear the windows just before execution of the protocol

[numset] Set of windows to be cleared (if not specified, all will be cleared)

==========

NoEEG
A simple way to make sure that this porotocol will never be run nwhen the EEG is running.

==========

Filesystem(text)
Define defaults for data file system:

text text to be used as the first part of the file name (normally we take user name = recognize your files!)

==========

ADCrate(num1);
Set the ADC sample rate in Hz

num1 defines ADCrate in samples per second

==========

DACrate(num1);
Set the DAC sample rate in Hz

num1 defines DACrate in samples per second

==========

Samplerate(num1,{num2});
Define Samplerate of ADC & DAC with one statement

num1 if the only parameter defines equal ADCrate and DACrate in samples per second;
num2 for 2 parameters, first one is ADCrate, second one is DACrate

Remark: Samplerate, ADCrate, DACrate are evaluated in the order they are encountered and each next one overwrites earlier ones. So you can redefine the sample rate in a protocol.

==========

Interval(num1, num2, num3);
Define intervals between sweeps

num1 -- Time gap between start of two sweeps in the set (in s)
num2 -- Extra time to add to the first sweep in a set, if it follows one.
num3 -- Extra time gap after the last trace in a set (default is = 0)

==========

Time(num1)
Defines an absolute protocol time after the first one in a set.

num1 -- absolute time of this sweep in respect to start of the protocol (seconds)

==========

Delay(num1)
Hardware delay needed to start the first sweep of a series.

num1 -- time in ms between start of the sweep and start of ADC & DAC sampling

==========

Beep(num1)
Gives a beep at the end of a protocol

num1 -- nr of the tune to play. Convenient for long lasting protocols for which you are waiting to finish.

==========

Average(num1,text);
Define that you want to average this protocol and indicate over how much and in which mode you want to do it.

num1 -- if in Run (mode) than define number of traces to be used for the average and how to do it
text "**pertrace**" calculate mean of the same trace in one go
"**perset**" calculate mean set by set (see description of time line for explanation). (plan is also to do per protocol, but that is a bit more complicated)

==========

WriteOn
Makes sure that data measured in this protocol will always be written to disk. Does not affect your setting of auto save

==========

WriteOff
Makes sure that data measured in this protocol is never written to disk. Does not affect your setting of auto save

==========

WriteModus(modus);
Makes sure that data measured in this protocol is always/never written to disk depending on Modus, this is the version you can call with a parameter.

modus 1 = WriteOn
0 = WriteOff

==========

EqualDAC(target DACs, source DACs);
Opens the option to equate a DAC stimulus output to an already existing one;
you can define a list:
e.g.: EqualDAC([DAC3 DAC4 DAC2],[DAC1 –DAC1 -DAC1])
will make DAC2 and DAC4 equal to the inverse of DAC1 and will make DAC3 equal to DAC1

DAC3	will be equal to	DAC1
DAC4	will be equal to	DAC1
DAC2	will be equal to	DAC1

==========

SWDdetect (num1, num2)
Prepare the SWD neuronal network algoritthm to run. It will only run during recorTrace mode and use the amount of data that you display in the show interval.

-- **num1:** ADCchannel on which to run the detection algorhithm. It needs to be an existing channel.

-- **num2:** ADCchannel you want to use for overwriting the outcome of the detection algorhythm. Will be zero if nothging to detect, will be high if detected.
use 0 if you do not want this output, otherwise num2 needs to be an existing channel, that is overwritten.

You have to prepare the action in the protocol and can than switch the use on and off in the EEG panel.

==========

Electrodes(number1, number2,number2);
This statement allows you to define a set of measurements with different electrodes (worm data collection). You can define a set of sets, using similar rules as hold for the standard sets (thus, either the same number of values or single ones). You can combine this option with all other set definitions, to always make a rectangular set of stes (though numsweeps x numsets x channels (x singles). The name of the set will be expaned to **"setname(stim1-stim2)"**
-- **number1:** list of values for stim1
-- **number2:** list of values for stim2
-- **number3:** list of values for ref
Parameter(name = values);
This statement allows you to define a set of identical measurements to be performed when one defined parameter goes through a list defined with this statement. You can combine this option with all other set definitions, to always make a rectangular set of stes.
(thus: channels * singles * numsweeps * numsets)

Values must be expandable to a list of numbers that will be used. The name of the set will be expaned to **"setname(par)"**

Calibrating the amplitudes:

ADCscale(number, [a1 a2], [b1 b2], [c1 c2], [u])
number -- channels to which this scale applies 1:32 etc.
2 numbers -- a1 a2: sensor range
2 numbers -- b1 b2: inputrange [-10 -10], determines hardware gain 1 - 2 - 5 - 10 - 20 - 50 - 100
2 numbers -- c1 c2: units range the meaning in user units of the values [b1 b2]
text -- u: Units (e.g.mV)

ADCchannelnames(number, list of names)
A fast way to define names of ADC channel
number -- ADC channels to which this scale applies 1:32 etc.
texts -- list of strings that represent the names of the channels indicated.
e.g ADCchannelnames(1:5,EEG1,EEG2,EMG1,EMG2,Accel);

ADCchannelfname(number, formatstr)
Define names of ADC channel using the standard MATLAB string formatting and numbers
number -- ADC channels to which this scale applies 1:32 etc.
texts -- formatstring like: Chan-%u indicated.
e.g ADCchannelnames(1:3,EEG-%02u); will give EEG-01, EEG-02; EEG-03…

DACscale(number, [b1 b2], [c1 c2], [u])
number -- c1: channels to which this scale applies
2 numbers -- b1 b2: outputrange (ADC) in Volt [-10 -10], determines hardware gain 1 - 2
2 numbers -- c1 c2: units range the meaning in user units of the values [b1 b2]
text -- u: Units (e.g.mV)

DACchannelnames(channels, list of names)
A fast way to define names of DAC channel
number -- DAC channels to which this scale applies 1:2 etc.
texts -- list of strings that represent the names of the channels indicated.
e.g DACchannelnames(1:2,Stim1,Stim2);

DACchannelfname(number, formatstr)
Define names of DAC channel using the standard MATLAB string formatting and numbers
number -- DAC channels to which this scale applies 1:32 etc.
texts -- formatstring like: Chan-%u indicated.

Calibration (text1, text2)
Predefined special calibration sets
text1 -- "Axon", "HEKA", "EPs", none
text2 -- VC or CC

A few words about calibration:

MATLAB is so flexible in calibration that about twice as much is possible as you would ever demand. In principle everything can be used. For input we have three values for calibration:

SensorRange: the range of the sensor hooked on to the input. It is very important that the relevant sensor range is matched to the input range of the ADC. So if you measure a signal that has values between -5 and +5 Volt. Make sure that your input range is also in that range. In general the input range should be one higher than the sensor range. As we assume them coupled we scale the sensor range with the input range.

InputRange: The actual Voltage range of the ADC (in real Volts). It is hardware determined. If you select a non-existing range you get an error message. All channels need the same to avoid slowing down the system for settling time reasons. Most commonly set to -10 +10 Volt.

UserRange: The range of values that determines what the meaning is of the values of Inputrange in engineering coordinates (so the real values in user units). Itb translates sensorrange to world values. As sensorrange is (by us) strictly ncoupled to input range the whole scaling is settled in this way.

There are many situations where scaling is more or less fixed and you do not have to worry. Once it is set OK your values will always be OK:

DAC output when you measure EPs is assumed to drive the UvA stimulator with a range between -2000 and 2000 uA (for voltage between -10 and 10 Volts). To simplify life you can scale the input with a simple “pre-ampliufier” gain that defines the ampolification for -10000 to 10000 mV as input range.

When using the Worm boxes (32 and 16) stimulastion takes the standard Voltage output and input takes into account the 100 times amplification that is fixed in the boxes.

When driving the Axon (HEKA) amplifiers from DAC and to ADC with the gain read out to one of the ADC channel you only have to incorporate the definition in the Calibrate statement; also whether you are in VC or CC mode (will be read from the amplifier as well).

User specific panels

Definition of various user/experiment specific panels can be added into the menu. They will activate a panel that allows you interact with the hardware mostly in a specific setting, such as Voltage Clamp, Evoked Potentials EEG, Soft Clamp, Defining configuration of the 32 electroides in the nerve box etc. Panels can only be activated in the main part of the protocol, not locally in protocols

The aim is to easy expand on those panels for specific xperiemnts. Cutrently implemented are:

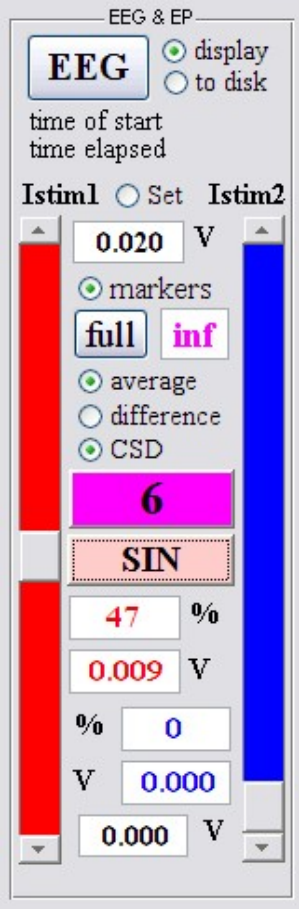
1) EPpanel
-- Panel for evoked potentials (stimulation/ monitoring)

2) EEGpanel
-- Panel for continuous r4ecording with or without stimulation

3) Wormpanel32
-- Panel for evoked potentials in the 32 channel box (stimulation/ monitoring)

4) Wormpanel16
-- Panel for evoked potentials in the 16 channel box (stimulation/ monitoring)

5) VCpanel



-- Panel for Voltage Clamp, Current Clamp and Soft Clamp. Complete integration with the axon amplifier, partly integration with the HEKA amplifies

EPpanel(string, DACx, gainADC)
This statement invokes the evoked potential panel:

-- **string**: protocol to run when you hit the start button in this panel (the actual protocol name is illustrated!)

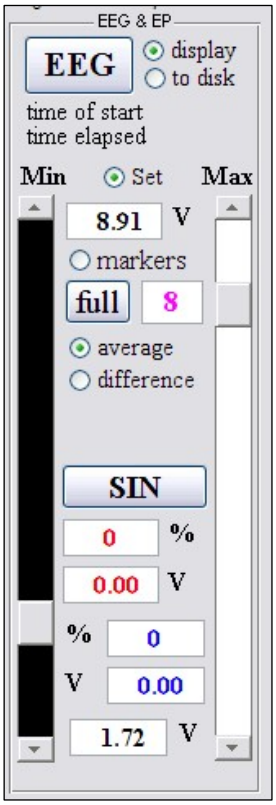
-- **DAC1/2/3/4** DAC channel used to define units and values in this panel

-- **pregain** indicates amplification before you enter the ADC (use gain in Record definition to set the actual ADC gain.

The top part of this panel is used to start EEG recordings (see below)

In the EP panel you can define two values with a slider, the left (red) or the right (blue). These values can be used in the protocol as Istim1 resp Istim2. They match in absolute value with the absolute numbers given in red and blue. The sliders move between a minimum value and a maximum value. You see the absolute level as well as the percentage value, where 0% is the minimum and 100% is the maximum. The overall range of the values is collected from the DAC scaling that you gave in the definition. (as are the units, mostly V, mV or uA). Default is our own +-2000 uA current stimulator, or automatically the +-10 Volt in the “worm” boxes.

You can set the range to “full” range by hitting the full knob. You can be helped to set the min and max values as explained below.



By activating the Set button your sliders do not show the Istim1 and Istim2 value but the min and max of the percentage scale. In most cases we normalize stimulation so that threshold is at 0% and saturation is at b100%. This makes it easy to predefine IO protocols etc. Once you have set the Set box, the sliders will change from colors to black for minimum and white for maximum and you now set with them the min value and the max value. (the interface might have some needs to be improved!).
The values of the original Istim1 and Istim2 are whenever possible retained if the new min/max scaling allows it, so that you will not be surprised by overstimulation. There are a few more option in this panel, that are useful for experimental interaction and for getting your response quickly in their working range:

-- **markers** (will mark the individual sample points (not active yet)

-- **difference** (will calculate the first order spatial derivative of the channels (thus the difference between all adjacent channels instead of the monopolar measurement. Be careful if this is exactly zero, because it could mean that your channels go in overload!! And is irreversible!

-- **CSD** calculates the second order spatial derivative between the cahnnels (always with a 1 -2 1) scheme.

-- **average** will do online cumulative averaging. It will recalculate the mean and standard deviation of the mean after each sweep and keep those values. You can continue for ever (and save the data to file if you like). Saved data contains the last sweep and the mean and the std. Individual sweeps are not kept. Averaging will continue until you reach the number in the pink box. If the pink number is **"inf"** then it will run forever. In the pink **"ticker"** there is a number that indicates how far we are.

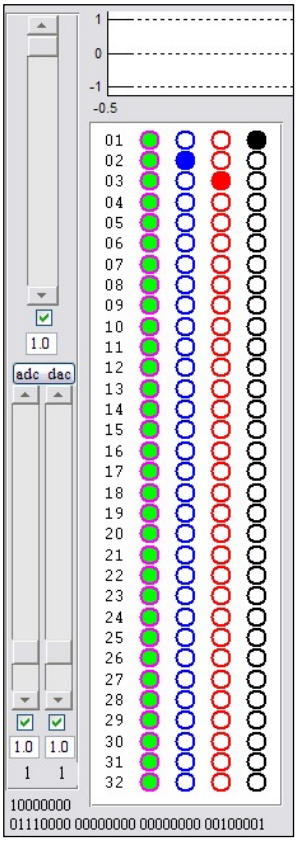
There is a pink ticker active whenever the program is actually collecting data. Flashing means you are running. If you want guaranteed timing of your data collection, you should not interfere with your program during the sampling time.

EEGpanel(string, number1, number2)
Defines EEG section in EP panel.

-- **string:** protocol to run when you hit the EEG button in this panel.
starts continuous EEG recordeing to screen or disk.
-- **number1:** fraction of EEG to display each time (e.g. 1 s of a 5 s screen

-- **number2:** Time in seconds to store in one diskfile (of an infinite series of numbered files). Be carefull to make them bigger than 1 Gb.

0Wormpanel32 and Wormpanel16 define the EP panel and add the electrode selection window in the left upper corner of the screen (see next page for further elaboration on electrode selection panel.



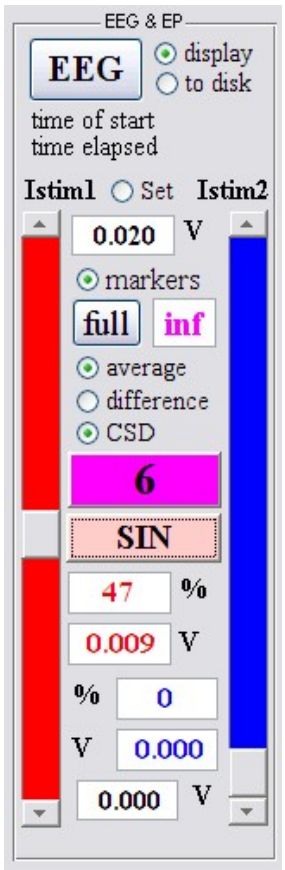
Worm panel assumes you have mV as ADC scale, and +-10 V as DAC scale. In accoirdance with the hardware pregain is set to 500

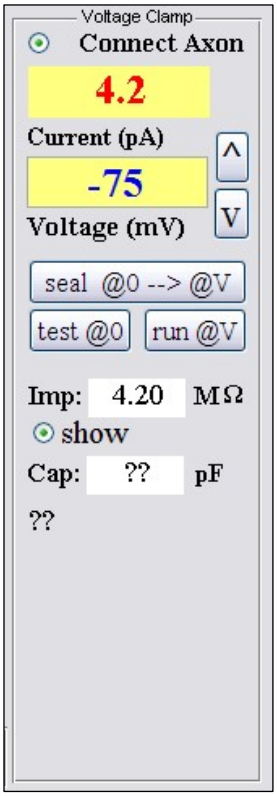
Wormpanel32(string) sets the electrode selection for 32 channel box

-- **string**: protocol to run when you hit the start button in this panel

Wormpanel16(string) sets the electrode selection for 16 channel box

-- **string**: protocol to run when you hit the start button in this panel





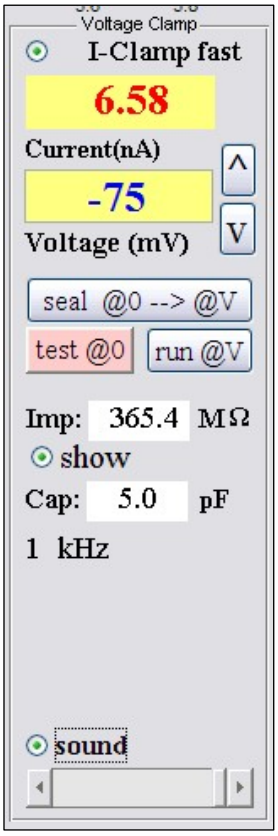
VCpanel (string, channels, threshold, [p1 p2 p3 p4])
Testpanel for voltage clamp applications

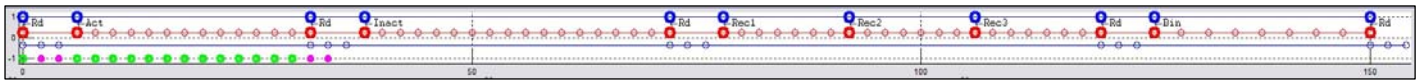
-- **string:** name of protocol to use for impedance measurement and seal/run/test trials

-- **channels:** channels to use for Axon interface (13:16) for cap, mode, filter, gain

-- **threshold:** impedance value where sealmode will switch from test to run!

-- **[p1 p2 p3 p4]:** moments in time to use for impedance measurement (R = dv/di)

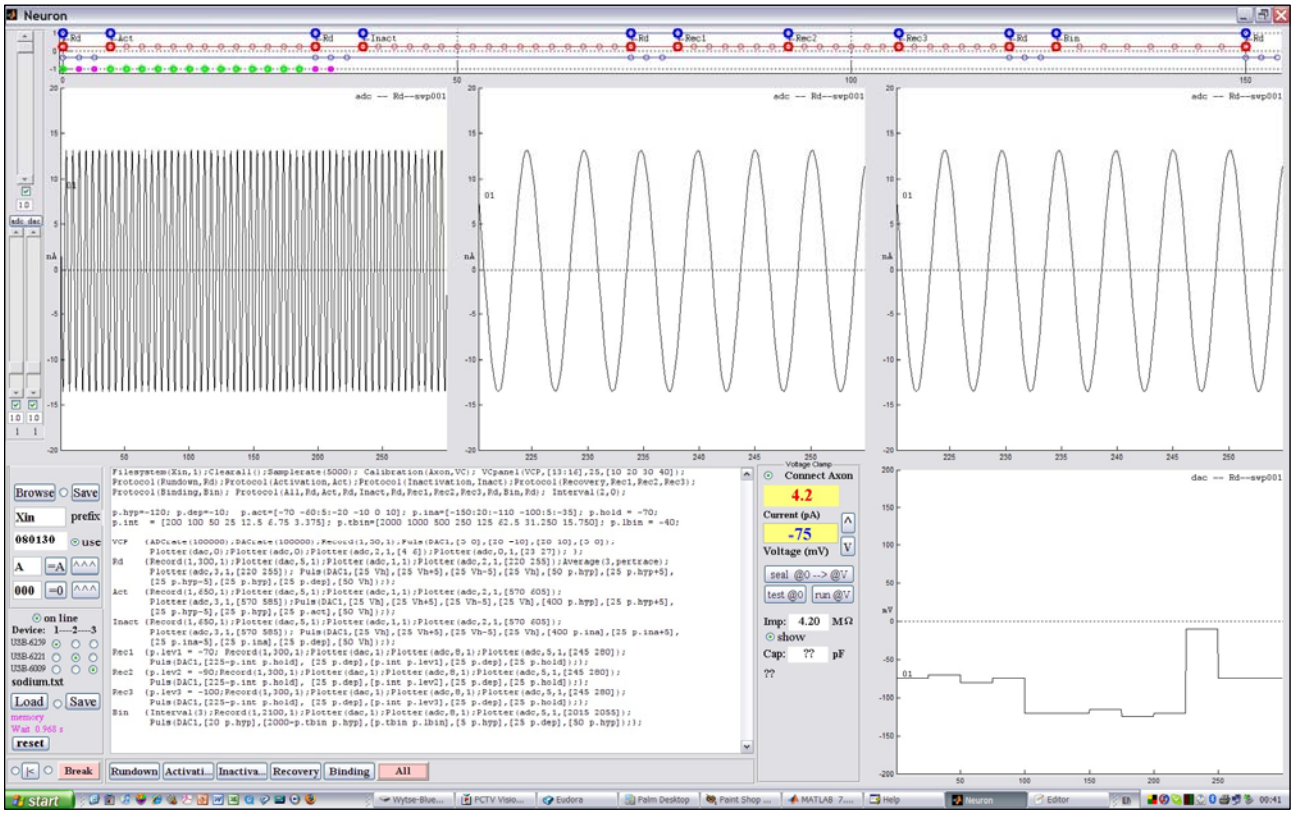




Softclamp(number1, number2, number3,number4.number5)
Define the parameters for the softclamp system
-- **number1**: samplerate for the softclamp system in seconds 9do not overdo! 0.1 is fast

-- **number2**: gain of the softclamp system (be aware of instabilities)
(this value is not dimensionless!)

-- **number3**: filter time of the softclamp system (half decay time in samplepoints)



Special additional parameters:

You can give a drugname and drug concentrations with every measutrement. If You have these numbers at NaN (not a number they will be construted out of the full directory/filename in the following way:
Drugconcentration is the first numerical value found between normal brackets (), searching backwards in the string. **Drugname** is the word found between the (of concentration and the previous \ So naming any one of the higher level directories '**CBZ(100)**" will extract CBZ as drugname and 100 as concentration value. The latter can be used in calculations (formula).

Here are the changes additions:

The XY command now logs all individual resampled IV curves from the traces that you display 9not the average any more, but it is very easy to produce in EXCEL.

The merge_set command will merge traces in different files but keeps the time information (it was there, but did not merge the DACs yet).

This is the easiest way of using it:
Load the first datafile with the LOAD knob, than add all the files you want to add with the ADD knob, They show up as a set. (At the moment it is your job to get the order right.
Then hit the MERGE_SET knob and the data will be merged and displayd (it show up in the standard NEW directory 9generated if not available) and will have a name composed about the first and the last one of the original files. For a while the gaps in the time are indicated with blue and red lines. To prevent false triggers the data in that period is equal to the last point of the previous file. For continuation reasons I fill the gaps, so do not make it too long!

I also implemented the DAC_Detect command that does similar things on DAC chanels as the Event_Detect does on ADC channels (but DACs are artifact and noise free). The options are simpler for that reason. Provide channel and crossing level (meaning equal to the ones in Spike detect). You can see the result with the new DACon knob in the Spike Window).

I also created the macro to analyse the step protocol.

For this reason I needed an option to run the formula several times over the same trace (all the pulses are in the same trace, normally a formula only produces 1 result and not 7.
The following additions were made (also useful for our spike questions!):
You can add a "repeat(n)' option to the List command, which will run the formula the requested number of times over the same trace. You can use the VERY SPECIAL n parameter in the formulas to do calculations based on the number of times we run the formula. (I might implement it slightly more versatile later).

I also needed a way to run this formula over all the sweeps we have (The extracted pulse sets). This is done with the FORI command with 3 parameters:

Fori (name, values, macro)

It will run the macro macro as many times as you give values in the parameter value, assuring that variable name will have the desired value.
So in the example I run the macro over the sweeps, assuring that each time a different sweep is analyzed.

Analysis using macros (scripts).

Describe general set-up of macros, loading, saving, running.

==========

All Instructions are separated by a comma (**,**).

All text commands need to be given in single quotes: (**'text'**). You may leave them out is the outcome is really unambiguous (but be sure! these are difficult errors to detect)

==========

You need at least one of these commands in your macro file, if you want to do something:

Macro {'knobname', instruction1 , instruction2, etc}
This instruction will make a recognizable knob with '**knobname**' in the command panel so that you execute it by simply pressing. The Knob will turn red, which lats for as long as the macro executes. You can only run one macro at the time.

De aware of the curly brackets!

You can maximally define 16 knobs in the list that result in direct action

When you hit the knob all instruction between the brackets will be executed (they can be nested to any depth).

==========

Defining parameters for later use:

p.somethong = 23,

any statement with an '=' will be executed in the workspace. Restrict your variables to the p.something type in order to prevent problems with my internal variables and make sure every routine can see them.

==========

Other procedures are of the general form:
procedurename **(par1, par2, par3)** **{*instruction1*(*parameters*,..,..), *instruction2*(*parameters*),..}**

-- ***procedurename*** is the name under which you can call this function in any other Instruction list.

--**par1, par2, par3** are variables that can be given to the procedure by the caller. They are given "by value". It is your duty to look whether the resulting texst after substitution in the macro is a sensible instruction (I try to do it, but I also understand the logic, so I do not know which errors to catch!)

-- ***instruction1*(*parameters*)** any list of instructions that perform some kind of function on the data. The list can be as long as you like. In general it is easier to debug short procedures will clear functions!

-- ***parameters*** Everything between the outer curly brackets will be executed.

==========

Each instruction is defined as:

Instruction (parameter, parameter, parameter, options)

Normal brackets have their normal meaning, they may be nested.

All text parameters should be placed between quotes ('), even though this is not always necessary, it can prevent difficult to trace problems. Also they may be nested.

There is a set of "common" commands that can be used in most Instructions, (if indicated in the manual). They have a different format (as they are paired (instruction word and variable. You can place them anywhere in the parameter list, but you make the code more readable by placing them at the end. The words reserved for these instructions should not be used anywhere else in the instructions as they will not be recognized in any other context. Thus

name (par1, par2, par3{, opt1, opt2, opt3, ..., optional instructions, can be placed anywhere} e.g.:

Instruction (1, 2, [10 12 13], scalexy([x0 x1 y0 y1]))

Instruction can have any of the following forms:

-- assignment

-- special

The pair **'scale'** and **[x0 x1 y0 y1]** should be next to each other (the scale command defines the meaning of the next parameter(set)), can be placed anywhere (but best is at the end) and can also be omitted then [-inf inf -inf inf] will be used. The meaning is also always the same. The parameter is not always needed, at this moment there are no options which use more than one (set of) parameters.
Here are the *options* (list will be extended over time):
---

clear	something will be cleared (context dependent, default will be no clear)
save	something will be saved (context dependent, default will be no save)
next	
fast	runs the formula without displaying everything (you can manually overwrite it with the off knob in the event panel
weight	used to define weighted Template matching
sweep	use sweep (context dependent, default will be no sweep)
offset	use offset (context dependent, default will be no offset)
contours	use contours (context dependent, default will be no contours)
device (n)	use hardware device n for this operation.
bursts (r)	make difference between bursts and spikes in spike detect using r as parameter for...

repeat (n) use a repeat count of n for this operation (list so far)

wait(time) depending on time"
-- time not defined, take it from the desktop
-- 0 skip it
-- number, wait number seconds before continue 9after most significant result)
-- inf, ask standard go, wait abort question

sortxy (n) sort the xy result columns generated by the Formula according to column n

---

logx will draw the x-axis on a log scale.
logy will draw the y-axis on a log scale.
logxy will draw both axes on log scales.

---

nologger Suppress logger for this instruction (context dependent, overrules the current logger state and will only have effect if logger is on)

---

window(n) Defines window to be used (take a number from the window list).
Precise meaning is context dependent, you might use a list of windows

---

list(n) Use list n for this operation. Default will be list 1

---

mets (numset) specify measurements to use (count as numbers)

swps (numset) specify sweeps to use (count as numbers)

chns (numset) specify channels to use (count as numbers)

index (mets, chns, swps)
shortcut to define mets, chns, swps with one statement.

---

mask Use mask defined in the Mask & Events GUI.
mask(n) Use mask n for this operation.
any number (1 .. inf): defines which mask in your list of maks should be taken.
Remember the special meanings of:
-1: means no mask at all, incorporate all data.
0: means use only the non-excluded data (based on the last exclusion criterion used in the tamplate matching).

---

show(n) something will be shown (context dependent, default will be no show)
in some situations a number can be given to specify whar will be shown, examples are as in the XY command:
-- 1, show raw traces
-- 2, show constructed mean
-- 3, show both
exclude(r1,r2) Exclude trace from before to after (triggerpoint)

fitspecs give details for fit procedure (type rtc...)

fullframe use the full time zoom window, just as can be set by fullscale.

frame use actual timeframe from the zoomwindow.

'range' range on which to operate.
This command follows the complex definition:
([a b]) = use the range from a to b
([a b], cc, dd) = use range a to b
repeat cc times separated by dd time units see {unit}

limit(v1,v2) set limits for some parameter to a range p1... p2 (e,g,fitting)

---

scalexy ([x1 x2 y1 y2]) standard option for graph x-y scaling

or

scale ([x1 x2 y1 y2])

scalex ([x1 x2]) standard option for graph x scaling (Redundant, identical to scale with 2 parameters)

scaley ([y1 y2]) standard option for graph y scaling

---

source (window, text) define the source of an operation, most often this is obligatory! The source will be drawn using text in window.

result (window, text {, window, text{, window, text}})
define the target for the result of an operation, very often you can define a list of targets, that will be used in the "original" plot mode. The targets will be drawn using text in window. You can call result multiple times.
(Will be outdated, use results!)

Macro commands

==========

Evalm (text)
-- use MATLAB eval to analyse text (no action taken, for test purposes only)

==========

Evalp (text)
-- as above, but after adding a ";" to prevent display.

==========

VerboseOn
-- Switch display of intermediate results (graphs and/or tekstlines) on. For test purposes.

==========

VerboseOff
-- Switch display of intermediate results (graphs and/or tekstlines) off. For testing purposes.

==========

OriginalOn
-- Switch plot original and auto original on

==========

OriginalOff
-- Switch plot original and auto original off

==========

FullOn
-- Switch zooming to full and autofull to on

==========

FullOff
-- Switch off auto full

==========

ZoomOn
-- Switch zooming mode on

==========

ZoomOff
-- Switch zooming mode off

==========

EventSaveOn
-- Automatically save events in the event file, after each relevant action

==========

EventSaveOff
-- Switch automatic saving of events off

==========

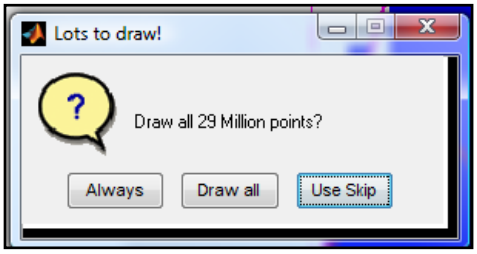
SkipDrawOn
-- Switch skip mode on (draw only a reasonable number of points on the screen)

==========

SkipDrawOff ({num})
-- Switch skip mode off

-- **num** number of points to be skipped when drawing. use numbers>0
If you use a number <0 and have detected events (spikes, mini's) than the number will be interpreted as the number of pouints to be drawn before and after the event detection moment. This can speed up drawing enormously, while still giving you the relevant information.

If in the **all:** mode you want to draw too many points (>5.000.000) on the screen you get this window.
There are three possible answers:



-- Always, from now on switch to **Draw all** mode and never ask this question again, until you explicitely switched back to the Use skip mode
-- Draw all, draw all the points (this time), be aware that this might take a while and depending on your computer, could result in an out-of-memory error.
-- Use skip, be fast and go to skip mode
Close window if it stayes open for some reason.

==========

SkipFitOn
-- Determines what to do on fit errors (see the fit error box). If you set this mode on, than fit errors will not be displayed, but the answers can not be generated, so there is a note in your log file. Pleasee be carful with this option, it is best to really solve this type of errors as it indicates that there is something wrong in your procedures.

==========

SkipFitOff ({num})
-- Switch skip mode off

==========

TriggerOn
-- Set Trigger display mode on

==========

TriggerOff
-- Set Trigger display mode off

==========

Clear (winset)
-- Clear windows involved in displaying any type data

-- **winset** set of windows that will bw cleared and removed from viewing.
Remeber you have only 10 windows, do not leave windows in the background open. They might interfere with interactive work you want to perform.

==========

SetTime (text)

Define the scaling of the time axis (as with the left balls in the scaling window).
Depending on text:

'ms'	milliseconds
'sec'	seconds
'min'	minutes
'hrs'	hours

==========

Setframe([real1 real2],{text})

	Set zooming to full (prevent out of scales!)
[real1 real2],	define start and end of the x-axis draw range (set in zoom window)
text	optional text defines the overall time axis scale (as command above)

==========

SetPause (number)

-- Define pause in formula and fit so that you can watch the result for the given time.

-- **number**	Delay in seconds after most relevant calculations. This value overrules the pause value in the scale window (it does not reset it!) If you set the value to **inf**, you go to step mode and will get the continue question (see below step command)

==========

Pause (number)

-- Instantly wait the number of **seconds** here.

-- **number**	Delay in seconds to wait instantly.

==========

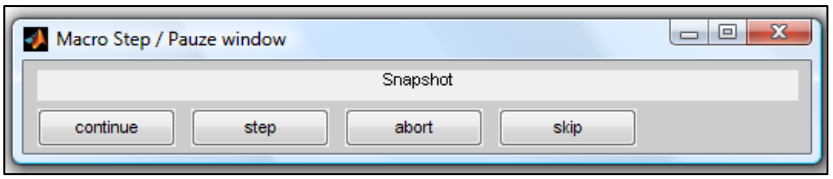
Stop

-- stop the macro unconditionaly.

==========

Step,

-- set the instruction interval to inf, thus forcing interaction with step mode:
--Top line indicates the macro command that will be executed (if relevant with filled in parameters).



Knobs:

-- continue,	switch back from step to continuous mode
-- step,	do one more step execution (the shown command).
-- abort,	get out of the macro as if you abort.
-- skip,	skip this macro and step to the next command.

==========

Ffile

-- Put the name of the current datafile in Logger

==========

Ffull

-- Put the full file and directory name of the current datafile in Logger

==========

Fcell
-- Put the cell name of the current datafile in Logger

==========

Fdate
-- Put the date of the first node in Meting in Logger (year-month-day)

==========

Ftime
-- Put the time of the first node in Meting in Logger (hrs:mins:sec:ms)

==========

Fsecond
-- Put time of first meting in seconds since midnight in Logger (rounded to 0.1)

==========

Drugs
-- Put the name of the current drug and concentration in Logger

==========

Capacitance
-- Put the value of the capacitance of the cell (in first node) in Logger

==========

Temperature (Any tekst)
-- Put the value of the temperature of the bath (in first node) in Logger

==========

Annotate
-- Put the tekst you give value in Logger at the current location

==========

Music (num)
-- Will play sound at the end of each measurement for any number of nr

-- **num**	0:	chimes	(windows sound)
	1:	ding	(windows sound)
	2:	clickers	(windows sound)
	3:	gong	(mathlab)
	4:	train	(mathlab)
	5:	chirp	(mathlab)
	6:	splat	(mathlab)
	7:	laugh	(mathlab)
	8:	halleluja	(mathlab)

==========

SetWindow (num, [x0,y0,dx,dy])
With this command you can change the scaling of any window, or define a new one.
If you run it without any parameter, all windows will be **reset** to the ones as defined in the manual.
-- num -- window to reset (number between 1.. 42). You can define higher ones!
-- [x0,y0,dx,dy] -- scaling in normalized units, the full GUI has [0 0 1 1]

==========

Draw ({nlist}, {window([n1 n2 n3 n4])}, {text1{,text2...{,... }}})
-- Draw is a multifuncional command that can be used to redraw the GUI in order to show the last results.
You can also draw something new and you can use it to define the 4 specific windows on screen. here are the ways to do it:

Draw in its simplest form redraws the screen

Draw (list(n)) will draw the list of nodes described by list

-- **list** the list of nodes that will be redrawn. Use the List command to define a list of nodes.

You can define the content of the 4 specific display lines in the Gui, in combonation with the windows that you want to use. Use the optional window instruction to attain that goal.

-- *text, text,...* quoted textxs that you want to insert into the display commend lines. It is yoiur responsabilities to give as many textx as you need (0 windows are ignored)

Draw ({nlist}, window([1 2]),'adc(1)','adc(2)/setx([10 20])', {save})
will draw the list of nodes described by list
If you define nlist this list will be shown. if you omitt the current picture will be redrawn

-- *window([n1 n2 n3 n4])*
defines which windows to use, use 0 for the ones you do not need to define
-- *save* Save this setting instead of resetting back to the original situation

==========

RunDraw (num1, num2)
-- Display a set of data
-- **num1** -- list to display
-- **num2** -- (list of) sweeps to display (0 = all)

==========

List (num,text1{;text2{;text3}}, {nodes})
-- Create a list of measurements with all the names 'txt'.
-- **num** -- number of the list to make
-- **text1** -- quoted text to match, you can use multiple texts by separating them with a semi-colon: **'txt1;txt2;txt3'**, the final list will be the && of all individual ones

options
-- *mets([any])* you can select nodes from the list, use simple numbering (1 to last)

-- *swps([any])* you can select nodes from the list, use simple numbering (1 to last)
NOT YET IMPLEMENTED!

==========

LeaveOnEmpty (num)
-- Test whether list num is empty. If so exit macro. (Nice in combination with do all).
-- **num** -- list to test on empty
as an alternative yoiu can define the lost as

-- list(num)
prefered for clarity!

==========

RemoveEmpty
-- This command will remove all not run measurements (broken), tested as time = 0;

==========

SnapshotOn ({num})
-- Set snapshot mode to on, so that all snapshots will function and create image files in the snap directory, made in the current datafile directory.
-- **num** -- number of the first snapshot TIFF file.

==========

SnapshotOff
-- Set snapshot mode to off, disabling snapshots commands

==========

Snapshot
-- Save snapshot (covers about window 23, with bullet-bar). Will create (if necessary the snap directory, overwrites existing files.

==========

Movie
Make an avi-Movie of the same frame as saved in snapshot to be displayed with about 1 fps and best quality (not compressed). Currently only works for display of a measurement set.
Movie will have same name as datafile (with -avi extension) and written in the datafile directory.
Parameters can not be set, but that is possible

==========

SetDAQconvert (text)
Defines how to convert DAQ files into segmented mat files. Text defines the mode:
-- **text** -- **'selected'** convert the file(s) that you selected in the selection box
-- **'all'** convert all file(s) in the selected directory
-- **'set'** convert a set of files with a number system as: XX000-XX999.

==========

SetADCrate (real)
-- Redefine the ADCrate (be careful, this can corrupt your data), but if you recorded data with the wrong sample rate (non-existing, like 30.000 Hz), than you can correct this, using this statement (mostly old data).
-- **real** -- **correct ADC sample rate to be used.**

==========

SetDACrate (real)
-- Redefine the DACrate (be careful, this can corrupt your data), but if you recorded data with the wrong sample rate (non-existing, like 300.000 Hz), than you can correct this, using this statement (mostly old data).
-- **real** -- **correct DAC sample rate to be used.**

==========

Keep (text)
-- Protocols to keep.

-- **text** -- names of protocols to keep.

==========

Delete (text)
-- Delete named protocols.
-- **text** -- names of nodes to delete.

==========

Remove (num)
-- Remove list.
-- **num** list to remove from all measurements

==========

Downscale (channel, gain, [t0 t1 {t2 t3}] {,[t0 t1 {t2 t3}], etc})
-- Rescale your data, because it was to small, or gain was incorrect.
-- **channel** channel to use
-- **gain** gainfactor to use for correction
use two values per group
-- **[t0 t1]** t0 -- t1 times of the data range to be corrected
or four values
-- **[t0 t1 {t2 t3}]** correction as above, now adjusted to the mean level of range t2 - t3
etc can be repeated as long as you wish

==========

Upscale (channel, datagain, scalegain)
-- Rescale a full channel of your data, because it was to small, or gain was incorrect.
-- **channel** channel to use factor to use for increasing the amplitude of the signal
-- **datagain** factor to use for increasing the amplitude of the signal
-- **scalegain** factor to use for adjusting the scaling of the signal

==========

Copy (target, [= A] {,mets {,swps}})
-- Copy source channel into target channel (may create a new one! careful, they can be large!)
-- **target** target channel
-- **source** target = A number of source channels
optional:
-- **mets** measurement on which to run the high pass filter (default = 1, 0 = all)
-- **swps** sweeps on which to run the high pass filter (default = 1, 0 = all)

==========

Add (target, tags, [A + B + C + ...], {,mets {,swps}})
-- Copy addition of terms into target and add tags to the name. (target = term1 + term2 +...
target target channel
tags tags to add to the name
[A + B + C + ...] numbers of channels to add to each other
optional:
-- **mets** measurement on which to run the high pass filter (default = 1, 0 = all)
-- **swps** sweeps on which to run the high pass filter (default = 1, 0 = all)

==========

Subtract (target, tags, [A - B] {,mets {,swps}})
-- Copy terma - termb into target channel adding tags (target = terma - termb)
target target channel

tags tags to add to the name
[A - B] numbers of channels to subtract from each other
optional:
-- **mets** measurement on which to run the high pass filter (default = 1, 0 = all)
-- **swps** sweeps on which to run the high pass filter (default = 1, 0 = all)

==========

Divide (target, tags, [A / B], gain, {,mets {,swps}})
-- Copy division of teller/noemer into target channel multiply by gain and add tags to the name.
target target channel
tags tags to add to the name
[A / B] numbers of channels to divide from each other
gain gain factor needed to keep the scale correct (result is same as A for display purposes
optional:
-- **mets** measurement on which to run the high pass filter (default = 1, 0 = all)
-- **swps** sweeps on which to run the high pass filter (default = 1, 0 = all)

==========

Power (target, tags,, real1, real2, gain, {,mets {,swps}})
Calculate power of channel, multiply by gain, add tags.
target target channel
tags tags to add to the name
real1 duration (s)
real2 shift(s)
gain gain factor needed to keep the scale correct (result is same as A for display purposes
optional:
-- **mets** measurement on which to run the high pass filter (default = 1, 0 = all)
-- **swps** sweeps on which to run the high pass filter (default = 1, 0 = all)

==========

Transfer ([met1 met2], [chan1 chan2])
-- Move data from meting1, channel1 to meting2, channel2; data is resempled if necessary
-- **[met1 met2]** move data from meting1 tot meting2
-- **[chan1 chan2]** move data from channel1 tot channel2

==========

ReduceChannels (num)
-- Reduce channels, so that you can calculate more!
-- **num** list of channels to remove from the data

==========

ExpandChannels (num)
-- Reduce channels, so that you can calculate more!
-- **num** number of channels to add to measurement

==========

lpass (orde, freq {,chans {,mets {,swps}}})
Low pass filter, permanently changing the data in memory! (otherwise use display modifier)

orde butterworth filter order
freq filter frequency

optional:

chans()	channels on which to run filter	(default = 1, 0 = all)
mets()	measurement on which to run the low pass filter	(default = 1, 0 = all)
swps()	sweeps on which to run the low pass filter	(default = 1, 0 = all)

==========

hpass (orde, freq {,chans {,mets {,swps}}})
High pass filter, permanently changing the data in memory! (otherwise use display modifier)

orde	butterworth filter order	
freq	filter frequency	
optional:		
chans	channels on which to run filter	(default = 1, 0 = all)
mets	measurement on which to run the high pass filter	(default = 1, 0 = all)
swps	sweeps on which to run the high pass filter	(default = 1, 0 = all)

==========

bpass (orde, [freq1 freq2] {,chans {,mets {,swps}}})
Band pass filter, permanently changing the data in memory! (otherwise use display modifier)

orde	butterworth filter order	
[freq1 freq2]	lower and higher filter frequencies	
optional:		
chans	channels on which to run filter	(default = 1, 0 = all)
mets	measurement on which to run the band pass filter	(default = 1, 0 = all)
swps	sweeps on which to run the band pass filter	(default = 1, 0 = all)

==========

bstop (orde, [freq1 freq2] {,chans {,mets {,swps}}})
Band stop filter, permanently changing the data in memory! (otherwise use display modifier)

orde	butterworth filter order	
[freq1 freq2]	lower and higher filter frequencies	
optional:		
chans	channels on which to run filter	(default = 1, 0 = all)
mets	measurement on which to run the band stop filter	(default = 1, 0 = all)
swps	sweeps on which to run the band stop filter	(default = 1, 0 = all)

==========

power (target, gain {,mets {,swps}})
Calculate power of the signal, adjust scale to match graphs

target	channel that will be squared	
gain	gain used to set adc = gain * adc * adc / 32676. For normal signals gain = 1 prevents overload.	
optional:		
mets	measurement on which to run the band stop filter	(default = 1, 0 = all)
swps	sweeps on which to run the band stop filter	(default = 1, 0 = all)

==========

cabs (target, {,mets {,swps}})
Calculate absolute value of the signal

target	channel that will be absoluted	

optional:

mets	measurement on which to run the band stop filter	(default = 1, 0 = all)
swps	sweeps on which to run the band stop filter	(default = 1, 0 = all)

==

mpass (filterlen {,chans {,mets {,swps}}})

Median pass filter, permanently changing the data in memory! (otherwise use display modifier)

filterlen length of median filter in samplepoints

optional:

chans	channels on which to run filter	(default = 1, 0 = all)
mets	measurement on which to run the median pass filter	(default = 1, 0 = all)
swps	sweeps on which to run the median pass filter	(default = 1, 0 = all)

==

Zero (list, start, end, {,chans {,swps}}})

--Zero channels by subtracting the mean value of the range from start to end (ms)

list	list of nodes to zero	
start	start time of Zero range (ms)	
end	end time of Zero range (ms)	
optional:		
chans	channels to zero	(default = 1, 0 = all)
swps	sweeps to zero	(default = 1, 0 = all)

==

Reload (num)

Without any num1 parameter: Reload the current data segment

num

==

Leak ([listset], text, [condrange], {[adc,dac]}, options)

Leak correction determines leak impedance and leak reversal voltage from a set of traces that you specify and then uses it to leak correct a (if needed different) second set

(For details of Leak method see later)

Limitations:

-- Works currently only for adc and dac that were sampled with the same sampling rate!

-- Gains in different sets can be different

[listset] may consist of list1 (then list2 is the same as list1) or [list1 list2]

list1 nodelist to use for calculating Leak

list2 nodelist of nodes you want to have corrected

text

use one of the three following commands to define how to get leak conductance:

-- **'individual'**: correct all traces individual

-- **'mean'**: calculate the mean conductance from all individuals

-- **'onemean'**: calculate conductance from mean of traces and use

-- **'trig'**: calculate the startpoint for leak correction from a trigger in the dacchannel

= =

for trig:

Leak ([listset], 'trig', level, rangebefore, rangeafter, {[adc,dac]}, {options})

level	-- trigger level to use to calculate the start point in the dacchannel, defined as follows:
[lvl1,lvl2,n]	-- lvl1 is the actual crossing level for the trigger. -- lvl2 is either -inf or inf and determines the direction of the crossing (up/down) -- n is optional (default: 1) and determines which trigger to take if more exist.
[rangebefore]	-- determine in the MATLAB way the range to take before the trigger point, this range will be negated, trigger will be considered 0.
[rangeafter]	-- determine in the MATLAB way the range to take after the trigger point, trigger will be considered 0.
{[adc dac]}	-- optional set that defines channel to use for current (adc in pA) and for voltage (dac in mV), default is [1 1] using adc channel 1 and dac channel one.

= =

for all other options one obligatory poarameter is need, that can have various forms:

Leak ([listset], txt, range, {[adc,dac]}, {options})

The parameter **range** can be defined in various ways:

[r1 r2] begin and end of the range (ms), include all points in between

or

[subrange] best done as:

[r1:dt:r2] --begin and end of the range (ms) if you want to make many ranges and be use you have all points

[subrange { subrange } { subrange } ... etc .. etc]

-- specify the selected range (ms) in a traditional MATLAB way

{[adc dac]} define channel to use for current (adc in pA) and for voltage (dac in mV), default is [1 1] using adc channel 1 and dac channel one.

= =

options:

'*show*'	show the corrected result (blue lines indicate start of range, red ones the end) use this option to control where and how you fit (uses current draw specs)
'*nologger*'	switch logger temporally off and do not drop the result data in the EXCEL logger file
'*shift*'	shift the dac trace a specified number of **data points** (prevents transients)
'*wait(0-r-inf)*'	only relevant if the show option is also there: will wait r seconds after the corrected trace is shown, for inf, you continue after pressing a continue button. If you do not add the wait option, it is taken over by the general wait in the GUI main setting. In that case use 0 if you do not want to wait!

===

Average

Calculate average std for all Measurements in memory

===

Spectra (text1,text2, text3,num,{unit},
{list, chns,range, logx/y/xy, scale, [n1, n2, n3, n4] })

Calculate and display a power spectrum of one channel (adc or dac). Spectrum is for the moment calculated

text1 'adc' will use a adc trace.

	'dac' will use a dac trace.
text2	instruction on how to display the results 'trace' will use xyd(1:traces,1). 'time' will use xyd(1,1:ranges) 'both' will use xyd(1:traces,1:ranges).
text3	name of the node that will contain the results of this operation
num	time period for single spectrum (length determines spectral resolution: 1/time). Spectra will be calculated with 50% overlap for the whole time period of the trace. A Blackmann-Harris window is used to prevent leakage)
{unit}	you can specify the units of the time definition. If not defined tghe units in the GUI will be used.

Optionals:

'window'	window in which to show the power spectrum.
'list'	list of nodes on which to operate.
'chns'	channel(s) on which to operate. If you specify 1 channel you only get power of that channel. If you specify 2 channels, you get power-1, power2, coherence and phase difference.
'range'	range on which to calculate the spectra. This command follows the complex definition: [a b] = use the range from a to b [a b], c, d use range a to b repeat c times separated by d time units see {unit}
'logx'	will draw the frequency axis on a log scale.
'logy'	will draw the amplitude axis on a log scale.
'logxy'	will draw both axes on log scales.
'scale',[n1..n4]	scale x and y axis
'scalex',[n1,n2]	scale x-axis
'scaley',[n1,n2]	scale y-axis

Spectra creates a complex multi-dimensional structure of spectra in column format.

The first 2 dimensions are the classical columns of the formula type,

first dimension = frequency axis
second dimension is timerange axis (if you use multplie times)
third dimension is (power1, power2, coherence, fase-shift) if you used 2 channels

therse is an intrinsic frequency axis (dim 1), and a trace and time axis (dim 2 & dim3)

The fourth axis of this set is:

1 = raw spectra
organisation see above
2 = mean spectra in choosen dimension
dimension 2 is mean, -sem, +sem
dimension 3 is metingen or timeranges (over which was averaged)

The first dimension can be easily accesses by the standard xy command using all five indexes.

==========

PCA (text1,numset1{, numset2}{, numset3}{, numset4})
Perform a PCA on the spectra vectors (or anything else) on the 2 dimensional column dataset.
It will add three elements to the xy data construvct

text1	nodename of the xy kolom data structure (see spectra?) used to perform the analysis.
numset1	defines (sub)set within the 2-diemnsional column dataset.
numset2	add this (sub)set to the 2-diemnsional column dataset.
etc	

3 = PCA vectors
dimension 2 is vector number
4 = PCA resultvectors
kolom 1: measurement number
kolom 2: time range number
kolom 3: mean from ztransform of original vectors
kolom 4: std from ztransform of original vectors
kolom 5: result variance
kolom 6: cumulkative variance
kolom 7: percentage excplained
5 = PCA coefficients
kolom is weight factors per vector

==========

PCAconstruct (text,num)
Reconstruct the vectors of a PCA analysis using num components and add the results to the xy data construct 9element 6 in case of a spectrum analysis

text	nodename of the xy kolom data structure (see spectra?) used to perform the analysis.
num	number of components to include in the reconstructions.

6 = reconstructed vectors
dimension 2:
dimension 3:

==========

Overdracht (win,time,'txt1',num1,'txt2',num2,{num3},{'log', 'scale', [n1, n2, n3, n4] })
Calculate and display the transfer function of channel 1 to channel 2 (adc or dac). Transfer is calculated over the full time period of the first node of a measurement.

win	**window in which to show the power spectrum.**
time	**time period (in seconds) for single spectrum (length determines spectral resolution: 1/time). Spectra will be calculated with 50% overlap for the whole time period of the trace. A Blackmann-Harris window is used to prevent leakage)**
First trace	
'txt1'	**'adc' will use a adc trace. 'dac' will use a dac trace.**

'num1'	**determines which adc/dac trace will be used.**

Second trace

'txt2'	**'adc' will use a adc trace.** **'dac' will use a dac trace.**
'num2'	**determines which adc/dac trace will be used.**

Optionals:

'num3'	**optional, will allow you to filter result with a moving average filter of number3 points (units of 1/time frequency).**
'logx'	**will draw the frequency axis on a log scale.**
'logy'	**will draw the amplitude axis on a log scale.**
'logxy'	**will draw both axes on log scales.**
'scale',[n1..n4]	**scale x and y axis**
'scalex',[n1,n2]	**scale x-axis**
'scaley',[n1,n2]	**scale y-axis**

==========

XY ([numset] {,resample,{options}})

Make a dotted X-Y graph of the points in ytrace (y-coordinate) as a function of xtrace (x-coordinate)
Logger output will always contain both

numset1	numset to define the x and the y channel in this order;
[xchan ychan]	
xchan	channel to use as x-coordinates.
ychan	channel to use as y-coordinates.

If you give the optional parameter resample, than a resampled graf will be produced (logged!) that used the resolution defined by resample (and bins the data in a cleaver way onto this axis

resample	determines the new sample with which the x-axis will be sampled (and the data averaged) in order to get a continuous new monotonous equispaced function

What you will see depends on the optional parameter show 9so it is not so optional!)

show (cmd)	define what needs to be shown: 0 = noting; 1 = raw data 2 = mean data 3 = both types.

Optionals:

nologger	block the logger for this instruction.,
window (n)	window in which to show the XY (default is 2).
mets (nlist)	measurement from which to make the XY graph (at the moment only one at a time)
swps (nlist)	sweeps to use in the XY graph (any list allowed)
range ([t1 t2])	define time range to use in the XY graph**.**

==========

Forall ('macro',[num1 ***num2*****] , text1, text2, text3)**
Will run the 'executable macro' (must be a know, that can run!) for all files in directory (will be asked to define) up to a certain depth.

macro	Macro to be executed over all datafiles selected.
num1	-- 0 is indicated directory, -- 1 is one deeper etc, -- default is inf implying running over all *.mat files in all subdirectories.
num2	optional parameter indicating how many files to take (testing purposes mainly) -- default is inf 9or take all selected files)
text1	command to add to this macro each time
text2	
text3	Filename of 3D matrix that holds formula reult (spacial for Hayriye, to guarantee quick MSATLAB access

==========

SetTextFile (text1, {num1, {num2, {text2}}})

Allows you to define how txt data files are read, converted and scaled:

text1:	Define datafile type to convert
Harmony:	Files stored as text file in the harmony system.
Celine:	Files saved in celines (Baram-lab) text file format.
num1:	Maximum segment size in Mbyte to be allowed.
num2:	Fiull scale of the adc range at which the data was clipped (symmetrical).
text2:	units that belong to the chanels scales (all scalings are assumed to be in the same units and max scaling.

==========

Names (text, {numset, txt1, num1, txt2}, {repeat}, {*device(num)*})

Rename ADC channels or DAC channels or channels in a Loaded Datafile.

text:	What are we going to rename?
ADC:	Use in protocol, to name ADC channels
DAC:	Use in protocol, to name DAC channels
adc:	Use in analysis, to rename all ADC channels in current measurements
dac:	Use in analysis, to rename all DAC channels in current measurements
numset:	Set of channels to give a new name/group/color
txt1:	New name for the channel, leave empty if you want to keep
num1:	group to which this channel belongs
txt2:	color for this group (like 'r','b','y')

You can repeat the last set of 4 parameters as often as you like

in case op naming the channels in a measurement you may add:

option:

device(num):	Define the ADC device whose channels you want to rename. (default device is 1)

==========

Firing (text, [threshold, dir, holdoff], {*options*})

Analyse the firing pattern of one or a set of depolarizations. It will produce five graphs and write the results to Logger.
1) depolarization as a function of current injection
2) spike detections indicated on a time axis, with current injection as ofset in the Y direction.
3)

text:	Define the adcchannel to analyse, as in definening the display in the window line: e.g. 'adc(2,10)/setx(200,300)' You also need to specify the range to be analysed, using the setx (gebin,eind) modifier
threshold set:	of the form [level, direction, hold off]
level	level for detecting spikes (in trace units.
direction	give inf for positive and -inf for negative direction.
hold off	minimum time between two crossings in order for the second one to be accepted.

option:

logx,logy,logxy	Use log scale for graphs 4 & 5
wait(n)	will wait between adc trace displays and also at the end of the macro.

Fitting XY data to a function

==========

Fit ({*list, source, result, fitspecs (in any order)***})**
The Fit function is the general mechanism to fit a function in particularly to any of the produced formula results (XY columns). But the same mechanism can be used to fit parts of a Y-T curve within the formula mechanism (you specify the trace as usual and then fit the desired curve to that part of the trace (see under the fittrace function in Formula)

This function only uses options (but you need to define list, source, results and fitspecs for it to work. These parameters can be given in any order.

list (n): specifies which list **n** to use in order to find the X-Y data and where the results will be dumped.

source (win, cmd): Defines the data that will be used for the fitting. It uses the first node from list and displays in window **win** using command **cmd** (e.g. xy(1,8)) the curve that will be fitted.

results (win, cmd): Defines how the result will be displayed in window **win** using command **cmd** (e.g. fit(1)). The result will automatically added to the node where the XY data resided. If you want to simply overlay the fit over the original data in the same window, you may also omit the results, is equivalent to
results (winsource, fit(n))

fitspecs (type, parameters, options):
Gives precise specification on what how to fit. type is obligatory, the other parameters are context dependent and there are again options as in the main command.

-- **type**:

Expo Exponential fit (amps, taus)
followed by one obligatory parameter:

[n1 n2 n3]	determine:
n1	the number of raising exponentials (0-1-2)
n2	the power to which to raise the rising exponentials (0-1-2=...n)
n3	the number of decaying exponentials (0-1-2-3)

you can omit 1 or to of these paramets with the following meaning:

[n1 n2]	is interpreted as **[1 n1 n2]** 1 rising exponential to power n1 and decaying exponentials
[n1]	is interpreted as **[0 0 n2]** only decaying exponentials

useful options:

offset	use an offset in the fit (see formulas below)

fitrange (x1 x2 x3)

x1	start of the range where the actual fitting will start
x2	end of the range where the actual fitting will end
x3	point in time taken as t0, will have an effect on amplitudes if you omit x3, 0 will assumed if the parameter is relevant (Expo).

In many XY situations any points outside x1-x2 will be excluded from the fit; most often the X values will be sorted to increasing order (facilitates drawing)

nolog no logger output produced during calculations (useful to suppress double logging when fitting in formula mode.

The most general formula to fit is

$$result = \left(1 - a_4 \times \exp(\frac{-t}{\tau_4}) - (1 - a_4) \times \exp(\frac{-t}{\tau_5})\right)^n \times \left(offset + a_1 \times \exp(\frac{-t}{\tau_1}) + a_2 \times \exp(\frac{-t}{\tau_2}) + a_3 \times \exp(\frac{-t}{\tau_3})\right)$$

with the following boundary conditions:
all time constants are between 0 and *inf*
a4 is between 0 and 1
offset can be any value between -*inf* and *inf*
the signs of a1, a2, a3 are the same and either between -*inf* and 0 or between 0 and inf
n must be an integer number and is not fit

if n=0 or the number of rising exponentials is 0 the formula reduces to

$$result = offset + a_1 \times \exp(\frac{-t}{\tau_1}) + a_2 \times \exp(\frac{-t}{\tau_2}) + a_3 \times \exp(\frac{-t}{\tau_3})$$

if the number of decaying exponentials is 0, the formula reduces to

$$result = offset + a_1 \times \left(1 - a_4 \times \exp(\frac{-t}{\tau_4}) - (1 - a_4) \times \exp(\frac{-t}{\tau_5})\right)^n$$

All other configurations should be self explanatory.

If you need more configurations please let me know (but realize that fitting many exponentials always gives extremely difficulot results to interpret statistically 9exponentials are not orthonormal functions.

When inserted into the formula the results appear in this order:

-- GOF goodness of fit
-- offset (if asked for)
-- amp1 decaying
-- tau1 decaying
-- amp2 decaying
-- tau2 decaying
-- amp3 decaying
-- tau3 decaying

-- tau4 decaying if you asked for 1rsing exponential
or
-- amp4 ratio between 1 and 2 raising exponential (sum must be 1)
-- tau4 rising exponential
-- tau5 rising exponential

Boltzman Boltzman fit (amps, Vh, Vc)

There are no obligatory parameters.

You fit the following Boltzmann curve to the data:

$$\text{result} = \text{offset} + \frac{\text{amp}}{1 + \exp\frac{V - Vh}{Vc}}$$

Where V is the voltage x-axis

You can fit amp in the formula, that implies that you do not have to normalize your data to 1.

If you fit offset and amp, that will only be successful if the beginning and end of the data points are almost flat. Otherwise they will most like be extrapolated to extremely large values.

useful options:

offset use an offset in the fit (see formulas below)

fitrange (x1 x2)
x1 start of the range where the actual fitting will start
x2 end of the range where the actual fitting will end
Data lying outside x1-x2 will not be included in the fit. X values are sorted.

nolog no logger output produced during calculations (useful to suppress double logging when fitting in formula mode.

Logistic Logistic fit (amps, EC50s, Hill)

There are one obligatory parameters.

nhill Hill coefficient can have three parameters
1 first order equation
2 second order equation
n Hill coefficient will be fitted with the data

You fit the following logistic curve to the data:

$$\text{result} = \frac{\text{amp}}{1 + \left(\frac{[\text{conc}]}{EC_{50}}\right)^{\text{hill}}}$$

Where [conc] is the concentration x axis.

You can fit amp in the formula, that implies that you do not have to normalize your data to 1.

If you fit offset and amp, that will only be successful if the beginning and end of the data points are almost flat. Otherwise they will most like be extrapolated to extremely large values.

useful options:

fitrange (x1 x2)
x1 start of the range where the actual fitting will start
x2 end of the range where the actual fitting will end
Data lying outside x1-x2 will not be included in the fit. X values are sorted.

nolog no logger output produced during calculations (useful to suppress double logging when fitting in formula mode.

Poly1 Polynomial fit (1e order) of the form a1 + a2*x
Poly2 Polynomial fit (2e order) of the form a1 + a2*x + a3*x^2)
Poly3 Polynomial fit (3e order) of the form a1 + a2*x + a3*x^2 + a4*x^3)

Straight forward linear fits (no iterations needed).

useful options:
fitrange (x1 x2)
x1 start of the range where the actual fitting will start
x2 end of the range where the actual fitting will end
Data lying outside x1-x2 will not be included in the fit. X values are sorted.

nolog no logger output produced during calculations (useful to suppress double logging when fitting in formula mode.

---

fitvalues = fittrace ('fittype',parameters, {list of arguments})
This command will fit part of a current trace to the functions asked for. Most of these functions will be auto organizing, so you do not have to provide starting functions or detailed properties of the function. Many of the commands are identical to the FIT function that fits a function to any XY data in a node, but notice the benign details.

fittype **'Expo'** fits one or more exponentials
etc etc same options as under Fit, some make more sense than others.
The input parameters depend on the type of function; also the output is determined by the precise fit.

The output will create columns in the function output array. At the moment you have to indicate the names yourself as I have to work on an elegant way to do that automatic. Use the following rules:

==========

The Formula mechanism

The Formula to evaluate should be a correct MATLAB statement as it is evaluated by MATLAB, but you should use as many predefined functions as possible:

You can do any legal calculation, define and use parameters of the **p.anything** type.

The formula system is fully open ended so any function can be used or even added as an m file (if you work under the interpreter and not under the compiler. Function should use the data arrays in the global Formula that contains all information on the last data displayed (inclusive modifiers). This has not been tested extensively, so shout if you want to use it.

==========

SetSlope (real1)

Set the threshold value in the Slope1/2 function.

real1 Relative threshold for slope calculation (default = 0.2).

==========

Formula_Set (string1, string2, {*string3*,{*string4*}}, {window(n)})

-- **string1**: text that specifies which channel how to use. This final format will form the input for the formula (so filtering menas that the formula input is filtered. (slash separated options)

-- **string2**: text that defines the formula to use for evaluation. You can define as many formulas as you wish.

-- ***optional string3***: *defines the meaning of the output. It is a string in which items are separated by ";". Each text item defines the meaning of an answer column. These names will be used as legends in the EXCEL sheets that you produce. You need to know how many answers a specific formula is going to produce in order to fill in the right comments. Most formulas produce a single column answer, but there are a few that produce more. They even can be variable, if that is the case the final width of the answer in node or excel files is determined by the longest answer in the run. Additional values are filled in by NaN's*

-- ***optional string4***: *defines the name of this formula answer for use in subsequent formulas. This name needs to start with captal F for security reasons (it can interfere with my internal MATLAB code) e.g: "F". If your answer has more than one variable you can use indexing in the form of Fresult(nn) where nn is the index in the variable list. You need to know the variables from the documentation*

option:

-- ***window(n):*** window to use for the plot

==========

Formula_Run (list, formulas, setname,nodename, {*display instruction,..* })

-- **list:** which previously defined list of measurements/sweeps to use for evaluation.

-- **formulas:** List all formulas to be evaluated, all will be defined by **'all'** (use standard MATLAB format[1 2 3 4 etc]

-- **setname:** Name of node (made if necessary) in which the result will be stored or if it exists, to which it will be added as a new node.

-- **nodename:** Name of node (made if necessary) in which the result will be stored or if it exists, to which it will be added as a new node.

-- *source(window,'instruction')*
pairs of numbers and texts that will be used as default plot options (combination of windownr and plot text, such as 9, 'xy(1,2)'; you can continue for as many necessary). See all modifiers for details

option:
-- *result(window,'instruction')*
pairs of numbers and texts that will be used as default plot options (combination of windownr and plot text, such as 9, 'xy(1,2)'; you can continue for as many necessary). See all modifiers for details

Most functions produce 1 output value, but several can do more, even variable number of outputs. Some need no special input:

-- **time()** moment in time the trace started (trigger time), this is the same as time axis in the time bar, start of measurement is 0 (if needed different time definitions =can be defined (unit = seconds). Often easy to use as time axis!

-- **Time()** absolute moment in time since the birth of jesus. Better if your data spans severeal dats. (if needed different time definitions = can be defined (unit = seconds). Often easy to use as time axis!

-- **Anyname** You can define a name for the ouput of a formula and use this variable in subsequent formulas. If more than 1 output variable are produced by a formula (e.g. fitting) you may index them. Look in this manual or in the EXCEL file to see the meaning of the variables.

{extra}
In the folowing "extra" denotes options that define either color or symbol given by the added letters.
-- for color you can choose from: 'rbmkcgy'
-- for symbol you can choose from: 'odphx+*s^v><'
as these symbols are non-overlaping you may give each of them in the same string

anytime = lin (time *{extra}*)) draw vertical line at time, (optional color, symbol). This is an easy way to add information about ranges in your formula.

---

value = pnt (t1 *{extra}*) value of point t1 (time) (optional color, symbol)
value = pntbig (t1 *{extra}*)) value of point t1 (time) (optional color, symbol), use big symbol
value = avg (t1, t2 *{extra}*)) average of range t1 – t2 (optional color, symbol)

---

tmin = minp (t1,t2 *{extra}*)) moment of minimum in range t1–t2 (optional color, symbol)
value = minr (t1, t2 *{extra}*)) minimum in range t1–t2 (optional color, symbol)
value = minmean (t1,t2,t3*{extra}*)) average from -t3 to +t3 around the minimum found in range t1–t2 (optional color, symbol)

---

tmax = maxp (t1,t2 *{extra}*)) moment of maximum in range t1–t2 (optional color, symbol)
value = maxr (t1, t2*{,extra}*)) maximum in range t1–t2 (optional color, symbol)
value = maxmean (t1,t2,t3*{extra}*)) average from -t3 to +t3 around the maximum found in range t1–t2 (optional color, symbol)

---

tpnt = extp (t1, t2*{,extra}*) t-extreme: largest of abs(min) or abs(max) range t1–t2 (optional color, symbol)

value = extr (t1, t2*{,extra}*) extreme: largest of abs(min) or abs(max) range t1–t2, (optional color, symbol)

value = extmean (t1,t2,t3*{extra}*) average from -t3 to +t3 around the extremum found in range t1–t2 (optional color, symbol)

---

value = slope1 (t1, t2) determine (LS) slope in range t1 – t2.

-- **t1**: where to start the slope

-- **t2**: where to end the slope

procedure: find the slope of the :LSQ fit to the t1-- t2 range.

---

value = slope2 (t1, t2, 'dir') determine slope in a better way.

-- **t1**: where to start looking for the slope?

-- **t2**: how far maximally to look for the slope?

-- **dir**: 'up' pos slope, 'down' neg slope.

procedure: find first maximum (that lasts for at least 10 points or timepoint t2 whichever comes first,
than we go 10% up and down (level set with Setpars, parameter1)
and slope is calculated. seems very robust,
characteristic points are drawn

---

value = condition (test,c1,c2,c3)

-- Conditional statement. All four parameters have to end in a single niumber (as usual).

test	Conditional stetement. The value of this one detrmines value.
c1	Answer if test < 0
c2	Answer if test = 0
c3	Answer if test > 0

---

value = spike (t1,t2,t3) Determine spike amplitude using left and right border (t1, t3) and estimate of spike location (t2). Find two local maxima and then minimum in between, construct and show spike amplitude
A better (raaklijn!!) definition will be constructed.

[value, std] = mintrain (*{color}*)

In the sync mode will determine the minima within each sin-wave cycle, and then determine the mean value and the standard deviation. Symbols and line will be drawn in Color.

[value, std] = maxtrain (*{color}*)

In the sync mode will determine the maxima within each sin-wave cycle, and then determine the mean value and the standard deviation. Symbols and line will be drawn in Color.

[values] = getfreq (number1, number2)

Function will decompose the dac signal into two frequency components, or into a sine wave and an alfa function.

number1 If this number is >0 it determines where the spectrum will be split in order to determine 2 peak (maximum in each region)

number2 f this number is > 0 it determines how large the gap is to use in the sinewave fitting in order to separate the alfa function.

So for number1 = 0 and number2 = 0 only one sinewave is evaluated

-- offset meanvalue of the sine wave
-- peak frequency of the wave (based on maximum in spectrum.
-- cos amplitude of the cosine component in the wave (same units as original dac signal)
-- sin amplitude of the sine component in the wave (same units as original dac signal)

If number1 >0 and number2 = 0 then the dac signal will be decomposed into two sine wave signals

-- offset meanvalue of the sine wave
-- peak1 frequency of the first wave (based on maximum in left part of the spectrum.
-- cos1 amplitude of the cosine component of peak1 (same units as original dac signal)
-- sin1 amplitude of the sine component of peak1 (same units as original dac signal)
-- peak2 frequency of the second wave (based on maximum in right part of the spectrum.
-- cos2 amplitude of the cosine component of peak2 (same units as original dac signal)
-- sin2 amplitude of the sine component of peak2 (same units as original dac signal)

If number1 =0 and number2 >0 then the dac signal is decomposed into a sine and alfa function. The value of number2 is the time span that will cover the width of the alfa function.

-- offset meanvalue of the sine wave
-- peak frequency of the wave (based on maximum in left part of the spectrum.
-- cos amplitude of the cosine component of peak (same units as original dac signal)
-- sin amplitude of the sine component of peak (same units as original dac signal)
-- phase mean phase shift of alfa function in respect to start of the sine wave component.

---

value = DACpnt (t1)

-- **t1** Time in ms. Find the value of dac trace at point t1

---

timepoint(s) = trigger ('direction',level, {list of arguments})

Trigger can detect triggers (crossings) in the last displayed signal and also perform extra measurement in special cases depending on the added list of optional arguments.

'direction' **'up'** upward crossing of a level
'down' downward crossing of a level
'all' both crossings of a level in the relevant signal

level the level relevant for trigger determination (in trace units).

Rest of arguments are optional and can be given in any order:

number which crossing to take (1 to n), if number is negative than we count backwards (-1 is the last one).

'dac' determine the trigger in the additional metdac signal, so that these points can be used in the adc signal

'empty' You can define which value to return if there is no trigger found.
[t1] value to return when no trigger is found

'range'	optional parameter, next parameter defines range in trace on which to work
[t1 t2]	start & end within this trace for this function.
'sync'	functions that determine parameters in relation to a sinewave in one of the dac channels. Use metdac(n) in the display! Zero phase moments are indicated by a vertical red line
'trim'	make sure that last trigger point in the trace is at least 1/16 away from end (to prevent that it sometimes does not show!)
'skip'	optional parameter to set the maximum width of a burst. Next parameters:
t3	t3 defines the time within which a nect (spike) crossing needs to come in o9rder to define a burst.
'#waves'	triggerfunction will determine the number of full waves in the trace: procedure is: positive nul level crossings signal minus its mean value.
'#spikes'	triggerfunction will determine the number of crossings in the trace, for a skip value of 0 this is identical to #waves. Spikes are indicated with big red circle
'#bursts'	triggerfunction will determine the number of bursts in the trace: A burst is defined as an extra crossing of the trace within the skip time. Bursts are indicated with big red circle
'meanphase'	Calculate the mean phase difference between alfa pulse and sinewave signal
'phases'	Calculates the phase difference of each burst & spike in respect to the fullwave determined in the metdac signal. A value will be given for each waveperiod in order to keep the output signal at a constant length (multidimensional result and excel file). The value of phase is between 0 -- 360. For ease of information spikes are positive, bursts are negative coded zo abs(phase) is the real value.
'daps'	calculate size of depolarizing after potential. Next parameters mean:
[t1 t2]	t1 is the window after the spike to find the minimum (hyperpolarization). t2 is the window after the minimum to find the maximum of the dap. Procedure to determine DAP first works on the spikes. Find minimum after the spike within first time window(spk - t1). From there find maximum within the next window (spk - t2). Difference between these two values is the DAP. This is difficult to do in a burst. There I simply use the value at the mean moment in time where the DAP in the single spike was found.
'dapmin'	calculate size of depolarizing after potential. Give back the minimum value after the spike (see daps for details of the procedure).
[t1 t2]	t1 is the window after the spike to find the minimum (hyperpolarization). t2 is the window after the minimum to find the maximum of the dap.
'dapmax'	calculate size of depolarizing after potential. Give back the maximum value after the minimum after the spike (see daps for details of the procedure).
[t1 t2]	t1 is the window after the spike to find the minimum (hyperpolarization). t2 is the window after the minimum to find the maximum of the dap.

---

result = Twolines ([t1 t2],{xyweight})

This command will fit part of a current trace between t1 and t2 with two straight line. It will determine by itself where to put the breakpoint in order to minimize the total fiterror (in the y direction!). You get as an answer the x & Y cordinates of the crossing point of the first end the second line. If you also supplu xyweight you will get the point on the curve that is located the closest to the ncrossing point. As x and y coordinates are generally in a different coordinate systen (e.g x=time axis, y = volt axis) you need to give a number that tells the calculatiuon how to weight x in respect to y. numbers can vary between 0 (x = irrelevant) to inf (y=irrelevant).

This is wa routine useful for finding the AP threshold as an example. It is used within spikesfind and will there be displayed.

t1 startmoment of the range in the curve to fit
t2 endmoment of the range in the curve to fit

optional
weight how to weight the x-coordinate in respect to the y-coordinate when determining the distance to the curve

result depends on your command:
minumum:
--**xcrosspoint** (in time units)
--**ycrosspoint** (in amplitude units)
--**slope-line1** (in amplitude per time units)
--**slope-line2** (in amplitude per time units)
if you provide weight:
--**x-closest point on curve** (in time units).
--**y-closest point on curve** (in amplitude units)
---

result = spikesfind (r1, r2, r3, {r4, {r5}})
Command designed to analyse adaptation pattern of spiketrain evoked with depolarizations under current clamp.
Many commands are combined in this one, use whatever you need.

When no spikes are detected the response will be fitted with a single decaying exponential function

r1 **[t1 t2]** start and end of the range in which spiking is analysed (usual the depolarization)
t1 start the range in which spiking is analysed (usual the depolarization)
t2 end of the range in which spiking is analysed (usual the depolarization)

r2 **spike detect parameters**
[p1 p2 p3 p4 p5 {p6}]
p1 threshold for detection of the spike in the convoluted spike train signal.
p2 time to take before the crossing detection in which to find the spike peak (the length of p3 will be used after detection)
p3 halfwidth of the continuous average of the signal that will be subtracted from the trace before thesholding. Use a value that covers at least the half width of the broadest spike you want to detect.

p4 minimum amplitude of a spike in order to be kept (after all other criteria are met. Amplitude is defined as the difference between maximum value and minimum between two successive spikes (for the first spike: difference between spike maximum and the minimum 0.5 ms after the depolarization.

p5 absolute amplitude above which the spike must reach in order for the maximum to be counted as a spike peak.

As double detections will be removed, extra croissings within the same detection range are no problem. Also peak values detected at a border are not used.

p6 optional parameter, if you give it a value the detection signal and threshold will also be shown, Use for fine tuning, it may mess up your image

r3 **fit parameters that define the fitting proces**
[f1 f2 f3]

f1 timelength to be used for the exponential fit when there are no spikes.

f2 maximum time allowed before the spike when detrmining the two lines for threshold detection (strats at minimum or this point whichever is closest).

f3 relative weight between x coordinate and y coordinate when determining the curve point for spike thershold (see description of Twolines in the formula)

r4 **options parameter for plotting**
[w1]

w1 window number in which to show the adaptation fit (default is 1).

r5 **options parameter used to wait during display**
[d1{, d2 {, d3}]

d1 delay in (fraction of) seconds to wait after displaying a single spike (default 0).

d2 delay in (fraction of) seconds to wait after displaying of RC fit (default 0).

d3 delay in (fraction of) seconds to wait after displaying spiketrain (default 0).

Reults are Logged in the EXCEL/TXT sheet as usual:

-- first block is the summarized output of the formula

-- then follows one blok of analysis for each spike train!
In fact there are 3 forms: one if there are no spikes, describing the tau. A different one for 1 spike and for 2, 3, 4(using instantaeous freq, upto 4 parameter fit.
---

fitvalues = fittrace ('fittype',par, {list of arguments})
This command will fit part of a current trace to the functions asked for. Most of these functions will be auto organizing, so you do not have to provide starting functions or detailed properties of the function. Many of the commands are identical to the FIT function that fits a function to any XY data in a node, but notice the benign details.

fittype	**'Expo'**	fits one or more exponentials
	'Expot'	fits spike train adaptation formula
	par	the number of exponentials to fit
	'Boltzman'	fit a standard Boltzxman function
'nolog'	giving this command will suppress the logging of the individual fit (parameters end up in the EXCEL dataset anyway).	
'offset'	if given the fit will use a constant term.	

Examples of typical fitfunctions:
fittype can be:

Expo	Exponential (kinetic) fit
Expot	Combined Linear-Exponential (kinetic) fit
Boltzman	Boltzman fit
Logistic	Logistic fit
Poly1	Polynomial fit (1e order p1*x + p2)
Poly2	Polynomial fit (2e order p1*x^2 + p2*x + p3)
Poly3	Polynomial fit (3e order p1 * x^3 + p2*x^2 + p3*x + p4)

The input parameters depend on the type of function; also the output is determined by the precise fit.

fittrace ('Expo',par1,par2, 'offset','range',[t1 t2 t3],'nolog')
-- fits a function of the type

y = amp0 + amp1 * (1-exp(t/tau1))^npwr *
(amp0b + amp2 * exp(t/tau2) + amp3 * exp(t/tau3) + amp4 * exp(t/tau4))

where

par1	determines the value of npwr, for par1 =0, we only fit a decaying exponential, for larger (whole) values we make an activation time constant of 1 or higher order. The value of par1 also determines where the potential offset will be located: for par = 0 it will be amp0a, for higher values it will be at amp0b.
par2	determines the number of decaying exponentials (each with anplitude and time constant (max 3, easy expandable, but higher orders will not fit well)
'offset'	if this option is added the value of either amp0a or amp0b will be fitted.
'nolog'	if this option is added logger output of the direct fit will be suppressed.
'range'	this option is needed in order to define the fitrange. In principle we need 3 timepoints (units of the x axis). They have the following meaning:
[t1 t2 t3]	
t1	start of fit region
t2	end of the fit region
t3	point to take as t0, for evaluating the amplitude.

The output will create columns in the function output array. At the moment you have to indicate the names yourself as I have to work on an elegant way to do that automatic. Use the following rules:

output = [GOF, pars] with

GOF	goodness of fit (kind of correlation coefficient).
par1	is offset if present (otherwise non-existent)
for decaying exponentials only:	
par2	amplitude of component
par3	tau of component
etc for each following component	

if activation exponentials are also present the format will be

GOF	goodness of fit (kind of correlation coefficient).
par1	is offset if present (otherwise non-existent)
par2	main amplitude factor
par3	tau of rising component

return values for Expo:
-- **goff** Goodness of fit (LSQ, 1 = perfect)
-- **amp1** amplitude of first exponential function extrapolated to x0 if given.
-- **tau1** time constant of first component in ms.
if present:
-- **amp2** amplitude of second exponential function extrapolated to x0 if given.
-- **tau2** time constant of second component in ms.
if present:
-- **amp3** amplitude of third exponential function extrapolated to x0 if given.
-- **tau3** time constant of third component in ms.

Artefact suppression routines

==========

AutoMacro (number)
Set a value in the AutpoMacro knob.
If different from 0, that macro (you have to count), will be execurted whenever you (re)load a file or a asegment of a file. In paticular useful for filtering, zeroing, artefact removal. realize that on loading you can perform these action on the date, not only on the display 9saves a lot of time!

==========

Artefact (type, channel, [before after] {,list,met,chans,swps})
Suppress artefacts in adc traces based on the artefact channel generated in the protocol.
The procedure is as follows: assume no artefact and only nice digital pulses, the routine finds a threshold level, halfway between minmum and maximum. Then find upgoing thresholds and next downgoing thresholds, and then apply the artfeact suppression for all triggerpoints depending on the type (text) defined as first parameter.
The artefact will be corrected in all channels exept the one that contains the srtefact puls, unsless ytou define a selection yourself.

type	type of the artefact suppression -- **first**: use first point of the selection to set artefact range -- **nul**: set artefact range to zero -- **interpol**: interpolate from first to last point of artefact range
channel	adc channel to use for artefact detection (contains digital pulses)
[before after]	times (start & end) of in respect to positive trigger and to negative trigger. also, for [0 0] the triger points will be used.one or more timepoints where stimulation artefacts will be suppressed in datapoints.
{ channelset}	You can overrule the channels to correct by defining a selection yourself, if not defined all channels except the artefact channel will be used.

optional:

list (numset)	define list to use.	
mets (numset)	mets	(selection, default = 1, 0 = all)
chans (numset)	channels	(selection, default = 1, 0 = all)
swps (numset)	sweeps	

==========

Artifact_List (num1, [list1], [list2])
Remove artifact in a channel of Measurement **1** by simply setting level to 0 (more methods pending). The two lists need to be of equal length.

num1,	-- channel(s) in which to suppress the artifacts.
[list1]	-- list of moments (in seconds) where to suppress artifacts
[list2]	-- list of durations (in seconds) of how long to suppress artifacts

Example: *Artifact (2, 60*[17.6 33.2], [4 20])*
Suppresses two artifacts: one after 17.6 minutes for 4 seconds and one at 33.2 minutes for 20 seconds.

==========

Spike_Artifact (num1)

Do automatic artifact reduction and removal in the spike train situation. This holds in particularly for the fast oscillation type events that produce extreme high firing rates.

num1, Threshold for artifact reduction in the spike train.

For artifact reduction proceed as follows:
Detect spikes in the train as you would do if and calculate the spike rate in the channel that you wantto use. Make sure that you set the minimum interval parameter in the Detection command to zero so that you get the high rates. Also chose a quite low binning width in the rate command and make sure you use the '*points*' option. Look careful in the Spike_Rate grafic and decide where to put the high threshold for artifact detection. The Artifact command will now detect the artifacts and the duration of the artifacts and sets the ADC channel to zero (other options pending)

==========

Artefact_Trigger (num1, text, [parameters])
Suppress artefacts using the "trigger" generated by Event detect (and eventually the matching reversed triggers detected.

num1, Channel in which to suppress the artifact
text, Command that defines mode and also the meaning of the rest of the parameters

Following commands can be given for text:
'zero', Replace artefact by zero
[t1 t2], t1 ms before time and t2 ms after the artifact detection.

'interpol', Interpolate artefact between start and end points
[t1 t2], start is t1 ms before time and end is t2 ms after the artifact detection.

gent, Special for Gent configuration where one channel records artefact.

'mpulsen' then parameters are: time before (ms) skip if interval larger than (s)
Interpolates artefact from before to before next puls (useful for large depolarizations)

'submean' then parameters are: time before (ms) time after (ms)
Subtracts mean artefact from all other artefacts.

'submeaninterpol' then parameters are: time before (ms) time after (ms) freq(Hz)
Subtracts mean artefact from all other artefacts using sub sample interpolation for a frequency of freq Hz.

Spike/Event analysis routines

==========

Resample([numset1] {,average} {,options})

This routine can be used to resample a very high sampled trace into a more useful datatrace for further analysis purposes. Assume you recorded minis at a high sample rate (10.000 pnts/s) and now want the underlying voltage or current trace as a function of time. But 1/s is sufficient resolution. Instead of exporting 20.000.000 points you can get a trace with sufficient points to handle it further in EXCEL.

Use Event_Export to output the data, as the actual points are stored in the spike register, which allows easy over plotting and inspecting.

[numset1],	resampling definition: various formats hold: -- **num1**: sampling interval in seconds, use whole trace. -- **[begin, end]**: sampling interval in seconds plus start and endpoint (you may define inf for the whole trace). -- **[begin, interval, end]**: sampling interval in seconds plus start and endpoint (you may define inf for the whole trace).
average,	timewidth over which to average the resampling value. If you do not define single points will be taken. If you define a range larger than half the filtering interval, you start filtereing. Look at the result.

Use Spike on in the event window to set the display of the result on/off. various formats hold:

==========

Spike_Detect (num1,[numset1],[numset2],{num2}{,options})
Event_Detect (num1, [numset1], [numset2], {,options})
Puls_Detect (num1, [numset1], [numset2], {,options})
DAC_Detect (num1, [numset1], [numset2], {,options})

==========

Detect (cmd, numset1, {numset2}{,options})
This routene combines all the options previously included in Spike_Detect, Event_Detect en Puls_Detect. Irt also includes different modes of detection for specfic events like spikes, minis, digital pulses etc. Try to replace the earlier versions in your macros by this one, as this is the one that is maintained, and can be used with MEA etc.

cmd,	Selects the detection algorithm, many parameters are context dependent:	
	-- **spikepp**:	use the spike peak-peak detection algorithm (search for at least this difference in max-min within a certain range). **Detect (spikepp, numset1, [searchrange(ms)],** *options***)**
	-- **spikedrm**:	use spike threshold detection algorithm (search for an absolute threshold criterion in some direction) **Detect (spikedrm, numset1, [min-maxwidth(ms)],** *options***)**
	-- **event**:	use event the threshold threshold detection algorithm (see above), but therafter readjust the threshold relative to the peak of the detected event. Assume the have the same shape and you want to align and normailze them later)as in mini's) **Detect (event, numset1, {numset2},** *options***)**

-- **dactrg**: use simple threshold trigger detection in a DAC channel
Detect (dacdrmp, numset1, {numset2}, *options***)**
-- **puls**: use puls threshold detection algorithm
Detect (puls, numset1, {numset2}, *options***)**

Options(below) defines: measurement, channel, sweeps. Default is (chns(1), mets(1),swps(1))

The meaning of numset1 is context sensitive, as the "threshold criteria are different in tghe various algorithms.

numset1	Set of numbers to use for thresholding in amplitude units (context dependent): **auto / num*sd /pos(n1) inf / neg(n1) -inf / [n1,n2]**
*'auto',**n3***	Will use the thresholds automatically detected by Spike_Scan in MEA.
*'sd',**n3***	You may use the word sd in a formula (like [2*sd 3*sd] in order to use the standard deviation of the trace underconsideration determined over the full trace.
*[n1 n2,**n3**]*	range within which to detect events. [5 10] will detect events with a peak between positive level 5 and 10. [5 inf]) will detect any 5 level crossing in positive direction. [5 -inf] will do the same crossing in negative direction.
[n1]	Use single level automaticaaly extended in one direction: a positive n1 is interpreted as [n1 inf], a negative n1 is interpreted as [n1 -inf].
n3	is used only in event to indicate that the threshold needs to be rescaled to a percentage of the maximum value of the detected event (n3 is a fraction betwwn 0 and 1)
==========	==========
[n1 n2 n3 n4.....]	to be implemented! Detect multiple ranges (non-overlapping, continuous. events will automatically be classified as class 1- class 2class 3 etc until ninf.
==========	==========

numset2	Conditions used in event dependent (optional and context dependent):
In **spikepp**	*obligatory*
n1	n1 = width of the searchrange in ms. Be carefull!; the algorithm eats up lots of memory, so do not use too wide a range on very long traces! You neer tracelen * searchwidth in points! In any case limited between 3 and 999 points
In **spikedrm**	*optional*, default is: [0 inf]
[n1 n2]	n1 = minumum and n2 = maximum width of the event in ms (falling, rising for positive, inverse for negativer) in order to be accepted
In **event**	*obligatory*, no default
[n1 n2]	n1 is the searchrange (ms) in which you have to find a real peak (not the last point in a rising line) n2 is minimum interval (ms) between events, for the second one to be accepted (if smaller than this one, it will not be counted
In **dactrg**	*none*
n1	n1 = hold-of-time in ms, how long do you want to suppress a trigger after you found one?

options: Valid options are *fullframe, frame, range*:

'mets',	select your measurement (default: 1)
'chns',	select your channel (default: 1)
'swps',	select your sweep (default: 1)
'range(x1,x2)'	set the datarange to the hard zoomrange to x1 -- x2. You can also do this by using the Setframe command and then use "frame"
'fullframe',	set the datarange to the full time zoom window
'frame',	set the datarange to actual timeframe from the zoomwindow.
'bursts(t1)'	switches on burst detection (will use and override the classes!) and will classify each event as one of the following: isolated spike (class 1), first spike in a burst (class 2), or secondary spike in a burst (class 3). Classes are automatically assigned and displayd when you switch on the spike display. This option excludes multiple spike analysis.
'save',	save the event results in the special xy array than makes them accesible for formula, mask etc

==========

Spike_Rate (num1, text, num2 {,options})
Calculates the rate as a function of time from a event train.

Which events are counted:

There is a hierarchy in this process. If you define a "mask" with the mask option, it will overrule everything and only the events in the mask are taken. If you do not define the mask than the frame/range settings will be used: In "fullframe" the whole frame as defined in de zoombar will be taken (so it can be a segment), in "frame" the frame you selected in the zoombar will be taken, while the "range" option allows you to specify the range in a hard manner with range numbers directly following the range command: [begin end]. Use seconds or add the units as an additional third parameter ('ms' 'sec' 'min' 'hrs') to the range command.

num1,	
text,	-- select calculation mode and graph type: *'points':* produces a blue line for event rate and a red one (left axis) for event amplitude. Each point is calculated from **num2** event. *'ratebars':* will produce a blue bar graph of the mean event rate based on the number of events within the binwidth as defined by number2 *'ampbars':* will produce a red bar graph of the meanamplitude of all events within the binwidth as defined by number2
num2,	-- parameter that determines resolution (depending on mode parameter). *points:* Indicates how many events around this event will be used to calculate the spike rate (You get as many event rate points as you have events, resolution is high at high frequency, low at lower frequency, error is always the same order (SQRT(N))

ratebars:
binwidth (in s) used to calculate event rate (all data points used)
ampbars:
binwidth (in s) used to detect events, mean amp of detected events is used.

Options: *mask, scale, xscale, yscale, fullframe, frame, range*: (see general remarks)
'nologger' switch logger off in this statement

'fullframe' set the datarange to the full time zoom window
'frame' set the datarange to actual timeframe from the zoomwindow.
'range' [x1 x2] set the datarange to the hard zoomrange to x1 -- x2. You can also do this by using the Setframe command and then use "frame".

'mask' define the mask to use for this operation. Mask overrules range, frame, fullframe. (see detailed explanation of the masks)
-1:
no mask, use all datapoints; identical to not defining mask at all
empty value, i.e. []:
get the mask as interactively defined in the GUI.
0:
a special mask that includes only the included datapoints, exclude mask is empty.
any number >=1:
use this mask number, you are responsible that it exists (get a reasonable error message).

'scale' *[x1 x2 [y1 y2]]* standard option for graph scaling
'xscale' *[x1 x2]* standard option for graph x - scaling
'yscale' *[y1 y2]* standard option for graph y - scaling

==========

Spike_Amplitude (real {,options})
Analyze the spike amplitude distribution and fit it with various functions.

The histogram can be fitted with a (complete or incomplete) Gaussian function. The incomplete Gaussian will be fitted between the amplitude thresholds (lower and upper) that you used in the spike detection proces. Subranges can not be fitted (scream if you need it, it can be added).

Which events are used:
There is a hierarchy in this process. If you define a "mask" with the mask option, it will overrule everything and only the events in the mask are taken. If you do not define the mask than the frame/range settings will be used: In "fullframe" the whole frame as defined in de zoombar will be taken (so it can be a segment), in "frame" the frame you selected in the zoombar will be taken, while the "range" option allows you to specify the range in a hard manner with range numbers directly following the range command: [begin end]. Use seconds or add the units as an additional third parameter ('ms' 'sec' 'min' 'hrs') to the range command.

num, Window in which to draw the result
real, Binwidth in the actual amplitude units.

options: *mask, scale, xscale, yscale, fullframe, frame, range*: (see general remarks)
'window'(n) window to display the result

'nologger'	switch logger off in this statement
'fullframe'	set the datarange to the full time zoom window
'frame'	set the datarange to actual timeframe from the zoomwindow.
'range' [x1 x2]	set the datarange to the hard zoomrange to x1 -- x2. You can also do this by using the Setframe command and then use "frame".
'mask'	define the mask to use for this operation. Mask overrules range, frame, fullframe. (see detailed explanation of the masks) *-1:* no mask, use all datapoints; identical to not defining mask at all *empty value, i.e. []:* get the mask as interactively defined in the GUI. *0:* a special mask that includes only the included datapoints, exclude mask is empty. *any number >=1:* use this mask number, you are responsible that it exists (get a reasonable error message).

'scale'	*[x1 x2 [y1 y2]]*	standard option for graph scaling
'xscale'	*[x1 x2]*	standard option for graph x - scaling
'yscale'	*[y1 y2]*	standard option for graph y - scaling

==========

Spike_Interval (real1, {realset}, {,options})

Analyze the spike interval distribution and fit it with various functions e.g. exp distribution (red) / gamma distribution (pink):

The distribution is fitted with an exponential distribution and a gamma distribution. An incomplete gamma will also be fitted for a parameter range that you can define (leave out the lowest values, which are most likely not correctly detected.

Which events are used:
There is a hierarchy in this process. If you define a "mask" with the mask option, it will overrule everything and only the events in the mask are taken. If you do not define the mask than the frame/range settings will be used: In "fullframe" the whole frame as defined in de zoombar will be taken (so it can be a segment), in "frame" the frame you selected in the zoombar will be taken, while the "range" option allows you to specify the range in a hard manner with range numbers directly following the range command: [begin end]. Use seconds or add the units as an additional third parameter ('ms' 'sec' 'min' 'hrs') to the range command.
Once your events are selected, the intervals for the distribution are calculated and the whole proces is executed.

real1,	Binwidth in actual amplitude units (mainly for drawing the histogram, does not affect calculations.
realset,	two numbers to be used in the fitting of the exponential fit and the gamma fit. The first one is the startpoint of the expo nential fit (for gamma always 0), the second one is the endpoint for both fittings (can be critical for the gamma fit!
options:	*mask, scale, xscale, yscale, fullframe, frame, range*: (see general remarks)

'window'	window in which to plot the results
'nologger'	switch logger off in this statement
'fullframe'	set the datarange to the full time zoom window
'frame'	set the datarange to actual timeframe from the zoomwindow.
'range' [x1 x2]	set the datarange to the hard zoomrange to x1 -- x2. You can also do this by using the Setframe command and then use "frame".
'mask'	define the mask to use for this operation. Mask overrules range, frame, fullframe. (see detailed explanation of the masks) *-1:* no mask, use all datapoints; identical to not defining mask at all *empty value, i.e. []:* get the mask as interactively defined in the GUI. *0:* a special mask that includes only the included datapoints, exclude mask is empty. *any number >=1:* use this mask number, you are responsible that it exists (get a reasonable error message).

'scale'	*[x1 x2 [y1 y2]]*	standard option for graph scaling
'xscale'	*[x1 x2]*	standard option for graph x - scaling
'yscale'	*[y1 y2]*	standard option for graph y - scaling

==========

Spike_PSTH (num1, num2, num3, {text}{, options})
Calculate PSTH

num1,	-- time before trigger in seconds
num2,	-- time after trigger in seconds
num3,	-- binwidth in seconds
optional	
text,	-- units to use, either counts{default} or Hz

options:

'window'	window for result

==========

Amplitude (num1, num2, num3, num4, real1, num5 {,real2} {, options})
Analyzes the amplitude distribution within a trace (for threshold decision etc).

num1,	-- measurement to use
num2,	-- sweep to use
num3,	-- channel to use
num4,	-- window in which the result will be displayed
real1,	-- binwidth to use for the histogram (in x-axis units)
num5,	-- reduce factor (fractional samples to take (do not use more than 1.000.000 points, the fits will take a long time!)

Optional number:

{real2},	number that will indicate how many times the interquartile range will be used around the median, in order to set the limits for a better gaussian fit. The

alternative way to define these boundariers 9e.g. for very strange distributions is using the "limit" definition which sets lower and upper boundary.

For any of these forms of the option we will fit a normal distribution to the data between d1 and d2. The two boudaries are indicated by vertical pink lines. Also the "restricted" fit is given as a gauss curve in pink (next to the full range red normal fit. Outside the d1 d2 boudaries we indicate the difference between the best fit and the actual data with a green line. And also indicate which percentage of the observed points outside the boundaries is to be considered exceptional. (line in black). We indicate the 99% line. and the crosspoints with the curve. you could use this as a reasonable threshold to find mini's that are for 99% sure outside the normal noise level.

options: *nologger, limit, scale, xscale, yscale, fullframe, frame, range*:

'nologger'	switch logger off in this statement
'limit' [x1 x2]	limit the fitrange to the x-axis range between x1 -- x2.
'fullframe'	set the datarange to the full time zoom window
'frame'	set the datarange to actual timeframe from the zoomwindow.
'range' [x1 x2]	set the datarange to the hard zoomrange to x1 -- x2. You can also do this by using the Setframe command and then use "frame".

'scale'	*[x1 x2 [y1 y2]]*	standard option for graph scaling
'xscale'	*[x1 x2]*	standard option for graph x - scaling
'yscale'	*[y1 y2]*	standard option for graph y - scaling

==========

Event_Detect (num1, [numset1], [numset2], {,options})
Event detection in one channel trace of the first meting (spikes, minis) using threshold crossing.

num1,	Channel in which to detect the events
[numset1],	Threshold for spike detection (in amplitude units).
e.g:	Use two values to give direction and/or limit range (windowing):
[n1]	You may use single level: a positive n1 will be interpreted as [n1 inf], a negative n1 will be interpreted as [n1 -inf], explanation see [n1 n2] below.
[n1 n2]	E.g. [5 inf]) will detect any 5 level crossing in positive direction. [5 -inf] will do the same crossing in negative direction. While [5 10] will detect positive level 5 crossings, but peak must be smaller than 10.
[n1 n2 n3]	Adding a third number will mean that the triggerpoint will be backward reconstructed to this fractional level of the peak (useful if you intend to use Template matching as well). So if you set n3 to 0.5 the triggerpoint will ultimately end up at 50% of the peak value. this is a good idea if you are dealing with the same shape at different scaling amplitudes.
[numset2],	parameters that determine the further detection options.
[n1]	Defining one number inplies that the step observed at the trigger point needs to be at least as large as the absolute value you give here. In this way you can eliminate noise.
[n1 n2]	Provides more elaborate controle over the triggers:
n1 (ms),	This time range indicates the search range within which the peak needs to be found after the trigger point (if the last point of the range is the highest e.g. no real peak, this trigger will be ignored). Also if you define n3 in the previous set, the final trigger level needs to be found the last num3 ms before the

trigger point. Any trigger that is closer to the beginning or the end of the trace than num3 will be ignored.

n2 (ms), Minimum interval between the current trigger and the previous one (if it is closer by than num4 ms it will be ignored.

options: Valid options are *fullframe, frame, range*:

'fullframe', set the datarange to the full time zoom window
'frame', set the datarange to actual timeframe from the zoomwindow.
'range',[x1 x2], set the datarange to the hard zoomrange to x1 -- x2. You can also do this by using the Setframe command and then use "frame".

==========

Event_Template (real1 {,real2 {,real3}} {,options})

Routine does template construction and template matching for the spikes detected in Spike(s)_Detect and transfered to Events (uses same trace, channel etc).
First the mean of all events will be constructed. Next each individual will be fitted to this mean (using LSQ fit on certain range; eventually zerod). Next each event can be excluded based on the fiterror and/or fitfactor. Then using this set a new mean will be calculated for all the incl;uded events and the fit for all events to this secondary mean will be recalculated. gain factor, amplitude and error are saved. As is the mean event.

[real1 {two}] If the value of the mean fit error is larger than real1, the event will end up in the exclude list. The error is normalized for the range; on a per point basis. If this parameter is a set of two, the second one is the minimum value of the fitfactor for an event to be included in the include list. Allows to remove negative factors.

[real2 real2] If this set is present, it indicates the range to be used for nulling (assuming the start of the event is zero and the units are ms. You can use [] if you only want real3.

[real3 real3] If this set is present, it indicates range to be used for fitting (assuming the start of the event is zero and the units are ms. So you can restrict the fit to the most significant part.

options: *clear, show, save*:

'clear' will clean up the masks and creates a new mask 1.

'show' will display all events after the fir (include mean event).

'save' will save the fit parameters as data with specific meaning (display as xy). Will be overwriten by the Formula result (use templateresult as function in Formula to keep them on place 1 to 5).
1 = event time
2 = event amp (original)
3 = fit gain (event = gain * eventmean)
4 = fitted amplitude
5 = fit error (over whole or specified range).

==========

Event_Remove (num)

Will remove this mask from the set of masks.

num	Indicates the mask to be removed

==========

Spike2Triggers
Transfer all parameters of the spikes in the trace into the triggers system so that they can be used for trigger related action, mainly artefact suppression.

==========

MaskForm (text{,text{,text}})
Add new masks as logic combinations of existing maskes.
First all existing Masks are expressed as M1, M2, M3... etc
With them you can build new ones using "&", "|" and brackets. The expression should simply be a correct MATLAB fexpression. It will be added

text	MATLAB logic expression: '(M1&M3)|(M4&M5)'

==========

Event_Mask (masknr, realset1 {,realset2} {,options})
Define new masks based on selections made with parameter num in the xydata set.
realsets need a minimum of two values (or more)

masknr	choose parameter in the xydataset (as made with FormulaEventRun)

Based on what mask set do you want to make this mask, use either text or number. Options:

-- **time**	Use time of detected spike for mask
-- **amp**	Use amplitude of detected spikes for mask
-- **interval**	Make intervals and use associated spikes for mask
-- **error**	Use template match error for mask
-- **1--nn**	Any number: Use that column of the saved xy (formula) detection.

realset1	defines 2 or more levels to set the masks. Valid matlab expressions: [a b] -- defines range a--b [a:b:c] -- defines masks as a--a+b, a+b--a+2b, a+2b--a+3b, a+nb--c [a inf] -- defines >a [-inf a] -- defines <a
options:	*clear, show, save*:
'clear'	will clean up the masks and creates a new mask 1.

==========

Event_Export (text,num)

See Exporting NEURON data

==========

Spike2Events ([real1 real2])

Transfer all parameters of the spikes in the trace into the Event system as events of identical duration.

[real1 real2]	Timepoints for start and end of the selection defined in respect to the trigger (make sure that begin and end make sense. If a section defined by you falls outside the trace it is not included.
real1	Timepoint for start of the selection defined in respect to the trigger (mostly negative)
real2	Timepoint for end of the selection defined in respect to the trigger (mostly positive)

==========

Event2EP (text)
Transfer all events detected in a continuous datafile into a sweeped datafile as if they were recorded as EPs at the level of identical sweeps.

text	Name of the node in the new measurement.

==========

Event2EPdac (text)
Transfer all events detected in a continuous datafile into a sweeped datafile as if they were recorded as EPs at the level of identical sweeps. Will also transfer the dac signal into the sweeps (only one!)

text	Name of the node in the new measurement.

==========

Event_Stats (text, num, [num2 {num3}], [num2 {num3}],)
Transfer all parameters of the spikes in the trace into the Event system as events of identical duration.

text	Command to execute: *box*: make set of boxplots (followed by as many sets as you like) *compare*: compare two datasets {followed by 2 sets, to be compared by several tests.
num	Window in which to display the results
[num2 {num3}]	multiple sets of one or two parameters. num2 defines which parameter to take, while num3 defines the mask to use (if desired). num2 has the following meaning: 1 / N: parameter number from the Formula set -1: time -2: event trigger point -3: event amplitude -5 error from template fit

==========

PCA

==========

PCA (num1, num2, num3, num4, text)
Execute Principal Component Analysis

-- **num1** list of measurements to use for PCA
-- **num2** one or more starting point for PCA time stretch in vector
-- **num3** duration of each time stretch in the vector
from these numbers the display range will be constructed and added to the modifiers
-- **num4** window in which where PCA will be performed
-- **text** text containing modifiers of data to use for PCA (channels, sweeps, filtering etc)

There is a set of plot modifiers that can be used to watch the result (still a bit primitive, needs improvement, for overview see the plot modifiers)

==========

Datasets

==========

Set_Load (num1, text)

Command looks like For All, but now all the datafiles in the directory that obey the mask will be loaded into memory.

num1 The depth level of directories used is given by number (0 = only in current dir, inf,= use all subdirectories

text macrotext to execute for each file that is loaded for the set. Sets appear as single lightblue nodes in the time line. Clicking them makes them into the current Meting.

==========

Set_Save

Save the current Meting into the set (only in memory!)

==========

Set_Get (num1)

num1 Make this meting the actual Meting

==========

Set_FileLoad (num1, text)

Load a *.mat file that contains a saved Dataset file. It simply restores the situation that was saved. Only files saved with the next command can be restored.

==========

Set_FileSave (num1, text)

Load a *.mat file that contains a saved Dataset file. It simply restores the situation that was saved. Only files saved with the next command can be restored.

==========

Set_Replace (num1)

num1 Save actual Meting in set and make meting number the actual one.

==========

SetToNodes

If you have a series of measurements that consist of single nodes of similar composition, you can merge them into one measurement (file). The name of that file will be constructed from the name of the first and the last file in the set. The timing of the nodes will keep the original timing of the measurements. Files are sorted alphabetically.

Procedure: Load the data as a set (or as a series in the load window), Then invoke the macro.

==========

SetToTrace

If you have a series of measurements that consist of traces that were successively measured, but could logically form one Trace, you can merge them into one measurement (file). The name of that file will be constructed from the name of the first and the last file in the set. The timing of the trace parts will keep the original timing of the measurements. datapoints in the gaps are interpolated from the last value of the previous trace to the first values in the next trace.
You may take as many channels as you like, but only one node per file (at the moment).

Procedure: Load the data as a set (or as a series in the load window), Then invoke the macro.

==========

Set_Combine

==========

Set_Cell_Load

==========

Set_Keep (num1, text)

==========

Set_Subtract

==========

Set_Run (macro, macro1, macro2, macro3)

Will run this set of macros over all sets of Metingen.

macro1	Macro to run before the loop
macro2	Macro to run over each member in the loop
macro3	

==========

Extract (stringset)

This function will extract all the nodes with names defined by stringset from the DATAsets in memory and build a new Meting set with those nodes. The old dcataset is killed afterwards, A complete set is build, filenames of the original files that provided the nodes are kept for potential later use

stringset	each string in the set (texts sparated by ; like: '**text1;text2;text3;etc**') should define a unique node with a measurement in a single component of the set.

==========

Binding (name, num1,cmd1(),cmd2(),cmd3(),lim1,lim2,lim3,
***fitspecs(parameters)*)**

Analyse concentration dependent binding from a set of binding curves extracted from a dataset of measurements (specifically made for Xin binding analysis, but the concept could be applied)

name	name that uniquely defines the nodes that contribute to the binding analysis.
num1	maximum time interval between 2 timepoints to allow in the drift analysis. If the interval is larger it will be limited to this value. This allows you to do a drift correction of several measurements, performed days apart)
cmd1(pars)	determines 1e parameter to use in the analysis.
cmd2(pars)	determines 2e parameter to use in the analysis.
cmd3(pars)	determines 3e parameter to use in the analysis.
	for the binding analysis the required set is:
time(60)	make the time axis and express it in minutes.
drug	we need to have the drug concentration
xyresult(1,2)	use columns 1 and 2 from the node as the basic data
fitspecs(parameters)	Defines how to fit the single binding curves, for straight forward binding analysisd the prefered fit will be:

fitspecs(Expo,1,0,'offset')

lim1 '[left right]' string (obligatory!) that defines the x range to include in the tau fit.
lim2 '[left right]' string (obligatory!) that defines the x range to include in the tau fit.
lim3 '[left right]' string (obligatory!) that defines the x range to include in the tau fit.

==========

Shifting (name, num1,cmd1(),cmd2(),cmd3(), *fitspecs(parameters)*)

Analyse concentration dependent binding from a set of binding curves extracted from a dataset of measurements (specifically made for Xin binding analysis, but the concept could be applied)

name name that uniquely defines the nodes that contribute to the binding analysis.
num1 maximum time interval between 2 timepoints to allow in the drift analysis. If the interval is larger it will be limited to this value. This allows you to do a drift correction of several measurements, performed days apart)
cmd1(pars) determines 1e parameter to use in the analisis.
cmd2(pars) determines 2e parameter to use in the analisis.
cmd3(pars) determines 3e parameter to use in the analisis.

for the shifting analysis the required set is:

time(60) make the time axis and express it in minutes.
drug we need to have the drug concentration
fitpar(2) use the second parameter of the fitnode (was a Boltzmann, e.g. Vh)

fitspecs(parameters) Defines how to fit for the concentration dependence of the 3e parameter given above as a function of the 2e parameter.
For straight forward shift analysisd the prefered fit will be:

fitspecs(logistic,1)

==========

Segments

==========

Segment_Merge (num)

Merge segments into a new segment: a new file is generated in the subdirectory “new”

num	how many segments to merge into a new one:
	-- inf merge all segments
	-- anynum: e.g. merge two segments into one new one etc
	-- [numset]: merge this set into a new one (not yet implemented!

==========

Segment_Split (num)

Split the data into new segments, will use the current eventset (both starts and end times as split moments for the new data, a new file is generated in the subdirectory “new”

num	minimumtime in (ms) that is needed for a segment

==========

Reload

Coversion Routines for NEURON

There are several routines that can be used to reorganize data within NEURON so that the analysis or exportation is easier. There are several reasons why you may want this: 1) to adapt older files to newer analysis options. 2) because it was impossible to organize the data already the way you want it during the measurements. 3) The data is a composition of several sessions.

Most of the time (at least I try to let it work that way) tghe rearranged data will be dumped under an understandible name (that relates to the original one) in a subdirectory "new" in the data location. This implies that you should not try this while reading from a CD/DVD. Copy the data first to your computer and then convert/rearrange it. Most of these conversions were made for special cases, so careful check whether the requirements are met.

Internal organization (sweeps, traces, channels)

==========

Collaps

Merge all nodes with identical names into a collapsed measurenment. (create sweeps for averaging etc), mainly for measurements that were done before the online averaging options.

Exporting results and raw from Neuron

There are two fundamental different ways of exporting data from NEURON.

The standard way is using the Logger which will automatically write the results of all important functions (formula, fit, etc) to an EXCEL or TEXT file in the most appropriate format.

The alternative is that you export a definable portion of the raw data (potentially modified by commands that work on the data, be aware: not the ones that work on the display) to either a MATLAB file that you can use for further calculations in MATLAB, or to a (TAB separated) TEXT file that can directly be read into KALEIDAGRAPH, EXCEL or many another plotting/analysis program. Be careful, because in this mode many context dependent features will get lost. It is your duty to remember where it came from, what the scaling was etc, etc...

Logger

The basic idea behind logger is simply to record whatever is relevant in the analysis. This implies that you have no further control over LOGGER than switching it off or on, either temporarily or final, but it determines by itself (= by me) what will be saved (shout if you have desires in this area, anything can be added).
LOGGER was in principle designed to write directly into "xls" excel files and can use all the options available there. In particularly writing to separate datasheets in de excel file. It can operate in two distinct modes, you define the mode when you initiate Logger
It mostly generates an excel sheet for each analysis, you can specify its name and continue to write to it, or address new sheets as you like. Realize that there is no control over overwriting, this occurs automatically (and without erasing the old data completely if you select an existing file).

LOGGER determines whether an EXCEL COM server is present on your computer and by default will write to an excel file. If not (at the moment on the MAC for example) it will produce a text file (extension txt). These txt files are TAB separated and can be read into EXCEL (or the like). As textfiles have no subsheets, I have tried to come with acceptable alternatives for this mode..

There is one convenient MACRO command that can be used to overrule all coming LOGGER commands. In this way you can put logger commands in your script in the test phase of your analysis, but prevent spending time and clogging up your directories with EXCEL/TXT files. The LoggerOff/LoggerOn command also opens the possibility to overrule the txt/xls mode by defining the (only) text parameter that goes with it:

==========

LoggerOn({text})

-- Overrules all other Logger commands (default = ON), easy to put into your Base commands if you do not want to generate all the logger commands during a test phase

text: Type of file used for output, currently 2 options:
'xls' will produce classical multisheet excel file, default on a PC
'txt' will produce tab separated textfile (can be read in into excel). default for MAC
anything else generates an error.

==========

LoggerOff

-- Overrules all other Logger commands, by switching them off. This prevents cluttering your directories with files that you do not want while testing out your scripts/macros.

==========

There are also two commands that can temporarily stop writing data to the Logger file and resume it later. (You have to keep track yourself what the status is, if you switch modes in different macros!)

==========

Logger('halt'): stop logging (temporally)

==========

Logger('resume'): resume the logging that was stopped with the 'stop' command

==========

Result files can be organized in two different formats:
Sheet format, assumes that you dump the results in a two dimensional EXCEL sheet, oirganized the way NEURON wants it
Kolom format, assumes that you dump each analysis (mostly on a datafile to datafile basis) in one column of the EXCEL sheet, each value in the same column. New datafile analysis then give you a next column, which will facilitate the later averaging and statistics greatly if you ran the same protocol over many cells, files.
There are options to synchronize and solve problems of missing data.

If you cleverly design the column strategy you can even redo such an analysis with a different number of files or with a slightly different macro. Think about that strategy when you define the EXCEL file analysis, it can save you days of work and a mouse arm!

==========

Logger(text1, *{text2, text3, {nr}}*)
Logger controls the performance of the Logger during the analysis. Its main purpose is to write data into an EXCEL file during most of the standard analysis options of Neuron The result is context dependent, most serious analysis commands generate data and comments.

Meaning of **text1**:

'halt': see above

'resume': see above

'sheet': start logging in sheet format. You can start writing to the next sheet by giving the **next** command. At that point the current file is saved (to prevent loosing all if you crash somewhere). It will be located in the directory where the data resides.

or

'kolom': start logging in column format. You can start writing to the next column by giving the **next** command. At that point the current file is saved (to prevent loosing all if you crash somewhere) but continued to be filled.

text2 is obligatory and the meaning is context dependent:

somename: Name of the txt/excel file to be generated (no extension needed)
If you use "**file**" this means that the current file name be used, changing its extension from .NRN to **.txt** or **.xls**. **Files will be stored in the same directory as the datafiles. They are automatically overwritten.**

If you use "**dir**" this means that the file name will be the name of the current directory and start with the name of the directory of the NRN datafile and it will be stored at the same level as that directory
Otherwise somename will be used (**automatically overwritten**).

text3 is optional and most relevant in xls mode

sheetname: In excelfile mode this is the name of the sheet in the excelfile. In text mode this name will be added to the filename.
If not given in xls mode name will be the same as **'filename'**, but in txt mode it will be empty.
If you use "**file**" this means that the sheet name will start with the name of the last NRN datafile that was used

nr: Offset to use in column writing: Default is 0, No offset in use. If offset is not equal 0 then the first column with comments will not be written. In textmode mode nr can not be used as it is very difficult to add columns to a textfile

---

'next': start with the next column in column format, or with the next sheet in sheet format (keeping the same file) or the next datablock in multisheet format.
text: Add **text** to the sheetname at saving it.

'save': identical to next, but clears all the intermediates and resets counting
text: Add **text** to the sheetname at saving it.

Examples:

Logger('sheet', 'dir', 'file')

In 'xls'mode will generate an EXCEL file:
In the directory of the current directory (using the name of the current diractory as file name) and writing each dataset to a sheet with the file name of the current data to analyze, assuming you give a next.
In 'txt' mode this command will generate a set of TXT files:
In the directory of the current directory, using that name and concatenate it with the dafafile name, eventually also using an extension given with the next/save command.
You need to close the Logger with
Logger('save')
before you get a file!
Giving
Logger('next',{'extra'})
In 'xls'mode will generate an EXCEL file:
In the directory of the current directory (using the name of the current directory as file name) and writing each dataset to a sheet with the file name of the current data (plus _extra) to analyze.
In 'txt' mode this command will generate a set of TXT files:
In the directory of the current directory, using that name and concatenate it with the dafafile name (plus _extra).

Logger('sheet', 'name1', 'name2')
In 'xls'mode will generate an EXCEL file:
With name1 and sheet name2, localized in the directory of the data
In 'txt' mode this command will generate a TXT file:

With name name1_name2, localized in the directory of the data
Example:
Logger('kolom', filename {, sheetname{, nr}})

==========

Exporting raw data

The other way to export data is to use the Export function. This is the format to use if you do not want to have the analysis results but just raw data traces or event analysis. You can pretreat these traces with commands that affect the data itself (artifact suppression). But be aware that treatment of the display 9as in the display command lines, will not have any effect. Only provedures that treat the data in memory

Export (txt1, txt2, num1 {options})

Export provides you with several ways of outputting the raw data from NEURON files.

txt1: defines the output format. At this moment there are two choices.
two alternatives:

'mat' Output as a MATLAB matrix format, essentially a 2-dimensional matrix that gives you what is defined. As a first column time is given in units of your current display mode (ms,sec,min,hrs,points). The other columns are the requested displayed signals (if they are not of equal length, they will be 0-padded.

'txt' The alternative is to write data in text format to be imported in for example EXCEL or KALEIDOGRAPH. In this format the data is in TAB separated column format and a header (channel name if the second dimension is channel and the units are given. As a first column time is given in units of your current display mode (ms,sec,min,hrs,points).

'events' Export details of the events, you can use masks. One more parameter is needed that defines which elements(s) will be exported.

'text' Text contains ";" separated names that define what will be exported.
Possibilities are: 'Spikes;XY;Events' in any combination.
-- "**Spikes**" exports the 5 spike parameters
-- "**XY**" exports the mini analysis data
-- "**Events**" exports the relevant Event parameters.
-- "**Mask**" export the masked values.

optional:
-- mask (numset) select mask to use for output (default = -1)

The second parameter (only for mat of txt, not for events) defines what will be exported:
==========

txt2: defines what will be outputted.
options: there are two fundamental approaches:

'last' put out the last image that you have displayed in the last used window including the selected time span. To be sure, just use only one window. definitions of Channels, Sweeps, Measurements, Lists, Index hare ignored, but.

==========

'adc', **'avg'**, **'dac'** if you do not use **last**, you have to define what you want to output. At this moment you can only choose from:

The third parameter controls the time axis to be used when exporting traces:

==========

numset: time specifications with the following options:

[]	export all datapoints
[step]	use all points but skip "step" **datapoints (default in pnts)**
[start end]	begin and end of the trace, export all (**current time units**)
[start step end]	start and end as defined above, step as well

for the time specifications the following options can be used: ms, sec,min,hrs,points. For a single value, points is the default, otherwise the current set is the default, otherwise you overrule.

==========

other options can further specify what to use from the measurements (segments, not yet included):

optional:

-- list (num)	select the measurements from the list, num obligatory	
-- mets (numset)	select measurements	(default = 1, all = inf)
-- chans (numset)	select channels	(default = 1, all = inf)
-- swps (numset)	select sweep	(default = 1, all = inf)
-- index (set1,set2,set3)	select all three	(default = 1,1,1 all = inf, inf,inf)

==========

The Outputfile will get the same name as the file from which the data is taken, with an expansion 'exportnnnn' where nnnn is a follow number. Be aware that numbering restarts if you read a file and that files are automatically overwritten. (This also happens if you output more than 999 files (if that happens think about a more clever approach!

Examples:

Export = Export ('txt', 'last',1)
Export ('mat') = Export ('mat', 'last',1)
Export ('mat', 'adc')
Export ('mat', 'dac')
Export ('mat', 'adc', [10 1 20] ,index(1, 1, 1))
Export ('txt', 'avg', [10 2 20] ,index(inf,inf,inf))
Export ('txt', 'last', 5,'ms')
Export ('txt', 'last', 5,'points')
Export ('events', 'XY;Events', mask([2 3 4]))

Running a superset to test Neuron macros

With the superset knob you can run a macrotext file with primitive commands that can test a lot of macros. Mainly used to test whether all previously defined macros still run after an update has been made.
The set is simple, only one comand per line, which is interpreted and executed, here are the posible commands:

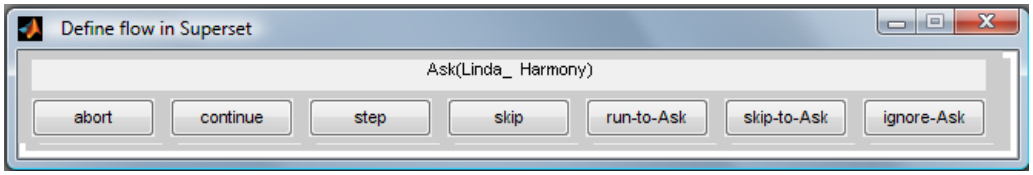
==========

Ask(command)

-- This command allows you to get a question about what to do with the instruction "command"

command: Any command that can be executed as a Superset instruction.

Will generate the following pop-up panel



The top panel gives the current line in execution (most interesting in the step mode.

You can select any of the following responses:

abort: abort the superset 9and other macros
cointinue: run untill a next stop
step: continue stepwise execution.
skip: skip this instruction.
run-to-ask: contunue execution until the next Ask command in the supersetfile.
skip-to-ask: skip over all statements until the next Ask command in the supersetfile.
ignore-to-ask: contunue execution and even skip the Ask commands (i.e. run all statements)

==========

Ask(text)

-- Execution will stop here and the Superbox will pop=up so that you can influence the command flow.

text: Text to be added to the Question box

==========

Loadmacro(text)

-- Load a macro file (default from macros directory in the NEURON directory

text: File name of the text file that contains the macro.

==========

Loaddata(text)

-- Load a data file Give the full file name (directories etc). Only NRN files can be loaded, no data conversions yet.

text: File name of the data file that will be loaded.

==========

Loaddir(text)

-- Load a directory as a set. Give the full file name of the directory (directories etc). All *.mat datafiles in that directory will be loaded.

text: File name of the directory, from which all files will be loaded.

==========

Runkey(keylist)

-- Run one or more macros in the macrofile over the current datafile.

keylist: list of numbers of macro keys in the macro to run. Count from one (upper knob, downwards.

==========

Setkey(keylist)

-- Run one or more macros in the macrofile over all the datasets.

keylist: list of numbers of macro keys in the macro to run. Count from one (upper knob, downwards (same as Runkey, but will run over all sets of data.

==========

Hitknob

-- Future options that will invoke an artificial knob press (you have to know its handle for it to work!)

==========

Wait(number)

-- Set the value to use in the pause command.

number: waittime in seconds (or 0, or inf = stepmode).

==========

Done

-- Will break the surerrun

==========

''

-- Empty lines have no effect

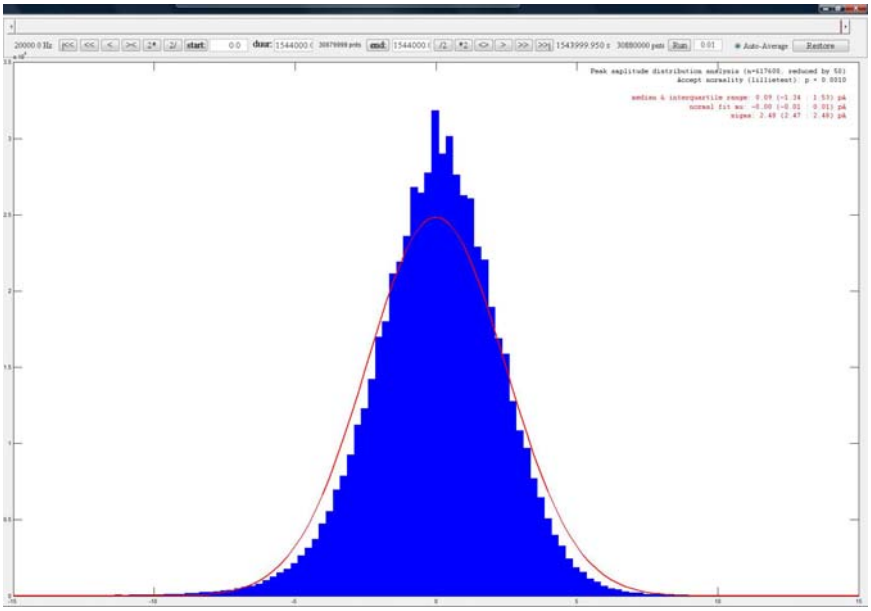
==========

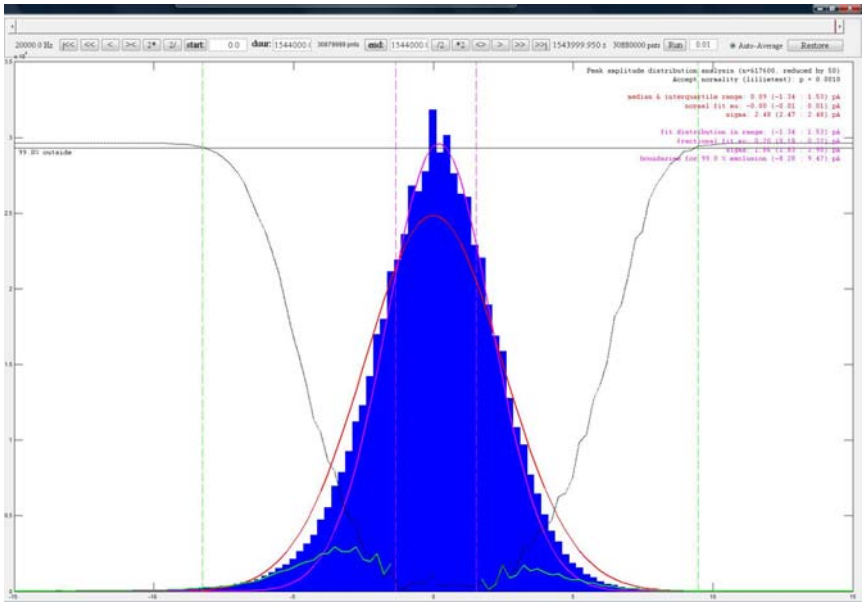
Explaining Specialized procedures in Neuron

Leak correction procedure

PCA analysis

Spike/Mini detection and analysis





Appendix A: Hardware documetation

Interface EEG opstelling "wireless"

Interface in de kast naar NI aansluiting:

National Instruments 68-Pin I/O Connector

Nr	Omschrijving	Nr	Omschrijving
1	Freq-Out	35	Digital Ground
2		36	Digital Ground
3		37	
4	Digital Ground	38	
5		39	Digital Ground
6		40	
7	Digital Ground	41	
8	+5 Volt	42	
9		43	
10		44	Digital Ground
11		45	
12	Digital Ground	46	
13	Digital Ground	47	Digital I/O Channel 3
14	+5 Volt	48	Digital I/O Channel 7
15	Digital Ground	49	Digital I/O Channel 2
16	Digital I/O Channel 6	50	Digital Ground
17	Digital I/O Channel 1	51	Digital I/O Channel 5
18	Digital Ground	52	Digital I/O Channel 0
19	Digital I/O Channel 4	53	Digital Ground
20	ExtRef	54	Analog Output Ground
21	DAC-Out Channel 1	55	Analog Output Ground
22	DAC-Out Channel 0	56	Ground
23	**Channel 15 Input**	57	**Channel 7 Input**
24	Ground	58	**Channel 14 Input**
25	**Channel 6 Input**	59	Ground
26	**Channel 13 Input**	60	**Channel 5 Input**
27	Ground	61	**Channel 12 Input**
28	**Channel 4 Input**	62	Analog Input Sense
29	Ground	63	**Channel 11 Input**
30	**Channel 3 Input**	64	Ground
31	**Channel 10 Input**	65	**Channel 2 Input**
32	Ground	66	**Channel 9 Input**
33	**Channel 1 Input**	67	Ground
34	**Channel 8 Input**	68	**Channel 0 Input**

Aansluiting naar 68 pins Circuit Board inside the amplifier. Other conectors at the back m(50-pole and 4 BNCs are currently not connected

Digital inputs outputs

P0.0 = 52
P0.1 = 17
P0.2 = 49
P0.3 = 47
P0.4 = 19
P0.5 = 51
P0.6 = 16
P0.7 = 48

Sense digital inputs

PFI 0 = 11
PFI 1 = 10
PFI 2 =
PFI 3 =
PFI 4 =
PFI 5 =
PFI 6 = 5
PFI 7 =

P2001-464 NI-DAQ Interface

Nr	Omschrijving	Kleur
1	Channel 1 (Molen 1)	bruin
2	Channel 1 Return (GND)	rood
3	Channel 2 (Molen 1)	oranje
4	Channel 2 Return (GND)	geel
5	Channel 3 (Molen 1)	groen
6	Channel 3 Return (GND)	blauw
7	Channel 4 (Molen 1)	paars
8	Channel 4 Return (GND)	grijs
9	Channel 5 (Molen 2)	wit
10	Channel 5 Return (GND)	zwart
11	Channel 6 (Molen 2)	bruin
12	Channel 6 Return (GND)	rood
13	Channel 7 (Molen 2)	oranje
14	Channel 7 Return (GND)	geel
15	Channel 8 (Molen 2)	groen
16	Channel 8 Return (GND)	blauw

Connection from central board to the NIDAQ connection board

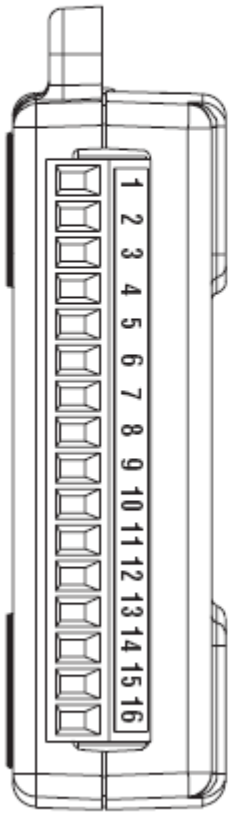
Connectors on the front (input from 8 headstages):

P2001-464 Molen Interface

BlueRibbon 24-polig L	
Nr	Omschrijving
1	Chan 1 +Input
2	Chan 2 +Input
3	Chan 3 +Input
4	Chan 4 +Input
5	
6	
7	
8	
9	Stimulus 1
10	Stimulus 2
11	Stimulus 3
12	Stimulus 4
13	Chan 1 -Input
14	Chan 2 -Input
15	Chan 3 -Input
16	Chan 4 -Input
17	GND
18	GND
19	Reference (Cal+GND)
20	Stimulus GND
21	- 15 V (-Vcc)
22	
23	
24	+ 15 V (+Vcc)

BlueRibbon 24-polig R	
Nr	Omschrijving
1	Chan 5 +Input
2	Chan 6 +Input
3	Chan 7 +Input
4	Chan 8 +Input
5	
6	
7	
8	
9	Stimulus 5
10	Stimulus 6
11	Stimulus 7
12	Stimulus 8
13	Chan 5 -Input
14	Chan 6 -Input
15	Chan 7 -Input
16	Chan 8 -Input
17	GND
18	GND
19	Reference (Cal+GND)
20	Stimulus GND
21	- 15 V (-Vcc)
22	
23	
24	+ 15 V (+Vcc)

Table 4. Analog Terminal Assignments

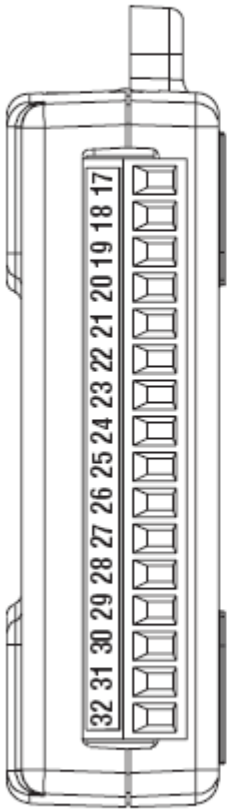
Module	Terminal	Signal, Single-Ended Mode	Signal, Differential Mode
	1	GND	GND
	2	AI 0	AI 0+
	3	AI 4	AI 0–
	4	GND	GND
	5	AI 1	AI 1+
	6	AI 5	AI 1–
	7	GND	GND
	8	AI 2	AI 2+
	9	AI 6	AI 2–
	10	GND	GND
	11	AI 3	AI 3+
	12	AI 7	AI 3–
	13	GND	GND
	14	AO 0	AO 0
	15	AO 1	AO 1
	16	GND	GND

Sub ADC system with NI6009

Connections to the analog side:
AI0 = current VC or Voltage (CC), scaled by gain on channel 7
AI1 = voltage (unscaled)
AI2 = not in use....
AI3 = temprature C (from pay heater controller)
AI4 = capacitance telegraph
AI5 = filter frequency ttelegraph
AI6 = mode of the amplifier (VC etc
AI7 = gain selector of the amplifier

AO0 = ene helft van de soft loop control
AO1 = andere helft van de soft loop control

Table 5. Digital Terminal Assignments

Module	Terminal	Signal
	17	P0.0
	18	P0.1
	19	P0.2
	20	P0.3
	21	P0.4
	22	P0.5
	23	P0 6
	24	P0.7
	25	P1.0
	26	P1.1
	27	P1.2
	28	P1.3
	29	PFI 0
	30	+2.5 V
	31	+5 V
	32	GND

Sub ADC system with NI6259 / NI6229

(difference is in the speeds)

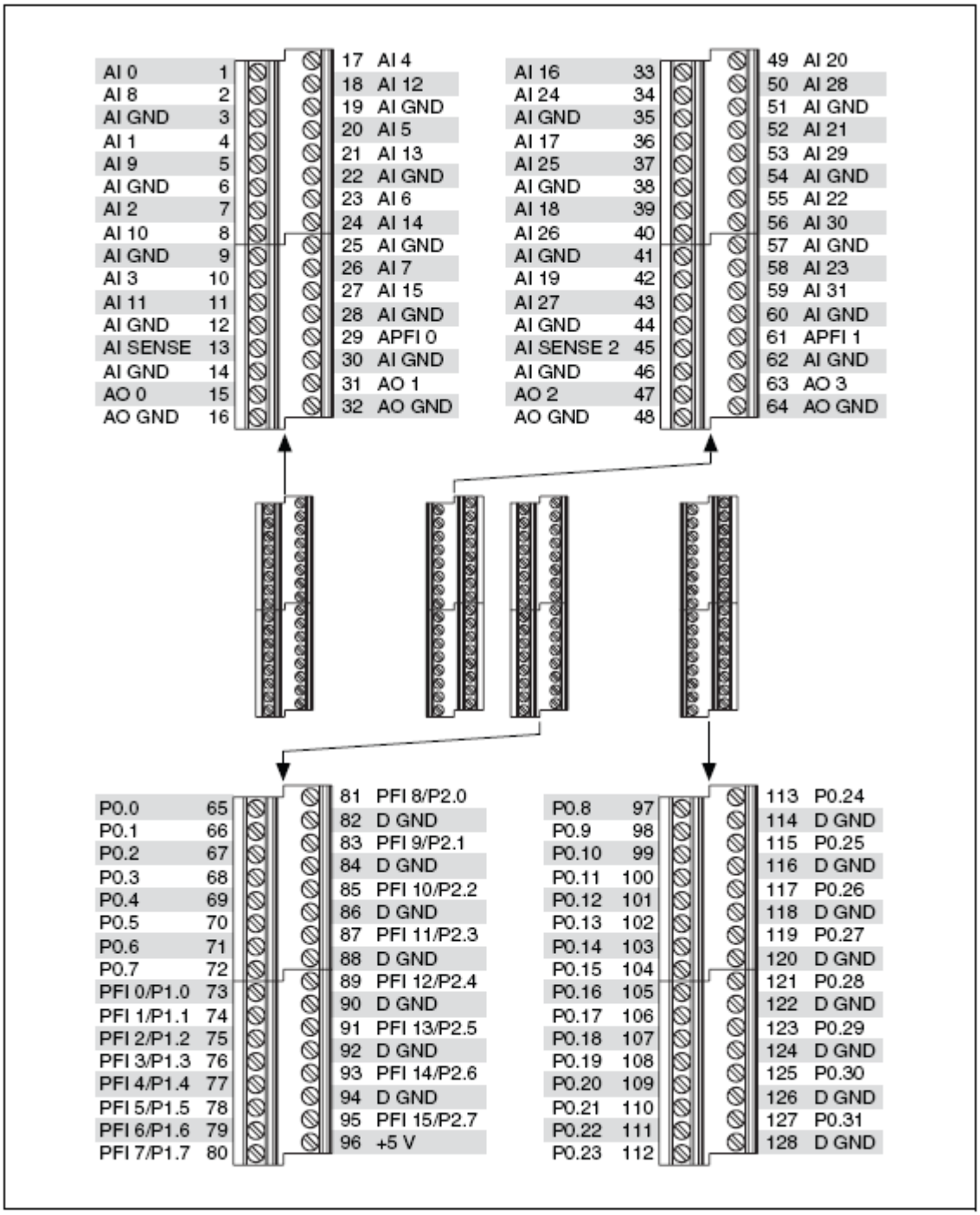


Figure A-20. USB-6259 Pinout

Sub ADC system with NI6251 / NI6221

(difference is in the speeds)

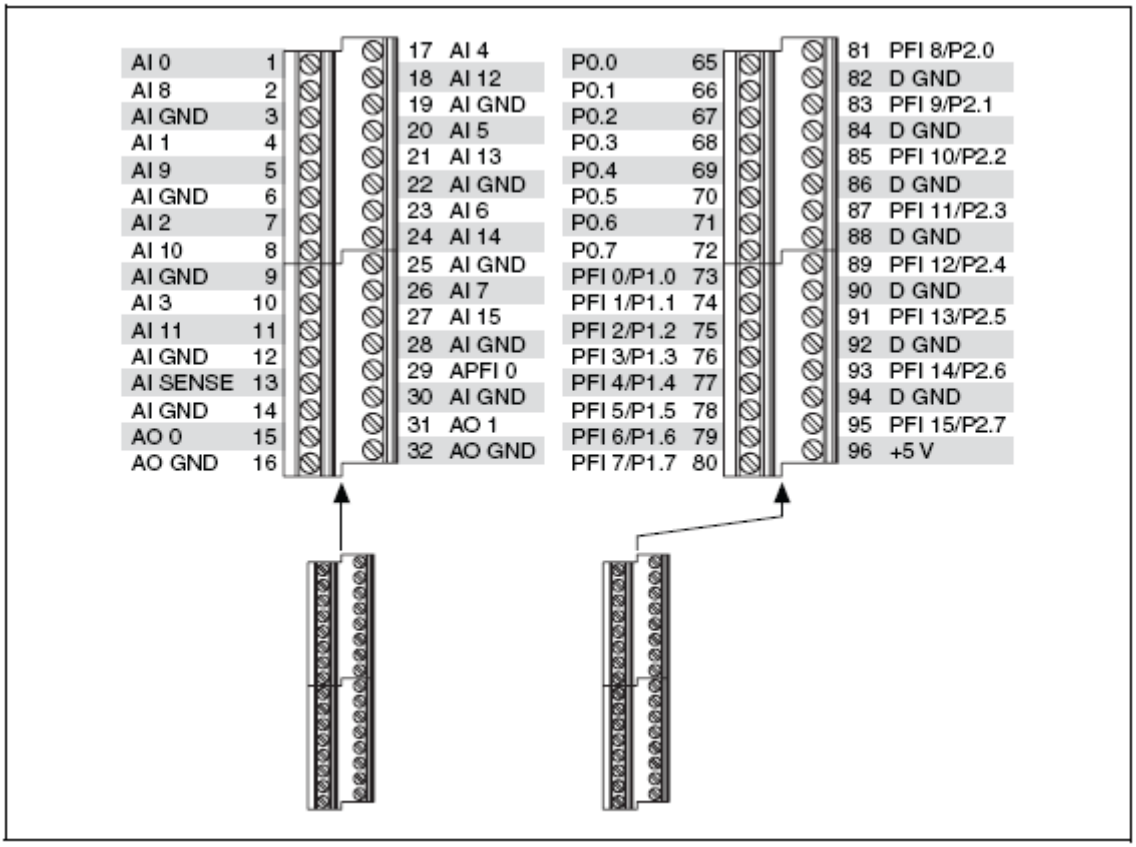


Figure A-13. USB-6251 Pinout

Index to protocol, macro, display commands:

/ **nosymbol** *(modifier)* 20
/ **PCAc** *(modifier)* 24
/ **PCAv** *(modifier)* 24
/**adc** *(modifier)* 11
/**any** *(modifier)* 11
/**avg** *(modifier)* 11
/**blank** *(modifier)* 16
/**bpass** *(modifier)* 19
/**bstop** *(modifier)* 19
/**catch** *(modifier)* 12
/**color** *(modifier)* 20
/**colormap** *(modifier)* 17
/**cross** *(modifier)* 16
/**csd** *(modifier)* 18
/**dac** *(modifier)* 12
/**diff** *(modifier)* 17
/**event** *(modifier)* 13
/**femke** *(modifier)* 22
/**fit** *(modifier)* 12
/**foto** *(modifier)* 12
/**gain** *(modifier)* 16
/**grid** *(modifier)* 19
/**group** *(modifier)* 21
/**hist** *(modifier)* 12
/**hpass** *(modifier)* 18
/**legend** *(modifier)* 19
/**line** *(modifier)* 16
/**logx** *(modifier)* 20
/**logxy** *(modifier)* 20
/**logy** *(modifier)* 20
/**lpass** *(modifier)* 18
/**map** *(modifier)* 16
/**marker** *(modifier)* 20
/**markers** *(modifier)* 20
/**mask** *(modifier)* 21
/**metdac** *(modifier)* 21
/**mets** *(modifier)* 12
/**mpass** *(modifier)* 19
/**mstop** *(modifier)* 19
/**mxy** *(modifier)* 14
/**mxyc** *(modifier)* 15
/**mxyd** *(modifier)* 15
/**mxym** *(modifier)* 15
/**mxyr** *(modifier)* 15
/**mxyv** *(modifier)* 15
/**noline** *(modifier)* 19
/**none** *(modifier)* 11
/**nonull** *(modifier)* 18
/**resample** *(modifier)* 18
/**sd** *(modifier)* 17

/sem *(modifier)*....17
/setx *(modifier)*....21
/setxy *(modifier)*....21
/sety *(modifier)*....21
/shift *(modifier)*....21
/spectrogram *(modifier)*....23
/spectrum *(modifier)*....22
/submean *(modifier)*....18
/swps *(modifier)*....12
/symbol *(modifier)*....20
/tdiff *(modifier)*....17
/template *(modifier)*....21
/type *(modifier)*....20
/vsd *(modifier)*....17
/width *(modifier)*....20
/xy *(modifier)*....13
/xyc *(modifier)*....14
/xyd *(modifier)*....14
/xyevent *(modifier)*....21
/xym *(modifier)*....14
/xyr *(modifier)*....14
/xyv *(modifier)*....14
/zero *(modifier)*....18
/Zero *(modifier)*....18
/zoom *(modifier)*....19
/zoomlines *(modifier)*....19
ADCmode *(protocol)*....30
ADCrate *(protocol)*....44
Add *(macro)*....67
alfa *--alfa (protocol)*....41
always*/modifier*....21
Amplifier *(protocol)*....30
Amplitude *(macro)*....97
Annotate *(macro)*....64
Artefact *(macro)*....90
Artefact *(protocol)*....42
Artefact_Trigger *(macro)*....91
Artifact_List *(macro)*....90
Ask *(superset)*....113
AutoMacro *(macro)*....90
Average *(macro)*....71
Average *(protocol)*....45
avg *(formula_macro)*....82
Barcode *(protocol)*....41
Basis *(protocol)*....29
Beep *(protocol)*....45
Binding *(macro)*....104
Bitdef *(protocol)*....43
Bitset *(protocol)*....43
bpass *(macro)*....69
bstop *(macro)*....69
cabs *(macro)*....69

Capacitance *(macro)* 64
Catch *(protocol)* 35
CCProtocol *(protocol)* 28
chirp *--chirp (protocol)* 41
Clear *(macro)* 62
Clear *(protocol)* 44
ClearNow *(protocol)* 43
Collaps *(macro)* 107
Colormap *(protocol)* 36
condition *(formula_macro)* 83
Copy *(macro)* 67
DACHold *(protocol)* 42
DACpnt *(formula_macro)* 84
DACrate *(protocol)* 44
Delay *(protocol)* 45
Delete *(macro)* 67
Detect *(macro)* 92
Detect *(macro, see Detect)* 92
Digs *(protocol)* 40
Divide *(macro)* 68
Done *(superset)* 114
Downscale *(macro)* 67
Draw *(macro)* 65
Drugs *(macro)* 64
EEGpanel *(protocol)* 51
empty *(superset)* 114
EPpanel *(protocol)* 50
EqualDAC *(protocol)* 45
Evalm *(macro)* 61
Evalp *(macro)* 61
Event_Detect *(obsolete macro)* 98
Event_Export *(macro)* 100
Event_Mask *(macro)* 100
Event_Remove *(macro)* 100
Event_Stats *(macro)* 101
Event_Template *(macro)* 99
Event2EP *(macro)* 101
Event2EPdac *(macro)* 101
EventSaveOff *(macro)* 61
EventSaveOn *(macro)* 61
ExpandChannels *(macro)* 68
Export *(macro)* 111
extmean *(formula_macro)* 83
extp *(formula_macro)* 83
extr *(formula_macro)* 83
Extract *(macro)* 104
Fcell *(macro)* 64
Fdate *(macro)* 64
Ffile *(macro)* 63
Filesystem *(protocol)* 44
Firing *(macro)* 76
first*/modifier* 21

Fit *(macro)* 77
fitrace *(formula_macro)* 80, 87
Forall *(macro)* 74
Formula_Run *(macro)* 81
Formula_Set *(macro)* 81
Ftime *(macro)* 64
FullOff *(macro)* 61
FullOn *(macro)* 61
getfreq *(formula_macro)* 84
Hitknob *(superset)* 114
hpass *(macro)* 69
Impedance *(protocol)* 35
Impedance1 *(protocol)* 35
Impedance2 *(protocol)* 35
Interval *(protocol)* 44
JunctionCC *(protocol)* 32
JunctionVC *(protocol)* 32
Keep *(macro)* 66
Leak *(macro)* 70
LeaveOnEmpty *(macro)* 65
lin *(formula_macro)* 82
List *(macro)* 65
Loaddata *(superset)* 113
Loaddir *(superset)* 113
Loadmacro *(superset)* 113
Logger *(macro)* 109
LoggerOff *(macro)* 108
LoggerOn *(macro)* 108
lpass *(macro)* 68
Macro *(macro)* 57
MaskForm *(macro)* 100
maxmean *(formula_macro)* 83
maxp *(formula_macro)* 83
maxr *(formula_macro)* 83
maxtrain *(formula_macro)* 84
minmean *(formula_macro)* 82
minp *(formula_macro)* 82
minr *(formula_macro)* 82
mintrain *(formula_macro)* 84
Modus *(protocol)* 31
Movie *(macro)* 66
mpass *(macro)* 70
Music *(macro)* 64
Names *(macro)* 75
Names *(protocol)* 36
never/*modifier* 21
NoEEG *(protocol)* 32, 44
OriginalOff *(macro)* 61
OriginalOn *(macro)* 61
Overdracht *(macro)* 73
Pause *(macro)* 63
PCA *(macro)* 73, 102

PCAconstruct *(macro)*....73
pike_Artifact *(macro)*....90
Plotter *(protocol)*....43
pnt *(formula_macro)*....82
pntbig *(formula_macro)*....82
Poisson *(protocol)*....39
power *(macro)*....69
Power *(macro)*....68
Protocol *(protocol)*....28, 29
Puls *(protocol)*....40
Rampup *(protocol)*....42
RecordSweep *(protocol)*....33
RecordTrace *(protocol)*....34
ReduceChannels *(macro)*....68
Reload *(macro)*....106
Reload *(macro)*....70
Remove *(macro)*....67
RemoveEmpty *(macro)*....66
Resample *(macro)*....92
RunDraw *(macro)*....65
Runkey *(superset)*....114
Samplerate *(protocol)*....44
Seal *(protocol)*....32
Segment_Merge *(macro)*....106
Segment_Split *(macro)*....106
Set_Cell_Load *(macro)*....104
Set_Combine *(macro)*....104
Set_FileLoad *(macro)*....103
Set_FileSave *(macro)*....103
Set_Get *(macro)*....103
Set_Keep *(macro)*....104
Set_Load *(macro)*....103
Set_Replace *(macro)*....103
Set_Run *(macro)*....104
Set_Save *(macro)*....103
Set_Subtract *(macro)*....104
SetADCrate *(macro)*....66
SetDACrate *(macro)*....66
SetDAQconvert *(macro)*....66
Setframe *(macro)*....63
Setkey *(superset)*....114
SetPause *(macro)*....63
SetSlope *(macro)*....81
SetTextFile *(macro)*....75
SetTime *(macro)*....62
SetToNodes *(macro)*....103
SetToTrace *(macro)*....103
SetWindow *(macro)*....64
Shifting *(macro)*....105
sine *--sine (protocol)*....41
SkipDrawOff *(macro)*....62
SkipDrawOn *(macro)*....61

SkipFitOff *(macro)* 62
SkipFitOn *(macro)* 62
slope1 *(formula_macro)* 83
slope2 *(formula_macro)* 83
Snapshot *(macro)* 66
SnapshotOff *(macro)* 66
SnapshotOn *(macro)* 66
SoftClamp *(protocol)* 32
Spectra *(macro)* 71
spike *(formula_macro)* 83
Spike_Amplitude *(macro)* 95
Spike_Interval *(macro)* 96
Spike_PSTH *(macro)* 97
Spike_Rate *(macro)* 94
Spike2Events *(macro)* 100
Spike2Triggers *(macro)* 100
spikesfind *(formula_macro)* 86
Step *(macro)* 63
Stim *(protocol)* 38
Stimulator *(protocol)* 37
Stimulus *(protocol)* 37
Stop *(macro)* 63
StopStim *(protocol)* 42
Subtract *(macro)* 67
SWDdetect *(protocol)* 46
Temperature *(macro)* 64
Tetanus *(protocol)* 39
Time *(protocol)* 44
Transfer *(macro)* 68
trigger *(formula_macro)* 84
TriggerOff *(macro)* 62
TriggerOn *(macro)* 62
Twolines *(formula_macro)* 86
Upscale *(macro)* 67
VCProtocol *(protocol)* 28
VerboseOff *(macro)* 61
VerboseOn *(macro)* 61
Wait *(superset)* 114
Wave *(protocol)* 41
Wormpanel16 *(protocol)* 50, 52
Wormpanel32 *(protocol)* 50, 52
WriteModus *(protocol)* 45
WriteOff *(protocol)* 45
WriteOn *(protocol)* 45
XY *(macro)* 74
Zero *(macro)* 70
ZoomOff *(macro)* 61
ZoomOn *(macro)* 61

Wishlist to be incorporated in NEURON

~~-- Start with external trigger (Natalie)~~

~~-- EEG collection epilepsy set-up.~~

~~-- Make database from multiple event files (with link to original datafiles!)~~

-- Reintroduce reading of EDF file

-- Synchronize camera with EEG

-- Printing/Plotting?

-- Fast sampling of multiple channels in VSD set-up

-- Drive stepper motors (position) in Scientifica set-up.

-- Compiler uittesten

Todo list for Neuron:

Documentation

~~Trace / Sweep mode~~
~~Tooltips completeren~~

Meten

~~Catch EPs during EEG based on barcode (write with lower resolution?)~~
~~Online rate analysis (catch events) for Taco (n channels)~~
~~Test oude ADCs op ADC/DAC samenwerking~~
~~Continuous stimulation during EEG~~
~~Poisson stimulation during EEG~~
~~Gent stimulatie implementeren~~
~~Gradual increase of stimulation intensity~~

Display

~~Maak modifiers in trace en sweep mode gelijk (waar mogelijk: markers, gain)!~~
~~Windows opnieuw definieren!~~
~~Uitlezen data met muis~~
~~——Absolute~~
~~——Control key to go relative~~

Analyse

~~Completeer Excel sheet schrijven~~
~~——fitting~~
~~——spike analyse~~
~~Post processing Mini's~~
~~Classificatie EPs etc~~
~~Complete PCA analysis~~
~~Incorporate Frequency analysis / spectrogram~~

Exporting data

~~Export data in various formats~~
~~——trace mode / sweep mode~~
~~——txtfile for Kaleidograph (EXCEL)~~