mafPC, mafScope, and mafDC
Instructions and Tutorial

Matthew Xu-Friedman
University at Buffalo

Contents

Last updated: 10/30/2019

General Comments

This document describes a bunch of useful routines to carry out electrophysiology experiments, and you could use it really for anything involving A/D and D/A systems. There are not a few of these packages out there. This package is in use in our own lab, as well as a bunch of others around the world, on a daily basis for doing whole-cell patch clamp experiments.

This package was written with generality in mind, to make it as powerful as possible. The interface is implemented entirely in Igor-native code, so it runs on Mac and PC. It was also written to work with both Instrutech and National Instruments boards, and can conceivably be adapted for any data acquisition system provided adequate Igor drivers. The main parts are:

mafScope: an oscilloscope window, primarily for patching

mafPC: a tool for designing and executing voltage or current protocols of arbitrary complexity

mafBrowse: a tool for reviewing waveform data collected by mafPC

mafITC: a set of utilities for interacting with the data acquisition system

mafCam: a tool for interacting with cameras via Bruxton's driver

mafDC: a tool for driving dynamic clamp experiments

The generality of the package makes it possible to conduct nearly any experiment you can imagine. However, non-programmers may have a harder time getting started. It may be advisable for an experienced Igor programmer to write macros for the non-programmers in the lab to simply hit a button to run their experiments. There are some example scripts down in the "Recipes" section to get used to controlling mafPC.

While this software is mostly reliable, it is not foolproof. If you try to do something bad, it may not stop you. In addition, there could be unknown bugs, so it is up to you to take precautions to verify your data are correct, and that you protect your experimental efforts. If you find bugs, please report them to Matthew Xu-Friedman (mx@buffalo.edu). The legal department here would probably want me to say you shouldn't use this program for anything involving human health, in case that wasn't obvious. We encourage you to suggest new features by contacting us.

If you publish a paper using mafPC, please acknowledge using it somewhere in your methods (e.g. "Electrophysiology data were collected using Igor (Wavemetrics) running mafPC (courtesy of M. A. Xu-Friedman)", or similar).

Compatibility

Below are system configurations that are known to work successfully.

System: Mac OS 9, Windows XP, Windows 7, Windows 10

Boards: Instrutech ITC16, ITC18, National Instruments E-series boards (e.g. 6040e), M-series boards (e.g. PCI-6221, -6229) or X-series boards (e.g. PCI-6361)

Amplifiers: Anything, but interactions are all via command voltage or TTLs

Igor version: Igor 7. Igor 8 will probably object to the mafDC XOP.

Preferences

In Windows, your preferences and settings files are typically found inside the Wavemetrics folder in your Documents folder. If your IT department does not grant you permission to use that, you will get an error message when you try to start mafScope or mafPC. In that case, go to

Misc→Miscellaneous Settings…→Igor User Files tab→Change Path… button to direct to another disk location.

mafPC

mafPC is based on Pulse Control by Herrington & Bookman (1995). The primary changes from that program are the generalization to controlling both Instrutech and National Instruments board, and also that it is written in native Igor code, so it can run on both Mac and PC platforms.

0. Installation

To use mafPC, you must first install the correct drivers for your hardware.

- For the ITC board, download appropriate drivers from the Instrutech/Heka website. There are bewilderingly many options. You can use the legacy XOP that is specifically listed for your device, if it is still there. For the PC, install the hardware drivers first. Then for both Mac and PC, put the Igor extension file (called "ITC18_X86_V71" or similar) into the "Igor Extensions" folder in your Documents area. You can also use the Igor Pro LIH XOP, which we have used in Igors 7 and 8, which covers multiple kinds of boards on the PC, but it cannot implement dynamic clamp, as far as I can tell.
- For the National Instruments board, install the board drivers using the CD that came with the board. Then, buy and install the NIDAQmx Tools from Wavemetrics. Note that the NI boards work with the PC only, as the drivers for the Mac are too primitive.

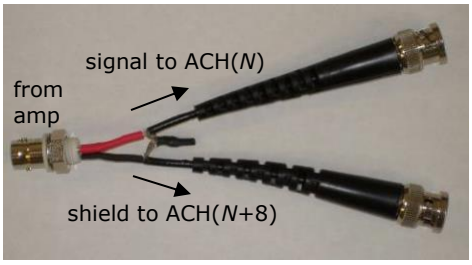
Once the board is installed properly, place the following files into your "User Procedures" folder in the Wavemetrics part of your Documents folder:

mafITC mafutils mafPC3

Then run Igor, and add the following line to your experiment's Procedure Window:

#include "mafPC3"

To make sure the system is initialized properly, it is currently necessary to "Show ITC AD Settings" in the "maf" menu. If you are also using mafScope, then opening the scope window is sufficient to initialize the system. If you are upgrading from an earlier version, you may get an error when you initialize because some preferences files are not compatible. Make sure to resave your settings from the AD Settings and Scope windows. Also, if you pre-save Igor experiments to use as templates, beware that mafPC may not be properly initialized after an upgrade, so try to start from a fresh template.



Our experience is that noise characteristics of National Instruments boards are quite different between "differential" and "single-ended" modes. To use differential mode with the BNC-2090 breakout box, it is necessary to make a simple adapter that splits a BNC cable into two outputs (see picture at right). The center wire goes to the channel (e.g. 0), and the shield goes to channel *N*+8 below it (e.g. 8). If you do this, be sure to set the switch on the BNC-2090, and click the "Diff" radio button in "Show ITC AD Settings" in the "maf" menu. This may not honestly be worth the trouble. Furthermore, we have found that the behavior of the USB versions of NI boards have unpredictable noise in the different modes. The bottom line is you may need to find the best configuration empirically.

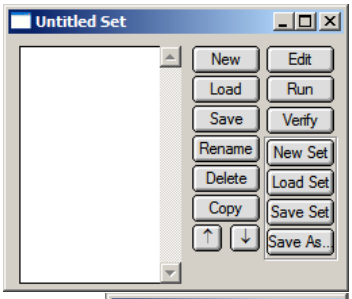
1. General Organization

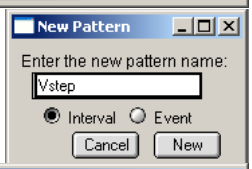
Patterns are created by the user and organized into sets. Each pattern specifies which DA and TTL channels it controls, and sets the timing and values of each DA and TTL channel. There are two kinds of patterns, interval-based and event-based.

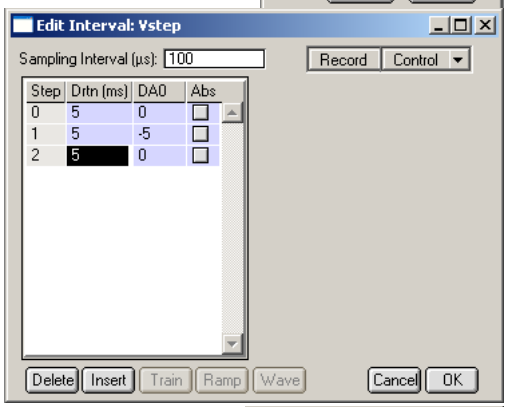
2. Interval-based Patterns

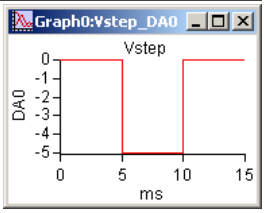
These patterns specify how DA and TTL channels act over a series of steps.

Example: Simple voltage step







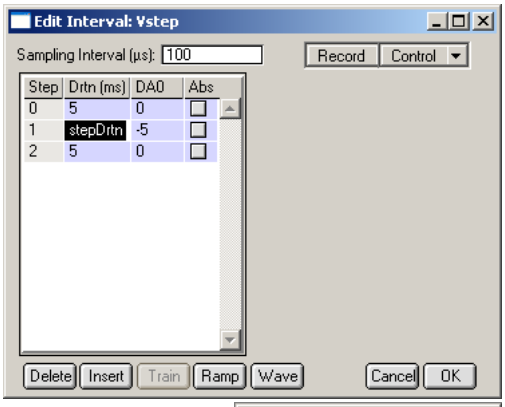


To make this pattern:

1. Choose “New Set” from the “maf” menu (maf→New Set)
2. In the “Untitled Set” panel, click the “New” button (at right)
3. Name the pattern “Vstep”, make it an Interval pattern, and hit “New” (see diagram below right)
4. In the Edit Interval Panel, insert 3 rows
5. Select to record AD0, and control DA0
6. Choose a sampling rate of 100 µs, and enter the information as in the diagram below right
7. Select “OK”
8. Now choose “Verify Pattern” from the “maf” menu. The Verify window shows you what DA and TTL channels will be set to when the pulse pattern is run.
9. If it gives you an error, you probably didn’t initialize mafITC. Make sure you do that by opening the mafScope window, or by showing the ITC AD settings.
10. If all went well, it should look like the diagram below right. We will be looking at patterns using the Verify feature, rather than running them, but the rule of thumb is that if it verifies properly, it will run properly. I will address running patterns further down.
11. To get a sense of what the columns in the edit panel mean, select “Vstep” and click the “Edit” button. The “Drtn (ms)” column controls the duration of that step, and the “DA0” column controls the voltage (for patterns run in voltage clamp) or current (for patterns run in current clamp). The “Abs” column determines if the voltage specified is absolute or relative to the current holding potential/current. Fiddle around with these to see how they change the wave in the Verify window.

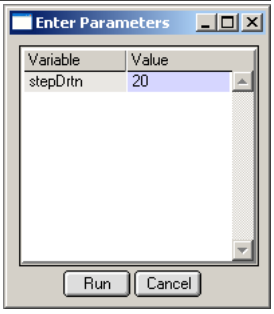
3. Using Parameters

Parameters allow you to control times and amplitudes when a pattern is run, without requiring you to edit the pattern by hand. Parameters are specified by their names. Everywhere you can put a number you can put a parameter (and vice versa). Parameters are not to be confused with Igor variables. The parameter names you use are not tied to Igor variable values in any way. Parameter values are stored so that they needn't be specified the next time the pattern is run.



Example: Simple voltage step

1. Select "Vstep" from the untitled set
2. Click the "Edit" button
3. Change the duration in step 1 to "stepDrtn", as in the diagram at right
4. Hit OK
5. Verify the pattern with the "Verify" button
6. Now you are presented with the "Enter Parameters" panel so you can set a value for stepDrtn
7. Enter a value, as in the diagram at right
8. Click "Verify" button, and watch what happens in the Verify window.
9. Verify it again, and enter different values, and see what happens in the window.



Warning: Some names should not be used for parameters, and may cause a pulse pattern to malfunction. They are any of the following words by themselves or followed by a number:

ABS	DEPENDENCY	REP
AMP	DYNAMICCLAMP	SAMPINT
CH	FINISH	SAVECHAN
COMPILED	NUMROWS	TIME
COMPILETIME	PADFINISH	TTL
CONTROL	PATTERN	WAVE
DCMINI	PATTYPE	WAVEDEPENDENCY
DCPULSE	RAMPEND	
DCVREV	RAMPSTART	

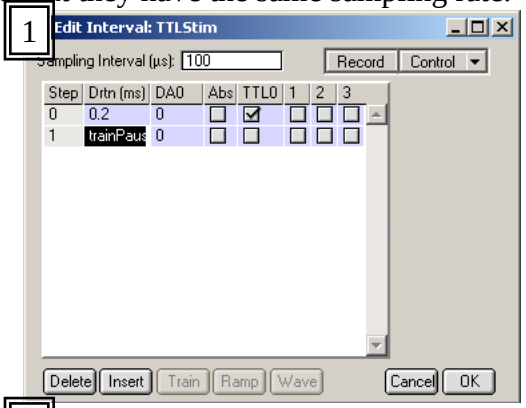
4. Using Trains

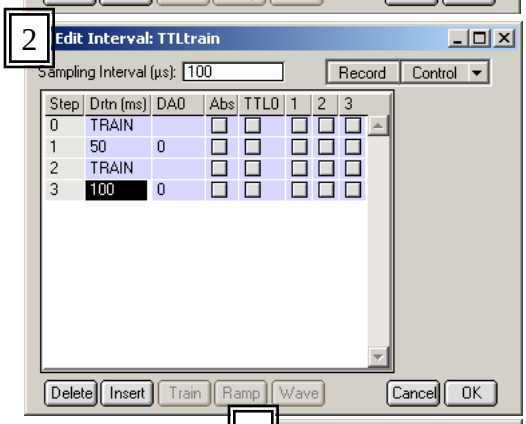
Trains allow you to insert patterns ("subpatterns") within other patterns, which provides a lot of flexibility. Subpatterns are run and compiled without your having to worry about them separately. They may share the same parameters, in which case one value applies to all subpatterns. For interval-based patterns, it is important that patterns and their subpatterns all control the same outputs (i.e. DA channels and TTL), and that they have the same sampling rate.

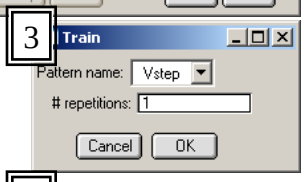
Example: TTL stimulus pulse

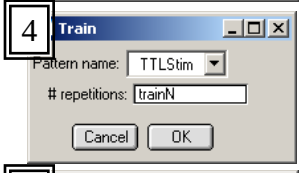
We want to design a pattern that gives us a short hyperpolarizing pulse (to check our series resistance), followed by a pause, followed by a train of TTL pulses, followed by a short tail.

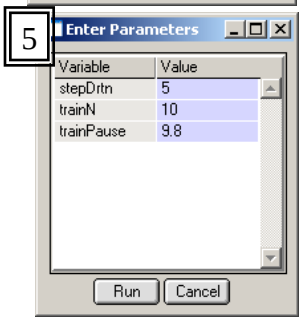
1. Select "Vstep" from the untitled set
2. Edit it, and change the channels controlled to also include TTL
3. Click OK
4. Create a new interval pattern called "TTLStim" (diagram 1)
5. Set it up in the Edit panel according to the diagram at right. The parameter name in step 1 should be entered as "trainPause".
6. Click OK
7. Create a new interval pattern called "TTLTrain" (diagram 2)
8. Insert the 1st line
9. Click in the "Drtn (ms)" field to select it
10. Click the "Train" button
11. This brings up the Train panel, which allows you to specify the pattern to repeat, and the number of repetitions. Set it according to diagram 3.
12. Click OK
13. If you want to double check the train settings, just click the "Train" button again, and it will show you what you entered
14. Insert a 2nd line into TTLTrain, and set it to be a pause of 50 ms, where nothing happens
15. Insert a 3rd line into TTLTrain, click in the duration field, and click the train button
16. Set the Train panel according to diagram 4. Note that we are using a parameter for the number of repetitions.
17. Click OK
18. Insert a 4th line, and set it to be a pause of 100 ms, where nothing happens
19. The TTL train pattern should now match diagram 2
20. Click OK
21. Verify the pattern
22. Enter the parameters according to diagram 5



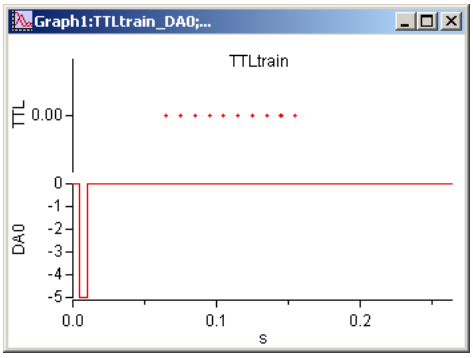








23. Bring up the Verify window for this TTLtrain. It should look like the diagram at right. We expect to get a prepulse of –5 mV for 5 ms, followed by a train of 10 TTL stimuli separated by 10 ms each (0.2 stim + 9.8 pause).
24. Verify it again and enter different values, and see what happens in the Verify window.

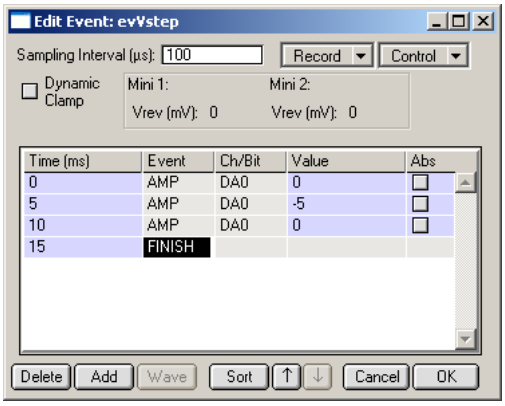


Note that we used trains in two different ways in this pattern. We used one to generate a genuine TTL train, but for the Vstep pattern, we only had 1 repetition. This highlights the strategy of creating a simple building block pattern to do a specific thing, and then plugging it into a more complex pattern to actually execute.

5. Event-based Patterns

In the interval-based patterns above, you specify what happens for all controlled outputs at every step, where you control the duration of each step. By contrast, in event-based patterns, you specify the absolute time that an event occurs, where an event can be a change in a holding potential/current (an "AMP" event), or a TTL going high or low (a "TTL" event), etc. This method can do everything interval-based patterns can do and more.

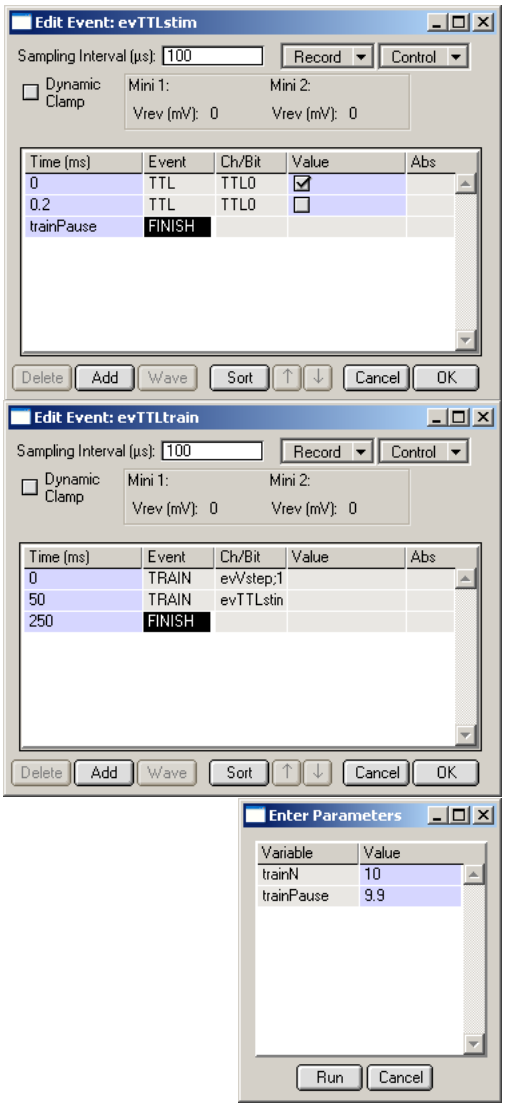
Example: Simple voltage step



1. Create event-based pattern "evVstep"
2. Set Control to DA0
3. Add 4 lines, and set them according to the diagram at right
4. The "Time (ms)" column determines at what time an event takes place. The "Event" column determines what type of event it was, which depends on which channels are being controlled. The "Ch/Bit" column determines which DA channel or TTL bit is undergoing the event. The "Value" column determines the holding potential or current for DA channels. For TTL bits, "Value" has a checkbox, which determines if the TTL line is high (checked) or low (unchecked). The "Abs" column determines whether the amplitude is absolute, or relative to the current holding potential. All event patterns must end with a "Finish" time (or "Padfinish" – see below).
5. Verify your pattern
6. It should look just like the output for "Vstep" that we did above

Example: TTL stimulus train

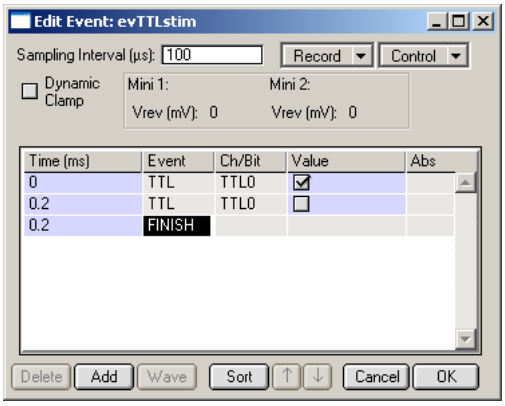
1. Create a new event-based pattern “evTTLstim”
2. Set Control to TTL
3. Add 3 lines and set them according to the diagram at right
4. Create a new event-based pattern “evTTLtrain”
5. Set Control to TTL and DA0
6. Add 3 lines and set them according to the diagram below right
7. The first event is a train of pattern “evVstep” for 1 repetition, and the second event is a train of pattern “evTTLstim” with the number of repetitions set to the parameter “trainN”
8. Now verify and enter parameters according to the diagram below right.
9. Verify the pattern. It should resemble the output for interval pattern “TTLtrain” above. Note that to get 10 ms spacing between TTL pulses, trainPause must be 10 – 0.1 ms (i.e. the sampling interval).

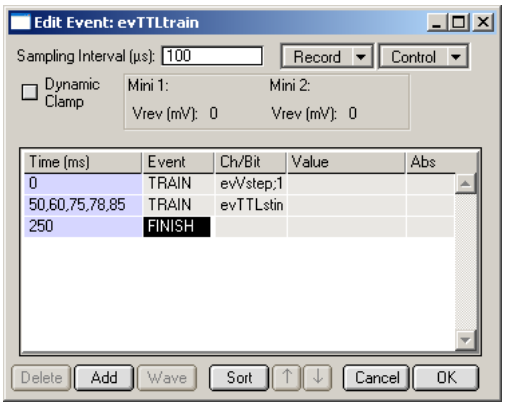


6. Time Lists

So far, we have explored the features of interval- and event-based patterns that are equivalent, but event-based patterns allow another feature for specifying the times and amplitudes of events, using lists. In this case, the given event occurs at each time specified. This allows some powerful behaviors that interval-based patterns cannot easily accommodate.

Example: Irregular TTL train





1. First, edit "evTTLstim" to be very short, according to the diagram at right
2. Now edit "evTTLtrain", and modify the line that specifies the TTL train according to the diagram below right. Make sure to change the number of repetitions for the "evTTLstim" train to 1. Notice that the "Time" on step 2 is now a list of times. Each value in this list must be separated by commas.
3. Verify the pattern. It will be an irregular pulse train.
4. Edit "evTTLtrain" again, and change the time list to parameter "TTLtimes"
5. Verify again. For the parameter "TTLtimes", give whatever values you want, and watch the effect in the Verify window.

This behavior is very useful for generating both regular and irregular stimulus trains, which are difficult to do any other way.

If our event had been an AMP, rather than a train, we could have specified the DA values using lists as well. If there is only one item in the list, then it is duplicated at each time specified. If the number of items in the value list matches the number of items in the time list, then each time is assigned its corresponding DA value.

Note that for a TRAIN event, the number of repetitions of the train is independent of the length of the time list. Usually with a time list, TRAIN events will have the number of repetitions set to 1. If the number of TRAIN repetitions were greater (say, 6), you would get 6 repetitions of the subpattern at each time specified in the list. This might be useful for generating multiple bursts of activity.

Another thing you can do with time lists in event-based patterns is nothing at all. If you set the time as nothing (i.e. blank) or not-a-number (NaN), then it won't execute that line. This may seem counterintuitive, but it is useful in experiments where you vary the time of, say, a sound or light pulse using a parameter, and sometimes you may want the pulse omitted. In that case, pass an empty time as a parameter. That way, you don't need a whole separate pattern that omits the pulse.

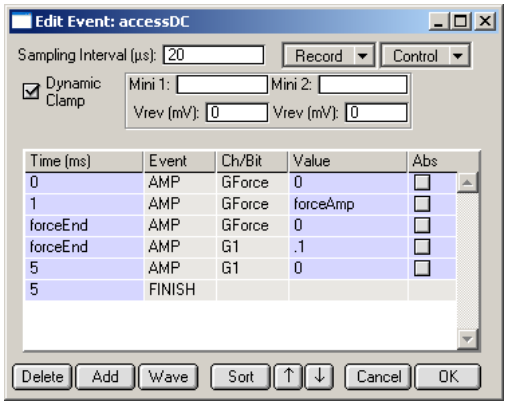
7. Dynamic Clamp

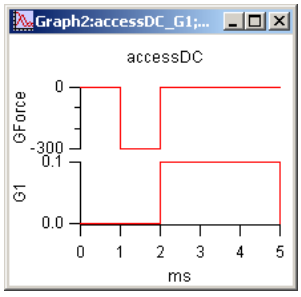
Dynamic clamp is a feature of the ITC-18, where the user can specify up to 2 linear conductances, as well as a "forcing" conductance. mafPC assists in gaining access to this feature, using event-based patterns. These are some critical elements to get started.

1. Only the legacy drivers provide access to the dynamic clamp capabilities of the ITC18. The new LIH 8+8 drivers do not. The most recent system we have confirmed the ability to use legacy XOPs is Igor 6 running in Windows 7. I am glad to hear if you are able to use a different configuration.
2. The Instrutech interface requires that DA0 be used to control the current to be passed into the cell, and AD0 records the membrane potential.
3. DA1, DA2, and the TTL outputs can theoretically also be used, however we find that only DA1 works properly.
4. The G1 and G2 conductances are assumed by the ITC-18 to be in nS.
5. The "forcing" conductance is used to inject a constant current. To use this, enter a value for Forcing Gain in the AD Parameters dialog box (choose "Show ITC AD Settings" from the "maf" menu). We determined our value of 0.0009 V empirically.
6. Conductances > 10 nS are not handled well by the ITC interface. This can be extended further by placing an amplifier between DA0 and the input to the current clamp amplifier. Be sure to set the External Gain value in the AD Parameters dialog box.
7. The ITC-18 must be explicitly placed in dynamic clamp mode to run these patterns, using the Instrutech extension **ITC18LoadDynamicClampMode** (to enter dynamic clamp) and **ITC18Reset** (to leave dynamic clamp), or **mafITC_SetDynamicClamp** (see below).

Example: Forcing conductance

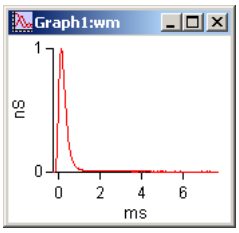
1. Create a new event-based pattern, named "accessDC"
2. Check the "Dynamic clamp" checkbox in the Edit panel
3. Add six lines and edit them according to the diagram at right.
4. Verify the pattern, and enter forceEnd = 2 and forceAmp = -300.
5. This pattern delivers a –300 pA current to the cell. The small G1 conductance is used to avoid a rounding error in the ITC interface.



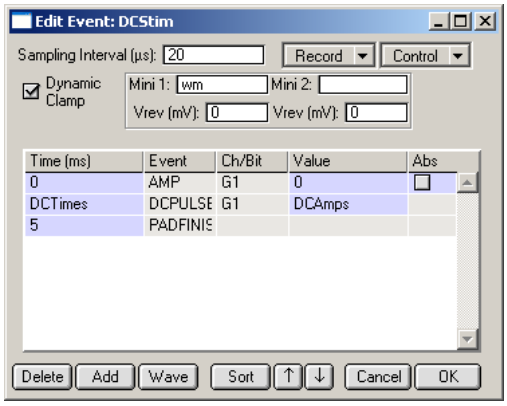


Example: Synaptic conductance

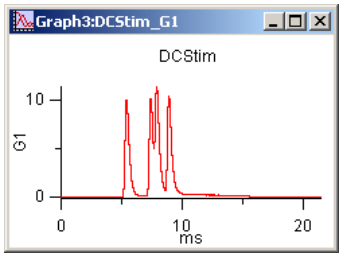
1. Extract a single EPSC from a voltage clamp experiment, scale it to +1, and call it “wm” (see example at right).
2. Create a new event-based pattern, and name it “DCStim”
3. Edit it according to the diagram below right.
4. Verify the pattern. Enter the value “10” for DCAmps, and “5,7,7.5,8.5“ for DCTimes.



Note that the synaptic conductances summate when they overlap. The synaptic conductances are scaled by multiplying the mini wave according to the value specified for the DCPULSE. With time lists, the value may also be a list, in which case, each synaptic conductance can have a different peak amplitude. Synaptic conductances can also be mimicked using an AMP event for a conductance, with a wave for the value.



This ITC18 implementation of dynamic clamp has some limitations. Most notably it can only do linear synaptic-like conductances, whose timecourse is specified ahead of time. The mafDC XOP described below has more functionality. mafDC does not make use of this G1, G2 business, but rather acts through the DA outputs just like any other pattern. Therefore, the specific dynamic clamp functions here are not used, although the CONVOLVE event, which acts like DCPULSE, may be useful.



8. Loading and Saving

Patterns are organized into sets, and sets and patterns can all be saved to disk for use by other experiments. Igor provides a folder to store things like this. In Windows 7, the folder is inside the Wavemetrics folder in Documents called "mafPC" (C:\Users*your name here*\Documents\ WaveMetrics\Igor Pro 6 User Files\mafPC). The Mac has it in an analogous place. This can raise some issues you should be aware of. Sometimes, the IT department doesn't let you touch this directory. If that is the case, you will need to specify an alternative. The other problem is that if you have multiple users with distinct log-ins, then they all have their own individual User Files area, which makes it much harder to share pulse patterns and procedure files. To resolve both these problems, Igor lets you set a different User Files area. To do that, go to the "Miscellaneous Settings…" in the "Misc" menu. As you can see by looking at the "Untitled Set" panel, patterns can be individually loaded, saved, copied, renamed, etc. You can also save an entire set at a time by using the "Save Set" button. Try saving your set as "TestSet", then creating a new set to clear out memory, and then loading in your old set. Edit the patterns to convince yourself they are still there.

There are two important points. First, when you load a pattern into an experiment, all the operations take place on that pattern, and do not affect the one on disk until it is saved explicitly. There is no reminder to save your patterns when you close an experiment. The experiment continues to hold its patterns, so if you need to check what patterns you used to use, or if you forgot to save them, just reopen the experiment.

Second, all your old patterns and sets will remain in the mafPC directory on your hard drive until they are moved somewhere. You can use the file managers in your operating system (i.e. the Mac Finder or Windows Explorer) to backup or clear out clutter.

9. Other Features

There are many features that will only be touched on here. Please experiment with them to find out how they work. They are mostly self-explanatory.

Ramps: Allow a voltage or current ramp. In interval-based patterns, ramps are specified by clicking in a DA amplitude field and then clicking the "Ramp" button. Start and end values are entered in the Ramp panel. In event-based patterns, ramps are specified by using RAMPSTART and RAMPEND events. Each RAMPSTART must be matched by a later RAMPEND.

Waves: Allow an arbitrary wave to be sent out a DA channel. In interval-based patterns, first click in the DA amplitude field, and then click the "WAVE" button to enter the wave's name. The scaling of the wave is ignored. If a wave has too many points for the time allocated, it will be truncated. Waves that are too short fill out their time with the last value.

In event-based patterns, waves are specified by clicking in the DA value field, and then clicking the "WAVE" button to enter the wave's name. You can also just type "WAVE:waveName" into the value field. The scaling of the original wave is ignored. Waves are copied in, and the last point in the wave is used to set the DA output for the rest of the pattern. Waves that are too long are truncated, so make sure the pattern is long enough to accommodate it.

PADFINISH: Event-based patterns require an event to end them. The FINISH event uses an absolute time. This is a problem if you use trains or waves, and may not know how much time to allow. In this situation, use a PADFINISH event. The time value here is added to the last event time specified, so that data may be collected before ending. In practice, I rarely use "FINISH", only "PADFINISH".

CONVOLVE: The CONVOLVE event allows you to send out a more complex output. You specify an Igor wave that is to be sent out at the specified event times. This is somewhat more complex than AMP events, which are basically square pulses. On the other hand, they are simpler than specifying a whole wave ahead of time to send out. You could use this to build complex stimuli, or as part of a dynamic clamp interface that interacts with a slave computer.

Sort, and arrows: Allow an event-based pattern to be organized temporally. The ordering of lines is ignored when an event-based pattern is compiled, however.

Subpatterns in interval- vs. event-based patterns: For interval-based patterns, every subpattern must control the same outputs as the parent pattern. This is not true for event-based patterns. There, a subpattern may control only the TTL or the DA. DA subpatterns that overlap in time supersede each other, whereas TTL patterns are OR'd together. Use the Verify window to check that these work the way you intend.

10. Programming with mafPC

All the functions you have been doing with the user interface are also accessible from Igor macros and functions. Of course, usually, you won't just be verifying, but rather actually running your patterns. Here are the important commands you will find useful.

mafPC_OpenSet (setname)

This allows you to automatically load a set at the beginning of an experiment.

mafPC_Run ("PatternName", ParameterList)

Runs a pattern after making sure it is compiled properly.

Note that the name of the pattern must be enclosed in quotation marks. Channels to record are determined by the pattern, and saved according the settings in the AD parameters window (see **mafITC**). The ParameterList allows you to pass the parameters for a pattern and its subpatterns. This list is formatted like Igor's keyword-list syntax, i.e.

"parameter1:value1;parameter2:value2;timelist:time1,time2,time3"

Note that parameter names are separated from their values by colons (:), parameters are separated from each other by semi-colons (;), and time list values are separated from each other by commas (,). These parameters can be joined together in one big statement, or built up using Igor commands, such as **ReplaceNumberByKey**. Ordering doesn't matter for parameter lists, and all parameters are passed to each train subpattern. Parameters are stored by a pattern when it is compiled, so on subsequent compilations, they need not be respecified until they change or the pattern is editted.

Examples:

```
mafPC_Run ("TTLtrain", "stepDrtn:5;trainPause:9.8;trainN:10")
mafPC_Run ("Vstep", "stepDrtn:" + num2str (myStepDrtnVariable))
mafPC_Run ("evTTLtrain", "TTLtimes:50,60,75,78,85")
```

FYI: For help in generating time and amplitude lists, mafutils also includes a utility function: **Wave2List** (wave, delimiter) takes a numeric wave and converts it to a delimited string list. So the last example could have been done as:

```
make /n=5 myTimes={50,60,75,78,85}
mafPC_Run ("evTTLtrain", "TTLtimes:" + Wave2List (myTimes, ","))
```

Parameters that are not used by a pattern are ignored, and all the parameters are inserted into the wave note. These two features give you a useful way of embedding additional information into your waves. For example you could pass the parameter "CURRENTDRUGS: NBQX,CPP", and retrieve that information later using Igor's **note** and **stringbykey** commands.

mafPC_RunInBackground ("PatternName", ParameterList)

Only works for NIDAQ users. This is the same as mafPC_Run, but it returns control immediately to the function that called it, rather than waiting for the data to be collected by the data acquisition system. The routine **mafPC_WaitingforBackground ()** tells you if the process is complete. This is necessary if you have other tasks to attend to while data are being collected. For example, Bruxton's SIDX drivers can capture images from a camera like Cooke's Sensicam QE into Igor, but those drivers don't automatically offload images from the camera, so you have to do something similar to the following:

```
mafPC_RunInBackground ("DoCameraTrial", "exposureTimes:100,200,300,400")
do
    // calls would go here that check the camera and download images
while (!allImagesCollected)
do  // dummy loop to wait for A/D data to finish
while (mafPC_WaitingForBackground ())
// all done collecting – now process the data
```

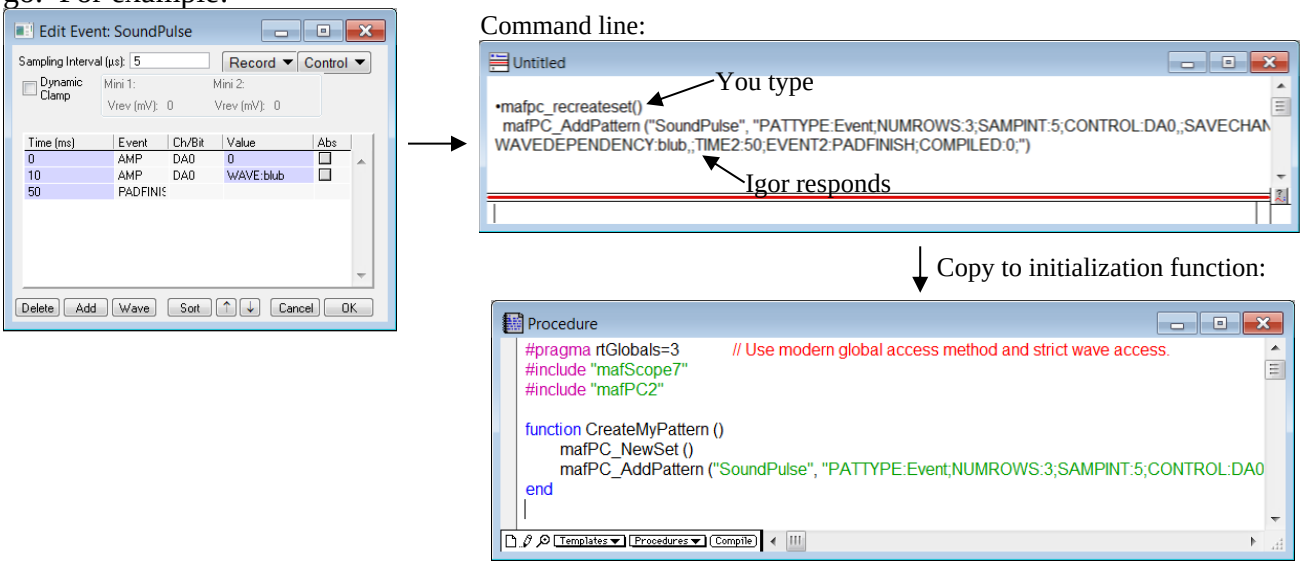
You need to be careful when you use this that you do not try to run another pattern until the previous one has completed. There is a reservation system in place to try to prevent that, but problems can arise if you abort a function before the reservation is cleared. If that happens, you may need to reset the AD. In addition, we have found that sometimes this can cause Igor to hang. Right now, we think this is a bug in the NIDAQmx drivers (it couldn't be my fault, could it). It seems this problem occurs when the scope window is also running. Therefore, it may be best to stop the scope when you call mafPC_RunInBackground.

mafPC_Compile ("PatternName", ParameterList)

Same syntax as for mafPC_Run. All patterns must be compiled before they are run. However, mafPC_Run calls the compile routine automatically to verify that patterns are up-to-date, so it is unlikely that you would ever compile patterns explicitly.

mafPC_RecreateSet ()
mafPC_AddPattern (patname, pattext)

These are useful routines if you don't want to have separate patterns and sets on the hard drive, but want them tied into your favorite procedure files. The idea here is to create a pattern from a text string in an initialization function, rather than loading it off the disk. To do this, first create your patterns the usual way with the user interface of mafPC. Then call "mafPC_RecreateSet ()", to generate the appropriate function calls to mafPC_AddPattern. You can just cut and paste this into your initialization function. Now when you open a new experiment, and run that initialization function, you will create all those patterns, and be ready to go. For example:

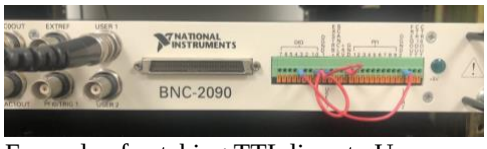


Sometimes patterns can be quite long, and the output of mafPC_RecreateSet () will be split on multiple lines, so you will need to copy and paste carefully, and repair the splits.

11. Some Common Issues

This is a list of things to watch out for, or tricks that you might find useful.

1. **Upgrading Igor.** When you upgrade versions of Igor, such as from 5 to 6, you need to watch out for some problems. mafITC and mafScope store default settings in a special preferences location. When you upgrade versions of Igor, those settings get left behind. To carry them forward, you will need to find them on your hard drive. The simplest way is to search for "mafITC_settings". Depending on the version you have, you may find several important files and folders: "mafITC_settings", "default.txt" that contains your default mafScope settings, plus the "mafPC" folder that contains all your patterns and sets. You can copy these to the folder named for your new Igor version to reduce your setup time.
2. **Upgrading Windows.** Windows 7 disallows things that used to be possible in XP. In particular, Igor is no longer allowed to write anything into its own folder. The practical upshot is that mafPC and mafScope can't save the preferences/settings files or patterns or sets where they used to (away from prying eyes in the Igor folder of the Program Files). Now these have to go in a folder that you have been ignoring in your Documents Folder, entitled Wavemetrics. This is the place to put your patterns and sets. So if you are upgrading to Windows 7 and want to bring over your old patterns and sets, put them here. The downside of this is that each user has a separate area for patterns and settings, so it is more awkward if patterns need to be upgraded among many users. You can avoid that problem by designating a different User Files area (see "Miscellaneous Settings…" under the Misc menu).
3. **Recovering the baseline.** For those that like to use the "Baseline subtract" option (e.g. in voltage clamp experiments), the value that was subtracted is stored with the wave in the wave note. You can retrieve the value by calling NumberByKey ("BASELINE", note(*w*)), where *w* is the wave you just collected.
4. **Non-events.** Suppose you want to write a generalized pattern that has some event (say, a TTL pulse) happen at different times, but on some trials you don't want that event at all. You could do that by having two different patterns, that are otherwise identical. A more tricky solution is to specify the timing of the event using a parameter. For trials where you want the event to occur, you specify the parameter as normal (e.g. "…;doStim:10;…"), but on trials where you want to skip it, pass an empty parameter ("…;doStim:;…"). When you do that, mafPC just leaves the event out. Note that this is not the same as not passing the parameter at all. In that case, mafPC will complain that you didn't specify it, or, worse yet, uses the parameters from previous calls (which are saved) and do it anyway.
5. **Digital outputs on the National Instruments boards.** NI boards provide digital input-output (labelled "DIO" or "P"). (I've been referring to digital channels as TTLs in this documentation.) The NI interface panels such as the BNC2090 do not provide obvious access to the digital lines. You have two options. One is to patch the digital lines you want through to the BNCs labelled User1 or User2, which may be convenient. To do this, connect a jumper from the digital line you want (e.g. DIO0) to the BNC you want (e.g. User1) (see picture). 1 mm pins work well. The second approach is to connect a wire from the DIO line directly to another device. If your experiment requires three DIO lines, then you have to



Example of patching TTL lines to User BNCs. DIO1 is patched to User1, and DIO3 is patched to User2. Also, note that digital ground (DGND) is tied to ground (GND)

figure out an adapter from a 1 mm pin to a BNC. In both cases, it is important to check that the target device and the 2090 share a common ground. Usually BNCs take care of this for you, but NI is unusual in allowing BNC grounds to float, which can cause very strange behavior. We find it helpful to explicitly ground the digital ground outputs on the BNC2090.

mafITC

mafITC is a package of routines for easing interactions with data acquisition boards. It is obviously named for the Instrutech boards, but also allows interaction with National Instruments boards, and theoretically could be modified to interact with any A/D interface that has Igor drivers. Many of its features can be controlled using "Show ITC AD Settings" in the "maf" menu (see diagram), or by using mafScope (see **mafScope**). AD settings that were stored using the "Save" button are automatically loaded the next time mafITC is initialized.

Collected waves:
When mafPC_Run is executed, the pattern specifies which AD channels to record. mafITC looks up what these waves are to be called, depending on the channel's base name and the output counter. So, for example, if AD0 and AD1 are to be recorded, and the base names for these channels are "w" and "ch1_", and the output counter is 37, then wave w37 and ch1_37 are created to hold the responses of these channels. These waves can be automatically saved (only useful for unpacked experiments) and displayed (a target window should be assigned), and details of the data collection can be printed to the Igor history area.

Routines you may find useful:
mafITC_Init (): Loads in the settings for the A/D system
mafITC_ADGain (channel): Reports A/D gain for the specified channel
mafITC_DAGain (channel): Reports D/A gain for the specified channel
mafITC_Units (channel): Returns the units for the specified channel
mafITC_ADisCC (channel), mafITC_DAisCC (channel): Returns 0 if currently set to voltage clamp, or 1 if current clamp
mafITC_SetADClamp (channel, newCC), mafITC_SetDAClamp (channel, newCC): Sets the specified channel to voltage clamp (newCC=0) or current clamp (newCC = 1)
mafITC_SetDynamicClamp (doIt): Sets the ITC-18 to dynamic clamp mode (doIt=1) or regular mode (doIt=0)
mafITC_SetHP (channel, value): Sets the holding potential of the specified DA channel to the new value. If you are in current clamp, it sets the holding current.
mafITC_GetHP (channel): Reports the holding potential of the specified channel. Equivalent to original Pulse Control variable "hp".
mafITC_nextWave (): Reports the wave output counter value. Equivalent to original Pulse Control variable "nextWave".
mafITC_ADPrefix (channel): Reports the prefix used when storing waves collected for the specified channel
mafITC_lastWave (channel): Reports the name of the last wave collected for the specified channel as a string. You can use this to analyze your data as it is collected, using wave indirection. For example, to get the minimum value from the last wave you collected through AD channel 0, you would enter:

```
print wavemin ($mafITC_lastWave (0))
```

mafITC_SetTTL (channel, state): Sets the digital outputs to be high or low, independent of running a pattern.
mafITC_whatAD (), **mafITC_isITC ()**, **mafITC_isITC18 ()**, **mafITC_isNI ()**: Reports on the current platform in use.

mafITC_GetAD (channelWave, valueWave): Returns the current values on the AD channels specified in the channel wave. Only works for the NIDAQ at the moment.

Known missing features:

Digital input is not implemented for the National Instruments boards (though, it is possible in principle).

Only one National Instruments device can be accessed, and it must be named "dev1". Use NI's Measurement & Automation program to verify that you have a working device called "dev1". If you swap boards around, the board that used to work as "dev1" will become disabled, and the new board will be called "dev2". You will need to delete the useless "dev1" and rename "dev2" as "dev1".

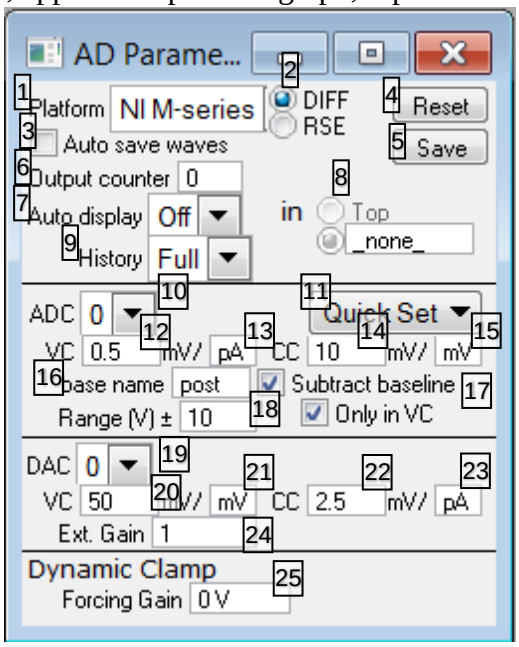
Application Notes

1. mafITC has been used successfully with the ITC18, ITC16, NI6221, NI6229, and NI6361 boards. It has been most thoroughly tested using the PC. The main limitations for the Macintosh are:
    a. NI did not make decent drivers available for the Mac, so the NIDAQmx XOP does not work, and consequently, neither do mafITC, mafScope, and mafPC.
    b. There are subtle differences in the drivers for the ITC18 and ITC16 between the Mac and PC, which are not documented. My recollection is that the calls to ITC16/ITC18StimAndSample inconsistently either have a ",0" at the end or not. You can try adding or deleting these if you are having trouble. (Search for StimAndSample in mafITC and mafScope, and put ",0" at the end of these statements, or delete them, and see if it helps.)
2. **National Instruments PCI 6229**. If you use a 6229, you should make sure you have updated drivers. The old drivers named half of the board "dev1" and the other half "dev2" as a kluge for doubling the number of input and output channels. The new NI drivers make this transparent. You can verify your drivers are current by running the "Measurement and Automation" program in your NI folder and looking at the names for your 6229. If there's only one name, you're good to go. If there's two, you should get the update.
3. **LIH ITC18**. The ITC18 is getting to be an old interface, and the newer XOP from HEKA behaves differently from the old interface. First, it does not implement dynamic clamp anymore. Second, the XOP requires that the board be explicitly activated to work. We find that the board can become inactive when the computer sleeps, or when you start from a template experiment. In that case, it is necessary to hit the Reset button in mafITC AD Settings, at which point, the "Ready" light should turn on, and it will be ready to go.
4. **Setting the gains**. We think as electrophysiologists in terms of pA and mV related to the cell, but the DAQ and the amplifier converse with each other using Volts. So your command currents or potentials that you use in Igor need to be converted to Volts when you control the amplifier. In the Multiclamp Commander, you can see the gains by clicking on the tools icon, and you can enter these (or rather their reciprocals) into mafITC AD Settings. For example, if the command sensitivity in current clamp is "400 pA/V" in the Multiclamp Commander, then you enter 2.5 mV/pA as the DA gain in mafITC AD Settings. In addition, there can be a discrepancy between your command and what the amplifier actually does. For us, we command a holding potential of 10 mV in voltage clamp, but the Multiclamp

Commander reports being at 9.5 mV, and the error gets worse with larger steps. We find that this can be corrected by modifying the DA gain to something like 53 mV/mV. I can't tell if the error is in the DAQ (the voltage sent out) or in the Multiclamp 700B (the voltage read in). If you can tell where the error arises, I'd be happy to know.

ITC AD Settings

1. Data acquisition platform (ITC16/ITC18/NI M-series/NI E-series)
2. Sets if NIDAQ is to be used in Differential or Single-Ended mode (we use Single-Ended)
3. Checkbox to autosave waves after collecting (only useful for unpacked experiments)
4. Reset button, in case A/D system freaks out
5. Save these settings, to be auto-loaded on next startup
6. Output counter, which provides the suffix for collected waves
7. Auto-display of collected waves. Choices may be off, append to specified graph, replace in specified graph (see #8), or mafBrowse (see **mafBrowse**)
8. Radio buttons indicating that auto-displayed wave should be appended to or replaced in the top graph, or in the graph with the name specified
9. Sets the information to be reported in the history area of the experiment. Choices may be off, full (reporting all parameter settings), or minimal (reporting just pattern name and time)
10. Popup of AD channels (inputs) to determine settings
11. Popup to quickly set AD and DA gains for a few amplifiers. Feel free to add your own.
12. Gain for AD channel in voltage clamp
13. Units for AD channel in voltage clamp
14. Gain for AD channel in current clamp
15. Units for AD channel in current clamp
16. Prefix for storing collected waves
17. Checkbox to indicate if baseline should be subtracted after collecting. Lower checkbox specifies that this should only be done in voltage clamp (not current clamp).
18. Range of AD channel, to control on-board gain. Typical values are 1, 2, 5, and 10. Smaller range gives higher resolution.
19. Popup of DA channels (outputs) to determine settings
20. Gain for DA channel in voltage clamp
21. Units for DA stimulus in voltage clamp
22. Gain for DA channel in current clamp
23. Units for DA stimulus in current clamp
24. External gain, for amplifier after DA (useful for ITC18 dynamic clamp[1])
25. Forcing gain, for ITC18 dynamic clamp



[1] **If you don't use dynamic clamp, set this to 1. Otherwise,** this issue is a little complicated. The problem is that the Instrutech board has a math error in dynamic clamp mode such that it can't put out large currents. So you have to put an amplifier between the D/A output and the microelectrode amp, and just ask for a smaller output. This actually works, but then you would rather think in terms of what you actually want, and not the smaller thing you are asking for. So, you set your external gain to 10 or something, and then it is nearly transparent to the user. A new approach to dynamic clamp is described below in **mafDC**.

mafScope

mafScope is an oscilloscope window. It runs in a normal Igor window, and can be left running without adversely affecting other data collection activity. It is very useful for setting the states of different DA and AD channels. All interaction is through the scope window user interface. Note that it does not directly communicate with the actual amplifier, such as to set clamp mode.

Installation

To install, place the following files into your "User Procedures" folder in the Wavemetrics part of your Documents folder:

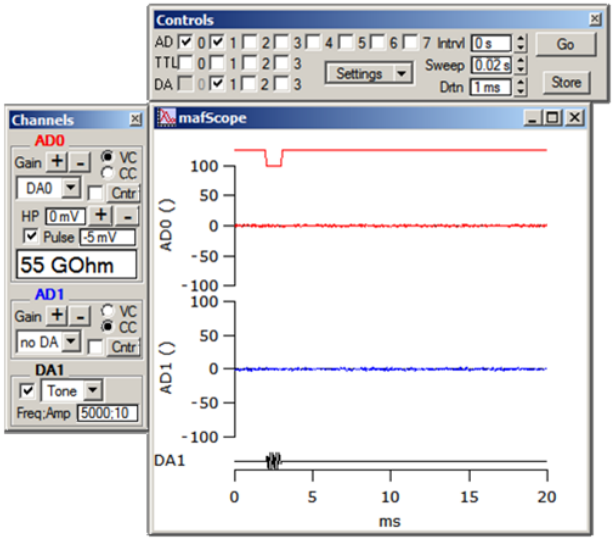
mafutils, mafITC, mafscope7

Then run Igor, and add the following line to your experiment's Procedure Window:

#include "mafscope7"

Then from the "maf" menu, choose "Show Scope Window"

Controls



The Controls Toolbar on the top contains controls that apply to the whole scope window.

ADC Checkboxes: Set which AD channels to display.

TTL Checkboxes: Set which TTL channels to put high during a pulse.

DA Checkboxes: Set which DA channels send out signals (independent of AD)

"Intrvl": Sets the interval between sweeps.

"Sweep": Sets horizontal scale.

"Drtn": Sets pulse duration.

Settings popup: Allows you to show a grid, display FFT of AD waves, set colours of AD traces, and save/retrieve scope configurations.

"Go/Stop" button: Starts/stops the scope.

"Store" button: Saves a copy of the currently displayed waves.

The Channels Toolbar at left contains controls specific to each channel.

Gain buttons: Change trace magnification.

VC/CC radio buttons: Change AD to voltage/current clamp

DA dropdown menu: Links AD channel to a DA channel (for setting holding potential or holding current).

Center buttons: Center the trace once (button) or always (checkbox).

HP/HC field and buttons: Sets holding potential/current.

Pulse checkbox and field: Sets amplitude of current/voltage pulse.

Resistance Display: Shows calculated electrode resistance, based on pulse amplitude.

For DA channels, the DA output can be a tone (specify frequency and amplitude), noise (specify amplitude), or an arbitrary wave (specify wave name).

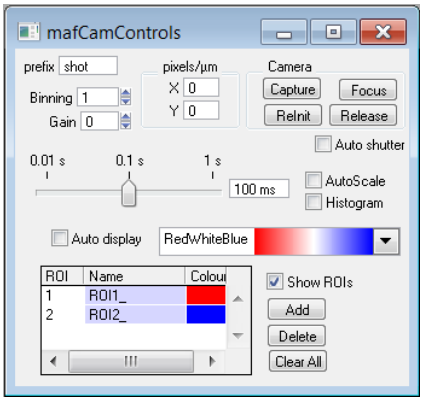
Application Notes

1. To patch onto a cell and do voltage or current clamp, we find it natural to link AD0 with DA0, AD1 with DA1, etc. The DA channels control the holding potential (or holding current) for the cell you record through the AD channels. If you want to do two-electrode dynamic clamp using the ITC-18, then you have to link AD0 with DA1 and DA0 with AD1 because of the system's requirements.
2. mafScope is not smart enough to detect how many AD, DA, or TTL lines there are, so not all the available checkboxes are necessarily usable. In particular, many data acquisition boards from National Instruments have only 2 DA lines, so selecting DA2 or DA3 will cause errors. If you need access to additional lines, let me know, as it is relatively simple to add them. It will just take up more space.
3. To change from voltage clamp to current clamp using a Multiclamp or Axopatch 200, first put the amplifier in "I = 0" mode. Then switch the relevant channel to current clamp on the scope window (making sure the holding and pulse currents are set properly). Then move the amplifier to IC. (You may see bleed-through of scope commands in "I = 0", which is Axon's fault, not mine.) These steps are necessary because the scope only interacts with the amplifier through AD and DA channels. There is an XOP that controls the Multiclamp, but mafITC hasn't had a chance to play with it yet.
4. The goal of the different scope configurations is that you can switch more easily between multiple configurations (e.g. "Patch Configuration" in voltage clamp, "Dynamic Clamp" configuration in current clamp, with different gains, etc.). One configuration has special importance, called "default". You should set up the scope window how you will want it to be when you first start up, and save that configuration as "default". The scope window will start up with that configuration on first opening.
5. The allowable sweep duration depends on the speed of the hardware, and the number of channels being controlled and displayed. Therefore, too brief a sweep duration will cause an error. If this occurs, hit the "Abort Procedure Execution", then the scope's "Stop" button. Then, increase the sweep duration, and hit "Go" again. If you are using NIDAQ, you may need to Reset the A/D using the Reset button in the AD Settings Panel. I agree with you that this behavior is extremely annoying, but I haven't found a good way to avoid it.
6. Sometimes the scope initiation is a little rocky, especially if you are using it for the very, very first time and there is no default file to start from. Some ways to force the scope to get a grip: hide and then show an A/D channel, click a pulse on and off, change the sweep duration, go to CC then back to VC. It is a good idea to save the default configuration when you finally get it where you want it.
7. Sometimes, Igor will draw a different panel's controls in the scope window, and vice-versa. I think I have caught many of those problems, but if one of your routines causes it, just close the scope window and open it again.
8. Sometimes (if you hit the up or down buttons too many times, too quickly??), the sweep duration or pulse duration may take on weird values. To fix them, on the command line, type "root:maf:mafscopesweepdrtn=X", where X is something reasonable. Ditto for "root:maf:mafscopepulsedrtn=X". Sometimes you will have to turn a channel or a pulse on and off to refresh the associated waves.

mafCam

mafCam is a set of routines to help with the SIDX package from Bruxton Corporation (www.bruxton.com). This package is an Igor XOP for controlling CCD cameras. mafCam provides a more intuitive user interface to gain access to its features. It can be used alongside mafPC and mafScope for recording imaging information during electrophysiology experiments. It requires that you have purchased and installed the SIDX XOP from Bruxton.

Basic outline



You would probably start an experiment by first initializing the camera ("Init mafCam" in the maf menu), which will bring up the camera control panel and a focus window after you select your camera. If you need to interact with the camera through another program (e.g. Cooke's CamWare), you will need to "**Release**" the camera from Igor's control, and if you want to use the camera again through Igor, you will have to quit that other program, and "**ReInit**"ialize the camera. This process is also useful to reactivate a messed up camera.

You focus on your sample by hitting the "**Focus**" button, which is analogous to the "Go/Stop" button in the oscilloscope window. Then you can use the slider to adjust the exposure time as well as the binning and the gain until your image looks right. You can check whether the camera is in a good exposure range using the "**Histogram**" checkbox. The "**AutoScale**" checkbox controls if the focus window uses grayscale values from 0 to 4095, or whether it uses the image's minimum and maximum values as black and white. If "Autoscale" is unchecked, saturated pixels are indicated in blue (for 0) or red (for 4095).

Many cameras have faster acquisition times for regions of interest (ROIs). To use one of these, you drag the marquee in the focus window over the area you are interested in, just as you normally do before you zoom in on a wave or image. Then click the "**Add**" button next to the ROI list, which will record the coordinates of that part of the image. You can then name your ROI anything you want (e.g. "soma" or "dendrite"). You can "**Delete**" these ROIs individually or "**Clear All**" at once. To see where the defined ROIs are, you can select the "**Show ROIs**" checkbox, which will draw a box around each ROI in the focus window. You can control the colour of that box by right clicking the colour in the ROI list.

Once you have focussed and maybe defined some ROIs, you can collect images. If you just want to collect the image you focussed on, you can "**Capture**" it individually. If you want to tie it into a physiology experiment, then you typically would run the camera in this manner (explanations below):

```
mafCamPrepareCamera (#images, useROI)
firstshot = mafCamNextShot ()
mafPC_RunInBackground ("mypattern", "myparameters")
howmany = mafCamRetrieveImages (#images, useROI, 1)
do
    doXOPIdle  // give NIDAQmx a chance to check on things
while (mafPC_WaitingForBackground ())
// now look at images…
```

The **PrepareCamera** call sets up the camera to take a number of images, and useROI is the number of the ROI you want to collect (use "0" for the whole field). These images must be triggered using a TTL output in your pattern, which is directed to the trigger input on your camera. Make sure that the number of images you request is the same as the number of triggers that your pattern will deliver.
The **RetrieveImages** call then gets the images after the pattern has been run. If the entire field is specified (useROI = 0), these images are named based on the "**Prefix**" (which is "shot" in the example above) plus a counter that increments each time you collect an image. If a ROI is specified, the name is based on the name of the ROI ("ROI1_" or "ROI2_" in the example). If you have checked the "**Auto Display**" checkbox, these collected images are displayed in a window.

Routines You May Find Useful

mafCamInit (): Sets up the camera for data acquisition, loads drivers, selects camera, etc.
mafCamPrepareCamera (howmany, useROI): Sets up the camera to collect "howmany" triggered images from ROI "useROI". If "howmany" is NaN, then it does continuous acquisition.
mafCamRetrieveImages (howmany, useROI, terminateAcquisition): Gets the specified number of images that you want to collect. If they are not available yet, it waits. You can interrupt with the Esc key, which will give an error. Alternatively, if "howmany" is NaN, then it gets whatever is available. Set "terminateAcquisition" to 1 if you want it to stop trying to collect images (i.e. if you got all that you expected to get). It returns how many were images were actually read. This is only useful in continuous acquisition.
mafCamNextShot (): Returns the next index value of the shot (analogous to mafITC_nextWave)
mafCamGetBase (ROI): Returns the prefix for the given ROI (analogous to mafITC_ADprefix)

Known issues
1. Sometimes the histogram window gets confused. If it does, just uncheck it and check it again. If it crashes your computer, my apologies. Save often. I blame Bruxton. Or Igor. But never myself.
2. Sometimes the camera gets confused. If it does, release it, and re-init.

Application Notes
1. mafCam has only been tested using the Cooke Sensicam. The Sensicam is only capable of dealing with one ROI at a time. So, unfortunately, you can't have a "soma" ROI and a "dendrite" ROI. You can draw your ROI as tightly as possible around the two for added speed. If you can, rotate the camera so that you use as few rows as possible (i.e. have them laid out on the camera as a short, wide rectangle). According to Dan Brown at Bruxton, you will get faster frame rates that way. Not so for columns.
2. One general issue for cameras is that they do not time-stamp the acquisition, which makes it very difficult to tell when exactly an image was taken. This diminishes the appeal of continuous acquisition. I would recommend always using triggered acquisition, because the timing is controlled and known.
3. Make sure that if you use mafPC with mafCam, that you use "mafPC_RunInBackground" and check whether you collect your images during data acquisition. The Sensicam can only

hold 2 images and it will act as though the others never appeared. This currently requires using the NIDAQmx drivers.

mafBrowse

mafBrowse is an interface to help you review the traces you have collected. It assumes that traces have a prefix followed by a numbered index, such as are generated by mafPC and mafITC. It also allows hiding of stimulus artifact and elimination of recording glitches.

Installation

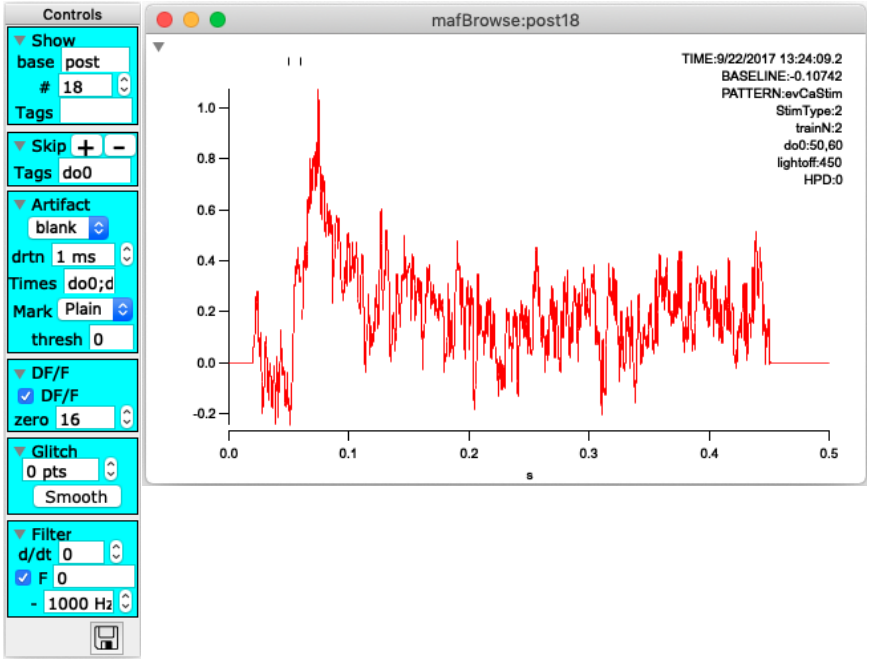
To install, place the following files into your "User Procedures" folder in the Wavemetrics part of your Documents folder:

mafutils, mafBrowse

Then run Igor, and add the following line to your experiment's Procedure Window:

#include "mafBrowse"

Then from the "maf" menu, choose "mafBrowse".



Controls

Show Group

Base: Specify the base prefix for waves you want to see.

#: Specify the index of the individual wave you want to see. Arrows allow quick scrolling through all the data.

Tags: List of keys in the wavenote that you **aren't** interested in seeing. By default all keys are displayed in the textbox at the upper right of the browse window, but many are of marginal usefulness (for example, "pattern", "time", "baseline", etc.). To hide all fields, enter "*" in the Tags field.

Skip Group

+/– buttons: Skip to the next/previous wave whose wavenote tags match the current wave. This is useful for reviewing similar stimulation conditions without intervening different ones.

Tags: Specifies the wavenote tags that must match the current wave.

Artifact Group

Popup: Allows automatic removal of stimulus artifacts, with the duration specified in the "drtn" field. Options are "blank" to simply erase the artifact, "smooth" to interpolate a straight line across the artifact, and "none". **The original wave is unchanged.** To make a copy of the modified wave, you would need to duplicate the trace (e.g. `duplicate mafbrowsewave_0 myNewWave`)

drtn field: Specifies the duration of the stimulus artifact.

Times field: Specifies the times of the stimulus. This can be either particular time points (e.g. "10") or parameters from the wavenote (e.g. "do0"). Multiple times or parameters can be specified, separated by semi-colons. Times are in milliseconds. Stimulus times are marked with vertical lines above the trace.

Mark popup: Indicates how stimulus times should be marked. Options are "None", "Plain" which uses vertical lines ("|"), "Fails" which uses an "X" if the response following a stimulus falls below the threshold specified in the threshold field below, "Num" which labels stimuli by their number.

Threshold field (unlabeled): Specifies the minimum amplitude for identifying a failed response after a stimulus. Useful when the Mark popup is set to "Fails".

ΔF/F Group

ΔF/F checkbox: Calculates the ΔF/F of the wave, with reference to the zero wave number specified. This is to be used for calcium imaging with a photomultiplier tube or photodiode.

Zero field: Holds the index of the wave that acts as a zero reference.

Glitch Group

Helps eradicate trace acne, i.e. brief jumps in voltage or current that originate from pump noise or the suction line. Place the round cursor (cursor A) on a point right before an ugly glitch, and set the number of points to erase. Hit the button and >poof< now that ugly glitch is replaced with a straight line. **This changes the original wave and is not undoable.**

Filter Group

d/dt field: Allows reviewing an arbitrary derivative of the wave.

F fields: Apply a 4-pole Bessel filter with the specified low and high cutoff frequencies. A frequency of "0" means no cutoff.

🖫 button: Saves all the settings in the window to a preferences file on the disk, except the wave base name and number, and the zero wave. These preferences become the defaults when a new mafBrowse window is instantiated.

Contract/Expand buttons: Click the little minus signs to minimize each control group. The little minus sign on the main window will hide the control bar completely. These can be restored by hitting the plus buttons. (A bug in Igor may make the control panel reappear when you resize the mafBrowse window. They said they would try to fix this.)

Routine You May Find Useful

mafBrowse (prefix, index, zero wave, instance): Opens up or updates a mafBrowse window. The first three parameters set up the wave to be looked at, and a zero wave for doing $\Delta F/F$ measurements. The instance parameter is a way for you to instantiate multiple distinct browse windows.

Known Issues

If the wave note has one really long line, the whole thing may seem to disappear. It is good to make use of the "Tags" feature in that case.

Application Notes

1. mafBrowse is useful both during data collection and afterwards. It can be configured to hide the stimulus artifact in various ways. These do not change the original data, just the display.
2. You can review two channels of data independently, using the *instance* feature. For example, say you are doing paired recordings and store data as waves with base names "pre" and "post". To review both, you would specify different *instance* values when you call mafBrowse:

   ```
   mafBrowse ("pre", 0, 0, 0)
   mafBrowse ("post", 0, 0, 1)
   ```

   This will display pre0 in one window and post0 in a second window. If you reuse an *instance* value (e.g. by calling mafBrowse ("pre", 1, 0, 0), that will update the display to show you the new wave (in this case, pre1). The menu call of mafBrowse assumes an instance of 0.

mafDC

mafDC is an XOP that controls a fast dynamic clamp. It runs inside Igor, mainly for the sake of the user interface. It must be run on a PC, because it uses the National Instruments hardware. The purpose of dynamic clamp is to mimic a conductance, and mafDC is capable of mimicking multiple types of conductances: a leak, inhibitory or excitatory synaptic conductances, and channel conductances. You can mimic GABA, AMPA, and NMDA type synaptic conductances. Channels can be a Hodgkin-Huxley-type sodium conductance or a general-purpose Markov conductance, with arbitrary reversal potential and number of states. Its main value is that it is extremely fast (thanks to Lorin Milescu for his invaluable help). It relies on the information maintained by mafITC to interact with the amplifier.

Installation

To install, place the following files (or shortcuts) into the appropriate folder in the Wavemetrics part of your Documents folder:

Into "User Procedures": mafutils, mafITC, mafDC helper, (optionally, mafScope7)
Into "Igor Extensions": mafDC XOP
Into "Igor Help Files": mafDC Help

Then run Igor, and add the following line to your experiment's Procedure Window:

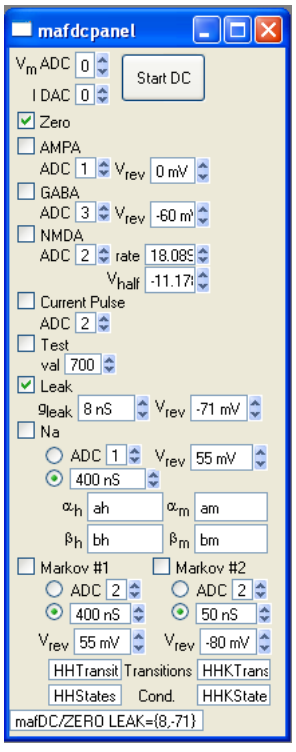
#include "mafDC helper"
(optionally also #include "mafScope7")

Then from the "maf" menu, choose "mafDC Helper Panel".

The scope window is not necessary, but it can be helpful in setting up the dynamic clamp for the first time.

Using mafDC

The features of the mafDC XOP are explained in detail in a paper (Yang et al., 2015, J Neurophysiol 113: 2713), and in the Help file. The user is encouraged to look at these resources for additional explanations. Briefly, mafDC runs on a slave setup, i.e. with its own PC, NI data acquisition system, monitor, keyboard, mouse, etc. Ideally it is just set running, while you interact primarily with the Master setup to specify conductances, and to record cellular responses. Here we explain the mafDC Helper Panel, and some strategies to get going.



Helper Panel

At right is the panel to facilitate calls to the mafDC XOP. The call itself is shown at the bottom of the panel. The checkboxes are used to indicate which components of the dynamic clamp are desired, i.e. synaptic conductances (**AMPA**, **GABA**, and/or **NMDA**), the current pulse pass-through, a leak conductance, and/or channel conductances (Hodgkin-Huxley **Na**, or general-purpose **Markov**). It also specifies whether the dynamic clamp should reset the current to **Zero** when it terminates, or hold the last value, and whether the dynamic clamp is run in **Test** mode. Each dynamic clamp component has its own parameters, the amplitude for static conductances or the A/D channel for changing

ones, and usually the reversal potential. When all the desired components are set up, you hit the "Start DC" button, and the dynamic clamp starts. To interrupt, it is necessary to hit Igor's "Abort" button at the bottom of the screen.

Issues

1) Amplifier issues. To calculate the correct currents to pass, the dynamic clamp on the slave setup needs to know the membrane potential (usually through A/D channel 0). The master setup, of course, also reads the membrane potential. In our experience, Axon amplifier outputs are too weak to be read on two A/D systems (they even seem to have trouble with one), so we do not recommend putting a BNC T-connector on the amplifier output. This will lead to measurement errors. Use a secondary output instead (e.g. "Scope" on the Multiclamp 700B).
2) Setting up conductances on the Master. When the conductance amplitude fluctuates with time (e.g. synaptic conductances), this is accomplished by specifying the conductance waveform on the Master setup, and sending it out a D/A to an A/D on the Slave. If you are using mafPC, you can use the CONVOLVE event to specify time-varying conductances. The explanation on p. 15 above is sketchy, but it works quite similarly to the dynamic clamp feature for the ITC18 (p. 12). That is, you should generate an Igor wave with your typical conductance waveform (e.g. an EPSC), scaled to +1, and clip off any extra junk where the conductance is very nearly 0. In your pulse pattern, specify a number of CONVOLVE events when you want the pulse to occur. mafPC inserts that wave scaled to the specified amplitude. This convolved wave is then sent out the D/A to the dynamic clamp setup. While this convolution approach works well for AMPA receptors (at least in our system), GABA and NMDA receptor currents may not add linearly. This is easy to verify using voltage-clamp recordings, and comparing the IPSC/EPSC resulting from train stimulation *vs*. the IPSC/EPSC predicted by convolving a single IPSC/EPSC with the amplitudes and times in a train. In our experience, this does not work well for NMDA trains. We instead record these train EPSCs and pass them entire to the dynamic clamp using the wave feature for AMP events (see part 9 of mafPC).
3) Gains. It is best to take this step by step, working with a model cell. The membrane potential readings on both Master and Slave (usually A/D channel 0), as well as the Slave current output (usually D/A channel 0) should be as according to the amplifier gains. Run mafScope on the Slave setup to double check it receives the correct conductance amplitude from the Master. (Caution: If you run the scope window on the Slave setup, you will need to Reset the A/D in the AD Settings Panel to free it up before invoking mafDC.) Also, apply a leak conductance in mafDC, and verify that the membrane potential reaches the expected value of $V_{rest} = g_{leak} E_{leak} / (g_{leak} + g_{model\ cell})$.

   For each time-varying conductance sent from the Master (e.g. synaptic conductances), the D/A gains on the Master setup should be the same as the A/D gains on the Slave (e.g. 100 mV/nS). You can verify these are set correctly by delivering square pulses of conductance from the Master while running mafDC, and seeing that the responses have the correct reversal potential and reach the correct membrane potential (using the formula above). For this, it is simplest to set the model cell's membrane potential by passing current using the amplifier controls.

   The current pass-through (PULS) feature lets you send current/voltage pulses so you can patch and then do dynamic clamp without moving cables around. You can verify the

current/voltage being requested using mafScope on the Slave. We set the output from the Master to 50 mV/mV (voltage clamp) and 2.5 mV/pA (current clamp), and the input on the Slave to 2.5 mV/pA, and it seems to act properly when the Master is in voltage or current clamp, even though the Slave stays in current-clamp mode.

4) Adjusting the amplifier. It is important to properly compensate for pipette capacitance. This should be as large as possible without causing the electrode to ring. This is done by hand in current clamp. If using single-electrode dynamic clamp, the bridge should also be balanced. This is done by passing brief current pulses, and adjusting the bridge until the fast transient is reduced. If the access resistance changes during the experiment, this will need to be adjusted. If at all possible, we recommend using two-electrode dynamic clamp. It isn't as bad as it sounds, and it eliminates weird capacitance transients from the recordings. In our lab, we lower both electrodes onto the cell at the same time, then seal both, and break in one by one.
5) Voltage glitches. Usually, the recorded membrane potential is used by both you as the experimenter as well as by the dynamic clamp to determine the currents to pass. So, how do you use one output on two devices? You would think you could just put a BNC T-connector and split the voltage output from your amplifier. However, this can cause two problems. One is that the output of the amplifier may not be powerful enough to support two readers. This was our experience using the Multiclamp 700B. The other problem is that when the NI boards read the voltage, they tend to introduce a glitch in the signal. This glitch doesn't degrade the signal read by that NI board, but if there is another NI board or an oscilloscope or whatever also reading the same line, it may pick up that glitch. So, don't use a T-connector on your voltage output. If at all possible, use two different outputs (the Multiclamp 700B has 3 separate outputs). If your amplifier has only one, well, you can always see if this causes problems. Maybe passing the signal through a second amplifier could help.
6) Ghost conductance. Sometimes, when you start mafDC, you may see an immediate change in resting potential, even though there is no conductance requested. This is what I would call a "ghost" conductance. It may indicate that the D/A from your Master has some offset. This offset is typically small (e.g. 5 mV), but if you are using high gain to deliver large conductances, this could still yield a significant conductance. There are three possible solutions. One is to add a bias on your Master output (e.g. apply some negative conductance so there is no change when you start mafDC). The second is to interpose a voltage follower on the output from your Master, that adds a DC offset (as in "direct current"). That's what we do. The third is to complain to the manufacturer, who will ask you to send it back for repairs, and you will wait a few weeks and pay several hundred dollars to get it back with a different offset, that is formally within specs. Look on the bright side, though, if you can think of one.
7) DAQ options. We obtained the highest speed using a PCIe-6361 board on the Slave setup, upwards of 70 kHz, as assessed using Test mode. We got this as of 2017 using a Windows 10 computer running Igor 7, and we had similar speeds using Windows XP running Igor 6. We have also had good results with PCI-6221 and PCI-6229 boards. We experimented with a USB-6361 interface, and it was terrible. Cycle times of greater than 1 ms, as well as huge latency and jitter of response times (using echo test mode). If anyone has better luck with one of these USB interfaces, let me know.

Recipes

This section gives little code snippets to help you get started. Feel free to suggest others.

1. Simple loop

Suppose you want to stimulate a pathway with pairs of pulses every 10 s. One solution is to make a pulse pattern that delivers the pulses (e.g. "TTLTrain" in part 4), and write a routine in a Procedure file similar to the following:

```
function simpleExpt ()
    do
        mafPC_Run ("TTLTrain", "stepDrtn:5;trainPause:9.8;trainN:2;trainIPI:10")
        sleep /s 10
    while (1)
end
```

This passes the required parameters (stepDrtn, trainPause, and trainN) to the pulse pattern. In addition it passes an extra parameter (trainIPI) so that it will be easier to figure out what we actually did. Programs like this will print data in the history window, and have to be interrupted with the "Abort" button, which is dangerous.

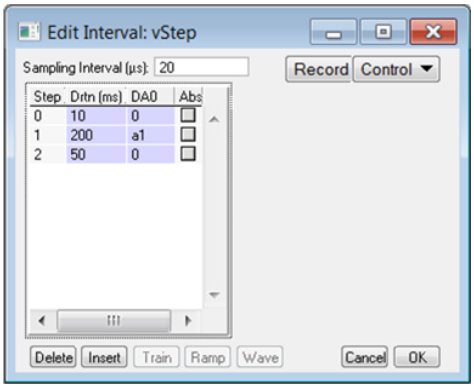
This basic outline can be significantly improved. Major improvements would be to:

1. Display the trace. The simplest solution is the AutoDisplay feature in mafITC (see ITC AD Settings). mafBrowse has more useful features.
2. Measure summary data. Analysis could go right after the call to mafPC_Run, to measure EPSC or EPSP amplitude, access resistance, leak current or resting potential, etc. The mafITC_lastWave function is useful to reference the trace you just collected.
3. Exit gracefully. Using the "Abort" button to stop the loop can mess up mafITC and mafScope. It is better to check for a key press or make the function run in the background.
4. Improve precision. Igor's sleep function is somewhat inaccurate, so for very time-critical situations, it is better to do it by counting ticks (or use the mafSleep function in mafUtils).
5. Mix in different stimuli. Additional calls to mafPC_Run could be added to execute different types of stimuli, or have trains at different rates, or with different numbers of pulses, etc.

```
function betterExpt ()
    variable nextstim, thisindex
    do
        nextstim = ticks/60 + 10        // aim for next stim to be 10 s from now
        mafPC_Run ("TTLTrain", "stepDrtn:5;trainPause:9.8;trainN:2;trainIPI:10")
        // do some quick analysis of the pair of EPSCs
        thisindex = mafITC_nextwave() - 1
        EPSC1[thisindex] = wavemin ($mafITC_lastWave(0), .011, .015)
        EPSC2[thisindex] = wavemin ($mafITC_lastWave(0), .021, .025)
        PPR[thisindex] = EPSC2[thisindex] / EPSC1[thisindex]
        leak[thisindex] = numberbykey ("BASELINE", note ($mafITC_lastWave(0)))
        doupdate
        mafBrowse (mafITC_ADPrefix (0), mafITC_nextWave () - 1, 0, 0)
        if (mafsleep (nextstim))        // returns 1 if esc pressed during sleep
            break
        endif
    while (!(getkeystate(0) & 0x20))    // runs until esc pressed
end
```

2. IV curve

To execute an IV curve, modify the Vstep pattern to look like the diagram at right.



Next, enter this simple function in a procedure window:

```
function doIVCurve (startV, endV, deltaV)
    variable startV, endV, deltaV
    variable i

    display
    make /o/n=((endV - startV) / deltaV + 1) wresult
    setscale /p x, startV, deltaV, "mV", wresult
    for (i = 0; i < numpnts (wresult); i += 1)
        mafPC_Run ("vStep", "a1:" + num2str (startV + deltaV * i))
        wresult[i] = mean ($mafITC_lastWave(0), .15, .2)      // get steady-state current
        appendtograph $mafITC_lastWave(0)
        doupdate
        sleep /s .2
    endfor
    display wresult
end
```

To run the IV curve, just type in the command line, something like:

```
doIVcurve (0, 100, 10)
```

Note that this is using relative voltages, not absolute. The cell's absolute voltage would be controlled with mafScope, then run "doIVcurve". These voltages could also be specified to be absolute by checking the "Abs" checkboxes in the pattern, and replacing the 0's in DA0 with −70 mV or something. StartV and endV would also have to be adjusted.