NEURON

Acquiring & Analyzing Neurophysiological Data

Wytse Wadman

2011-11-15

Content:

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

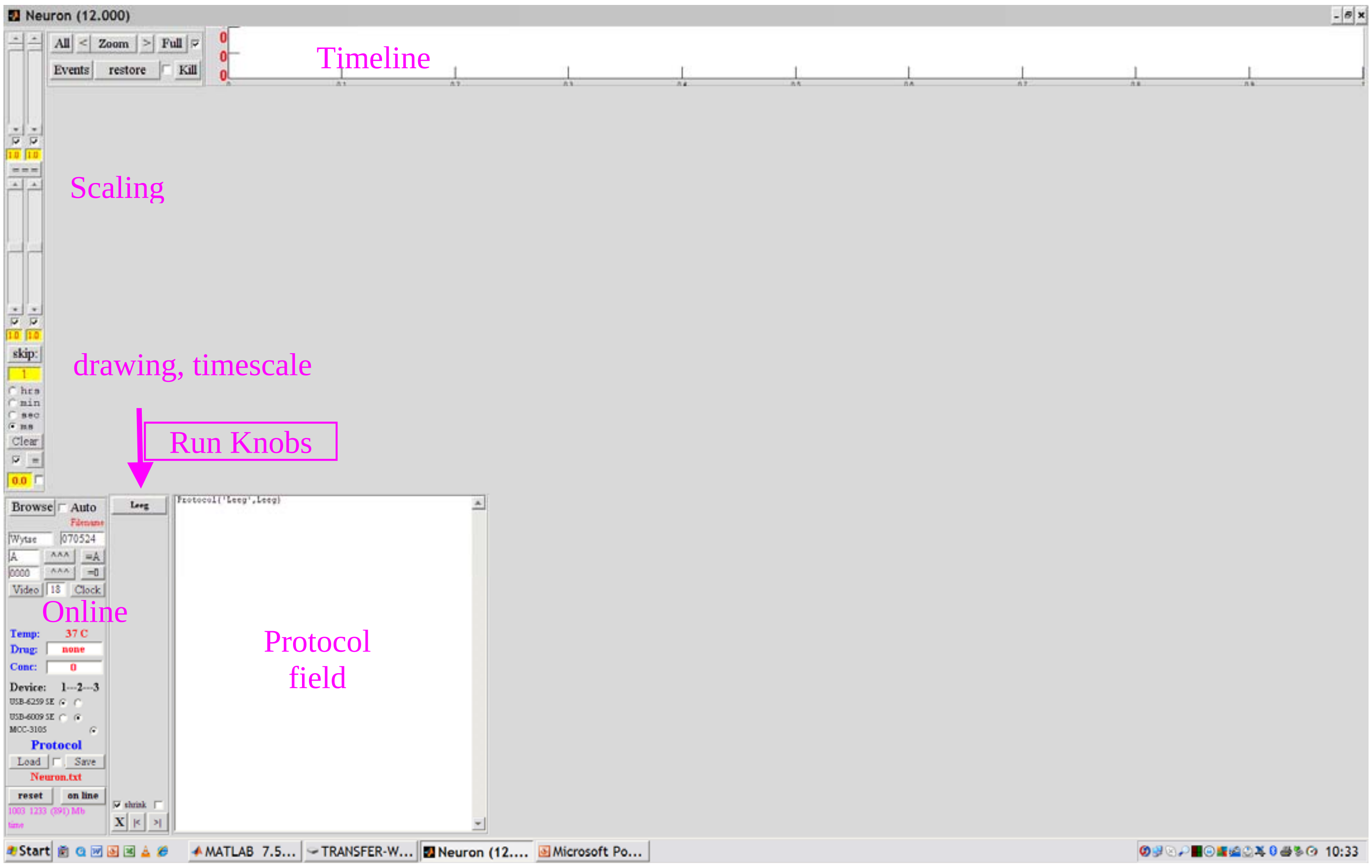
3) Planning and running an experiment, followed by a detailed list of commands needed to define a protocol
4) Examples of how to run specific experiments

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

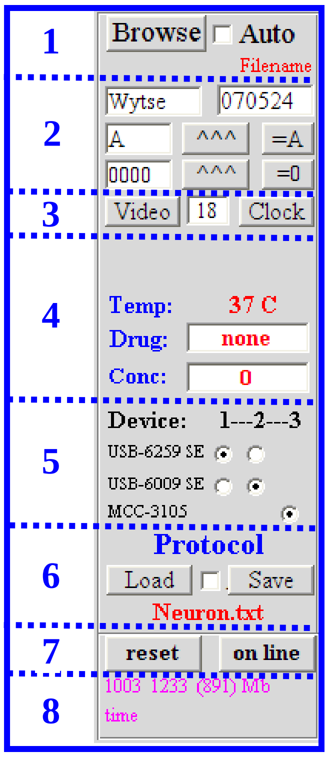
There may be a backlog in updating the description of the different programming parts, but the description of the various commands is mostly kept up to date and can always be used for reference.

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).

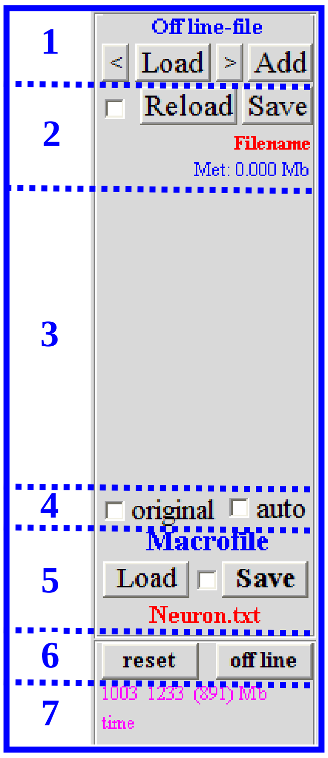
--- section 5
The next section on the way to the top of the panel is the hardware section, which explains you which ADC devices the program has detected. At the moment you can work with three different ADC devices simultaneously. Their names show up at the left, you can determine here, which one will 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

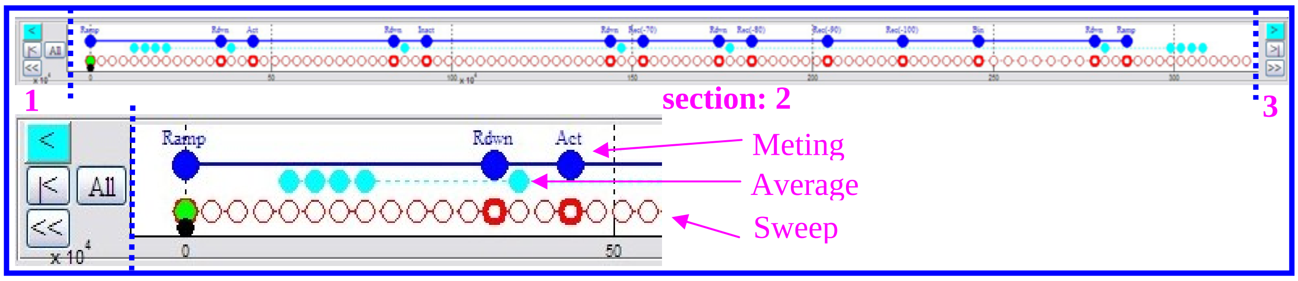


With the online/offline knob in section 6 you can switch to Offline mode, which changes the appearance of this panel

--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).

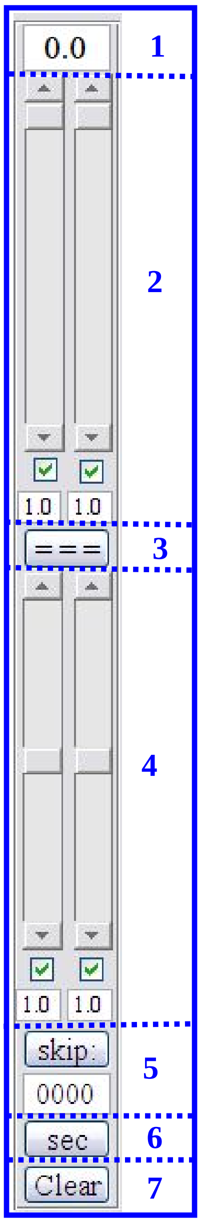
Now there are two markers more
-- *original*
when this marker is on, an offline file when loaded will be displayed in the same way as when it was measured. The advantage of this mode is that it will almost always work, while in other situations the display could easily crash if it is not at all suited 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

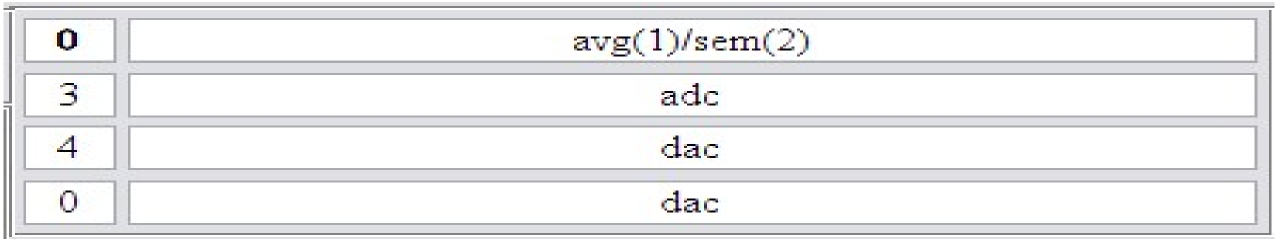


-- left mouse pointer: show this Meting (= all sweeps)
-- right mouse pointer: show this Meting (= all sweeps) in original mode.
--- section 1

--- section 3

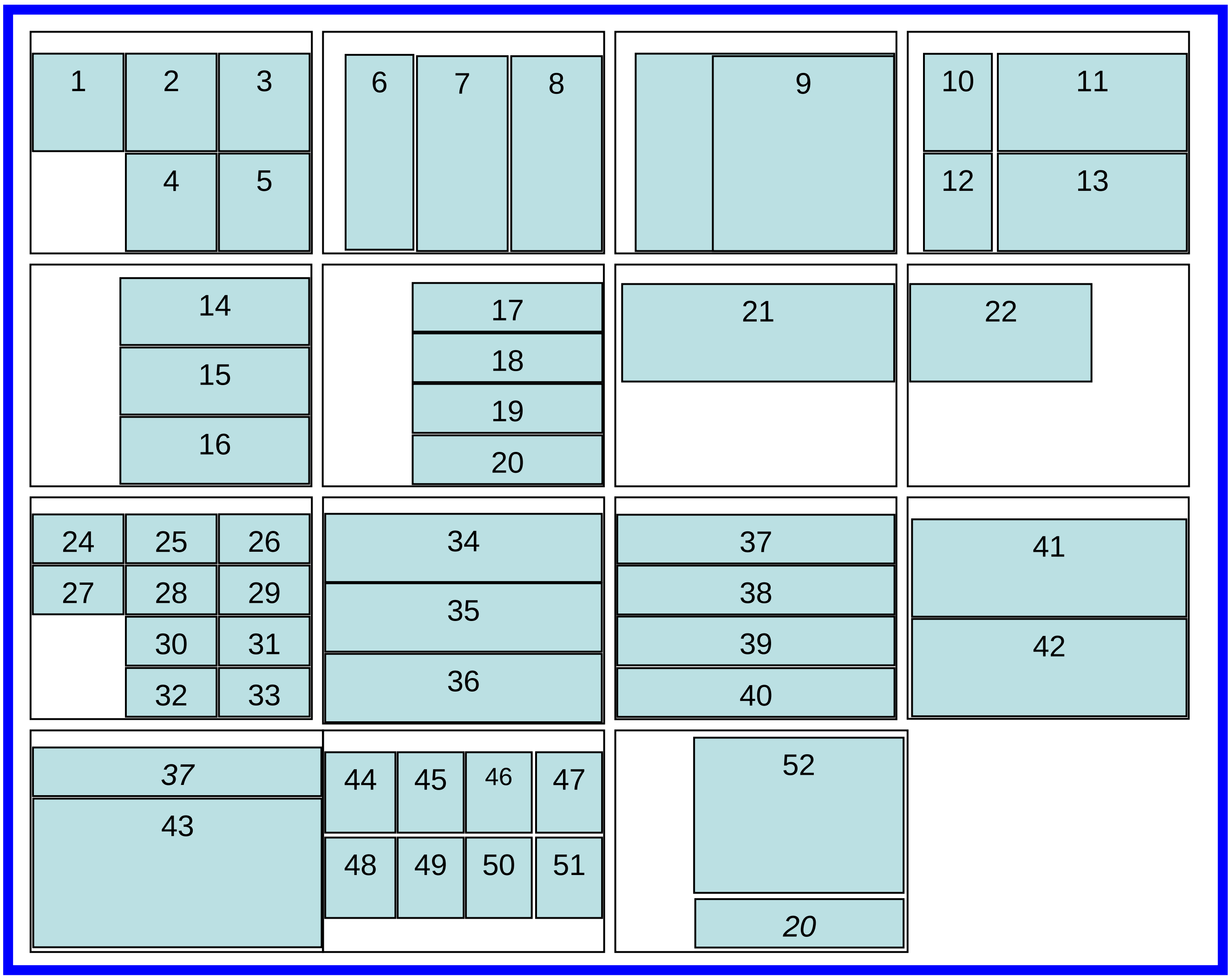
Gain and Stack sliders (work on all windows at the same time (except XY):
-- section 1
Topslider will define how to spread the traces over the plotframe (0 = superimposed, 1= maximal spread, non overlap, use a number as alternative, mark sets exactly to 1.
-- section 2
Lower sliders: amplitude gain of 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 setting of the window.
'sec'	
'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