MFP3D Software Manual

Version 20030912

Manual revision 1.0
9/03

Asylum Research
341 Bollay Dr
Goleta, CA 93117
USA

Voice: 805-685-7077
Fax: 805-685-5007
Email: Support@AsylumResearch.com, Info@AsylumResearch.com
Web: http://www.AsylumResearch.com

Table of Contents

Volume I Getting Started

Volume III Programming Reference

- **MFP3D Getting Started**

This help file is here to help get you started with basics of Igor programming from the point of view of the Asylum Research MFP3D software. It assumes that you have gone through Volume 1 of the Igor manual, but that you are quite new to Igor. If you have just gone through Volume 1 then you will notice quite a few similarities between this help file and chapter 2. If you have NOT gone through volume 1, then press F1 to open the help window, click on the Manual tab, and click on the open online manual. This will open the pdf of the Igor manual. You should take the time and go through the steps in chapters 1 and 2. Once you have a good feel for that, then you can come back to this help file and learn your way around the Asylum Research software.

TOC

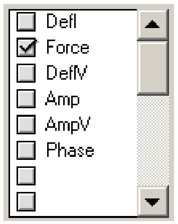
Advanced Topics

MFP3D Basics Glossary

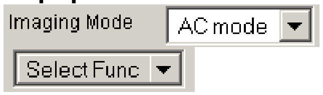
Some of the basic concepts of the Asylum Research Igor interface that are most often confused by new users.
Controls: Our help helps often refer to the controls by their type. This can be confusing if you don't know what these types are.
Button: Pretty intuitive.

Checkbox: again pretty clear.

Listbox: A control that lists multiple items that often has scroll bars on the right side. These control can (but only rearely) have multiple columns, they also some times have checkboxes in each element (cell). Most of the listboxes allow you to have multiple selections at once by holding down the shift key when you select them. Since Igor's listboxes can do so much, they tend to be difficult to use. One of the biggest problems is from going to multiple selections to a single selected force plot. I have found that the best way to do that is to hold down the mouse button on the list box and move it up and down a little, maybe going up to the next item in the list. For some reason this seems to help Igor out in realizing that Yes, you want to select one item, and that is the item you want to select. Then once you get Igor to show you that it has one item selected you can move the mouse over the item you really wanted selected and finally let up the mouse button.

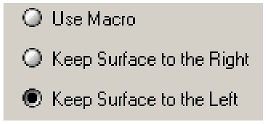


Popupmenus:



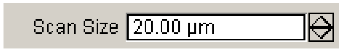
These controls operate in 2 basic modes. One where it shows the selected item, and one where it shows the title, and you have to click on the popup to see what options you have.

Radio buttons:



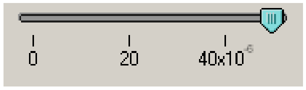
These are basically just checkboxes, but shown a little differently. They operate in exactly the same way as checkboxes. From the standpoint of intuitive interfaces, radio buttons should be grouped together, and all the radio buttons should only allow the user to select one at any given time. Selecting one should de-select the other radio button in the group. But Igor leaves that work to the programmer.

SetVars:



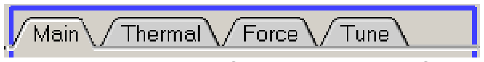
One of the most powerful controls in Igor. These controls allow you to enter new values for a variable. There are often up and down arrows to the right of the setvar (which we often refer to as clickVars), that will increase or decrease the variable.

Slider:



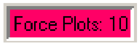
Also pretty intuitive.

TabControl:



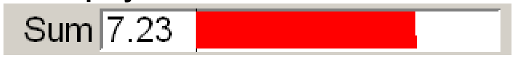
The tabs that are often on the top of panels. We use these controls extensively to hide other controls to make the interface easier to manage.

TitleBox:



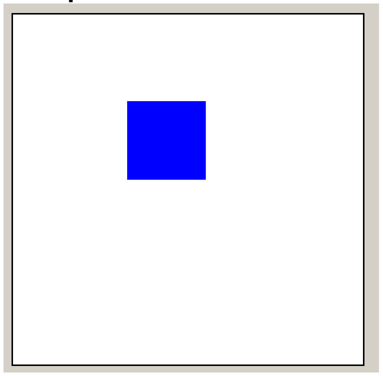
Just that, a little box with a title to help describe what the controls around it will do or to give you some info.

ValDisplay:



Much like a setvar, but without the ability to enter values. This control is only there to show what a value is. But they can have the nice little bar like on the sum and deflection meter. When a valDisplay does not have the bar, it looks <u>very</u> much like a setvar without the clickvars.

Group box:



This is just a colored box, mostly designed to add color to parts of a panel. We have built up on this to use them for the Force bar and the ScanBox, but these are not the intended uses of group boxes, which is why it takes so much work to make them do what they do.

Window Types:
Igor has a couple different types of windows it can have up. These are not windows in the classical sense, since they are all within the Igor window, but we will call them windows anyway.
Panels:
Panels are much like graphs, but can ONLY have controls on them. They are designed to provide a control interface.
Graphs: designed to contain data plots, traces, contours, Images, etc. They can have controls on them, but that often gets messy. Most of the time when controls are placed on a graph it is with the use of a controlbar, which basically just marks out an area on the top of a graph for use by controls.
Tables:
Spread sheets of wave data.
NoteBooks: Just what the name implies, a simple little text program built into Igor to contain notes, but advanced enough to allow you to insert graphics.
Procedures:
Windows much like notebooks (without graphics) that contain code.
Help Windows:
Same thing as notebooks, but locked down so you can edit them.
Layouts:
Layouts are a little weird. They are basically just a window that imports other windows into it. So you can have a layout with a table and a graph on it. They are designed to make pretty print outs.

XOP windows: An XOP is an external set of code that Igor uses. These XOPs are written in a lower level language (see MFP3D Generalized Waveform Input and Output), but they can create windows in Igor. Since they are created with a different set of code than the stock Igor functions, their behavior can be a bit different. The data browser and MFP3D video window are examples of that.

Function Types:
There are basically 3 types of functions, and 3 classes of functions.
In this help file I have been referring to all function types as functions, because I don't think there is much of a difference, but here is a summary of the differences between the different types and classes.
Types of functions:
Macros:
Suck, execute slowly, are not compiled, which means you will not get any warning about errors in the code, and harder to debug. They are a hold over from days of old, and supposedly are easier to write for beginners. I don't buy it personally.
Functions:
Your standard function with multiple arguments of strings, Variables, and waves (even FuncRefs if you really feel like it). Limited to 1 variable or string output.
Operations:
Much like functions, but do not return anything. Many operations place variables in your workspace. The biggest distinction between operations and functions from a syntax point of view is that operations do not have the (and) bracketing the arguments, the argument list is most often separated by ","

Function Classes:
Built - in:
Igor's stock functions and operations. You can't get to the code for these, but they have easy to find help on all of them.
User Defined:
These are functions and Macros included by us, and you. These you can get directly to the code, as they are all defined by the procedure windows in the experiment.
XOP:

External code: Advanced users can write XOP's which are sets of external code written in a lower level. This allows the functions and operations to run faster, and do work that is not easy to do using Igor's built - in functions.

- **Getting Help**

Sub Toc

Help files

There are numerous help files included in the AR software. Most of these describe how the controls work, with a few addressing advanced programming topics. To find these help files there are 2 primary paths to follow. The Igor menu, Windows -> Help Windows lists all of the help files that Igor knows about. All of the AR help files start with MFP3D, and clicking on one of these help files will bring it up. The other path to open the AR help files is from the Igor menu Help -> MFP3D Help Files, which lists all of the help files that we have included.

"?" Controls

On nearly every AR control panel there are controls on the right that have a "?" in them. Usually buttons, but sometimes popup menus. Clicking on them will bring up help topics related to the action of the controls to the left of the "?" help control. This way you can learn what each of the controls do.

PDF manual

We are working on a couple PDF manuals, none of which are done, and really require this file to be written first, so I really don't know where you will be able to find the PDF manual, but we will have one.

Asylum Support

For additional help you can contact Asylum Research. The support web page can be loaded from the Igor menu Help -> AR support, which basically just tells you to either call us at 1-805-685-7077 (Pacific Time), or email us at Support@AsylumResearch.com. If your problem is particularly difficult to describe over the phone, there is an internet tool that allows us to see and even control your computer (for as long as you allow us to). This internet tool can be accessed from the Igor menu Help -> AR - Web Control Support.

Igor Functions

But by far the easiest way to get help for built in Igor functions is something that is hardly mentioned anywhere in the help files. If select the name of a function, anywhere in Igor, and then right click on it you will have a menu item that says Help for FuncName. If you select that it will bring up the help file for that function. So later in one of the examples we use the function hcsr, which is pretty hard to figure out what it does from its name. So go ahead, put your mouse cursor one the word hcsr, and right click on that, and select Help for hcsr. This brings up the Igor help file for hcsr. Very nice hu?

This is really helpful when reading someone's code, say you don't know what the /K flag does for the Display function, so you click on the word display, right click on it, select help for display, and there it is, exactly what the /K flag does in display. This only works for built in Igor functions (and operations). For user defined functions (like the AR software), you get a different option, Go to FuncName. That brings up the code for that selected function. You want to try it out,
OK, try it out on the name MFPVersion.

• Guided Tour of MFP3D

Sub Toc

Overview

This chapter is here to help you jump start on some of the intermediate topics in Igor that are routinely used in the AR software. The goal being to help you navigate around the AR software's workspace and to allow you to start writing your own code.

Assumptions

Much is being drawn from Volume I from the Igor manual. if you have not gone through that guided tour, do it twice. The top of this help file describes how to find the guided tour.

Loading the template

The first step to starting the AR software is to load the MFP3D Template.pxt. This is what you are loading when you click on the MFP3D shortcut. The pxt file is a template, very much like a pxp file, but that you have to work save over the pxt. You should NOT overwrite the MFP3D Template.pxt file. If you want to have your own set of preferences, you should save a pxp or set up your own user parameters (see Creating a User Folder). You can also start the template over by selecting File -> New MFP3D Template, or it is often listed in the recent experiments, also in the file menu.

Data_Browser

One of the most complicated things to understand for a new user are Data Folders. But by the same token, they are one of the most critical components to getting around the AR software. Likely wavemetrics has provided the Data Browser. This tool is a great way to visualize where you are and what data is readily available, as well as figuring out where to get your data. The data browser can be opened from the Data - Data Browser menu. So go ahead and read the help on the Data Browser. Did you catch the little blurb about the Alt clicking next to the data folders listed? That is key to switching data folders, because dragging the red arrow around is painful. You should play around with the data browser a bit. Believe me, spending the time to familiarize yourself with this tool will pay itself back with time saved trying to work with Igor.

AR Data Folders

So now that you have the Data Browser open, I am sure you have noticed that there are quite a few data folders in there. Data folders are not even touched upon until chapter 8 of volume II, presumably because you can get a lot done with Igor without ever dealing with Data Folders. However you can't get any custom work done in the AR software without understanding exactly how Data Folders work. You should defiantly read up on Data Folders and Data Folder Syntax now.
OK, so let's go over some of the AR data folders.
So un-select waves, variables and Strings, so that the data browser is only showing data folders.
Make sure you are in the root: folder
From the root folder you see 4 folders listed.
Packages
Images
ForceCurves
WinGlobals
and
SavedLitho

3 of those are for saved data, ForceCurves, Images and SavedLitho, we purposely set them off the root to make it easier to get to. WinGlobals is there so that we can use some advanced features of Igor's Cursors, which require us to place the folder there with that name. Then there is the packages folder, which contains MFP3D. That folder root:Packages:MFP3D: is where we try to keep the vast majority of the data. This is so that most users will not have to be bothered by it. We will cover these deeper data folders later. If you select waves now you will see a big old list of waves spring up. As of this writing we have about 70 waves in there. Most of these are the waves used to store the Image data as is it being collected. Go ahead and select DeflectionImage1 (Deflection Retrace Image), and make sure the Plot checkbox is selected. Now you probably have not scanned anything, so your wave is empty, and it does not plot anything for you in the data browser. So go ahead and click on the Info checkbox. Here it lists some details of the wave, such as the fact that it has 1 row and 1 column. Not very exciting hu? Hang in there.

Entering Data

You know, you already did this in the Getting Started guided tour.

Make a Graph

OK, there are basically 3 ways to get a graph started from scratch. The quick way, the menu drive way, and the hard way.
The quick way is with the data browser. Yeah it shows you a nice little plot down there of your images and lines, but you don't have any scaling there, no idea just how big or small your data is. BUT, if you right click on a wave you get a little menu. One of those options is Display. Why don't you go down in the :MFP3D:Main folder and select ScanBoxY, and right click on it, and select Display. Now you have a graph of ScanBoxY, and you are asking your self what the hell am I looking at, hang in there. I am pretty sure that if you call up Jason C., he would be willing to give you some money for going through this. Before I tell you what ScanBoxY is, I want you to go down to the command line and see what the

amazing Data Browser has placed there for you. It should say:

```
  Display /K=0 root:packages:MFP3D:Main:ScanBoxY
```

I want to use that to jump start you on the hard method of making a graph (skipping the menu way for now).
So push the up arrow key to highlight the command that the data browser did, and hit return. That should bring that command down into the active area of the command window. Select the root:packages:MFP3D:Main:ScanBoxY part of the command and copy that to the clip board.
Then after the

```
  Display /K=0 root:packages:MFP3D:Main:ScanBoxY
```

add
vs and paste in root:packages:MFP3D:Main:ScanBoxY
finally changing the ScanBoxY to ScanBoxX (I am sure you noticed that wave earlier from the data browser).
So your command should read

```
  Display /K=0 root:packages:MFP3D:Main:ScanBoxY vs root:packages:MFP3D:Main:ScanBoxX
```

and hit enter.

Now you have a graph that is plotting ScanBoxY vs ScanBoxX, and it should make a graph in the shape of a box with an extra point at 0,0.

OK, I will finally tell you what the hell these 2 waves are. They are used with the hidden Scan Box Control on the Main panel. If you go to the main panel (see help topic MFP3D Main Panel, Show Box), you will see a button at the bottom that says Show Box, click on that. Now you have a little blue box surrounded by a big white box. The blue box shows you what region you are going to scan, the white the total range of the scanner. So go ahead and drag the blue box around with the mouse. Yeah, the blue box moves around. But hey, look at the graph you made, it is changing as well. pretty cool hu? Now this illustrates an important concept in Igor. Notice how the graph you made updates when you drag the blue box around. The code that is run when you drag the box around changed the waves ScanBoxX and ScanBoxY. In many software environments the graph would not update when the wave is changed. In Igor there are things called dependencies, which we will get into a little later, but it basically links one thing to another. In this case if the waves change, then the graph is updated. The graph updating is a built in dependency of Igor, so you never have to worry about setting it. But this concept of dependencies can be applied to just about anything in Igor.

OK, and finally how to make a graph with the menus.
First off, set your data folder to root:Packages:MFP3D:Force:
by hold down alt and clicking to the left of that data folder in the data browser.
now go to the Igor menu, Windows -> New Graph
which brings up the new graph dialog.
Well hey, you have already seen this from the Getting started tour
But select in the Y wave listbox, OptionY and in the X wave listbox select OptionX
And click DoIt.
Now you have a graph that shows you some force plot. This is actually the force plot shown on, well it is not shown on any thing anymore, it USE to be shown on the graph orientation panel for the force panel.

Getting info from the graph

The easiest way to get data values off of a graph is with Igor's built in cursors. Not cursors in the sense of your mouse, but little markers to be placed on points of the waves. To get the cursors up, hit Ctrl + I, now you have a round marker and a square marker. Those are your cursors. There is some help in the Igor files under Cursors and Info box. But that does not cover too much. On the other hand, it is pretty intuitive. Drag the round cursor (cursor A), onto your OptionY graph. Press the left arrow key, press the right arrow key, watch your cursor move. Hold down the shift button and press the right arrow key, watch your cursor move faster. I am sure you noticed the numbers down there on the "Info box", telling you what point, Yvalue and X value your cursors are sitting at. OK, let's try something simple, and get the values of the cursor from the command window

Let's make some variables
```
  Variable AvalX, AvalY, BvalX, BValY
```
so there are some functions that pull out the X, Y, and index values from the cursors.
```
  AvalX = hcsr(A)
  AvalY = vcsr(A)
```
horizontal cursor and Vertical cursor.
then let's put up the B cursor (square)
and type into the command line
```
  BvalX = hcsr(B)
  BvalY = vcsr(B)
```

so now we can calculate the slope between the cursors
```
  Variable Slope
  Slope = (BvalY-AValY)/(BvalX-AValX)
  print slope
```

Adding plots to the graph

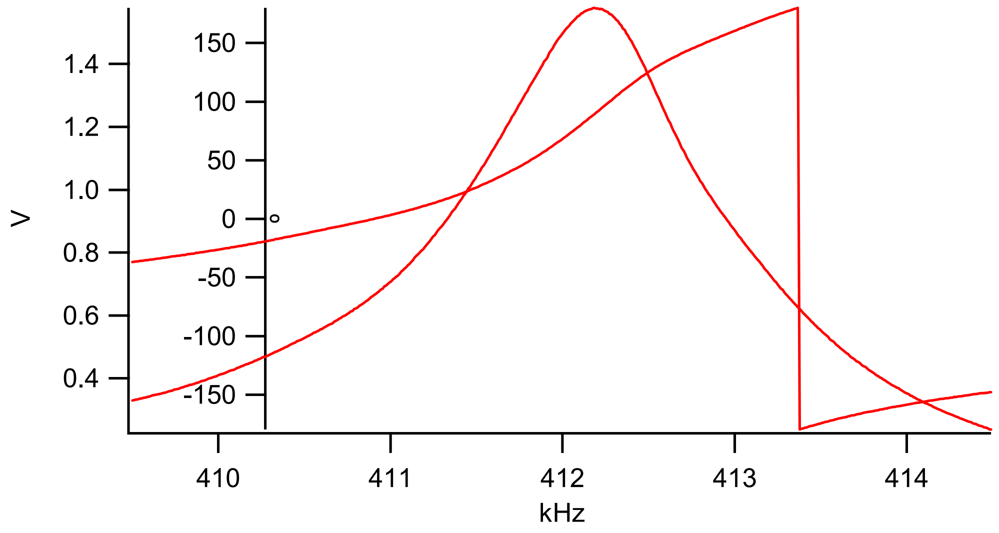
OK, so you can make a graph with one line on it, how do you make a graph with multiple axes? Let's go down to the root:Packages:MFP3D:Tune folder. Make a graph of Amp vs Frequency. OK, Now when the graph is topmost you have a different menu, the Graph menu. Go to that and select Append Traces To Graph.
This is very similar to the make graph menu and works in much the same way. So select Phase for Y and Frequency for X. Now, that alone would place phase on the same axes as amplitude. So let's select for a Y data an axis of Right. Notice how the command below changes when to switch the selection. That is another very useful tool, that gives you a good hint that the /R flag in the function AppendToGraph puts the plot on the right axes. ok, click DoIt. And now you have your graph with the Amplitude in volts on the left axes and the phase in degrees on the right axes.

Saving your Graph

OK, so you want to be able to make your graph again. Make the graph topmost. Hit Ctrl + Y. Now you have a well hidden dialog that gives you the Graphs name and title. The title is only for display, it is the string that shows on the graphs titlebar. The graphs name is what is used to address that graph. You also have a checkbox to create a Window macro. Don't worry about the style macro, that is for superficial extras, like line widths, fonts and the such. So select the create window macro checkbox and click do it. Now you have a macro in the procedure window. The procedure window is a procedure that is built into the Igor experiment, it is different than the included procedures in a couple of ways, but don't worry about them just yet. To bring up the procedure window hit Ctrl + M (or go through the windows -> procedure window menu). Now in there are a few functions we put in there, and down at the bottom there is a macro with the same name as the graph you had made. Calling this macro will make another graph just like the one you had. It gives the new graph a name that will not conflict with other graphs. But now you can see all the code that you need to make your graph. Notice that one of the first things it does is stores the string of the folder when it was called, then it sets the folder to where the data is, and then it does its work. When it is done it sets the folder back to where it started. This is generally good form, since other functions may expect to be in a certain folder when they are called, and you probably don't want the data folder changing around on you when you call functions.

Multiple Axes on your Graph

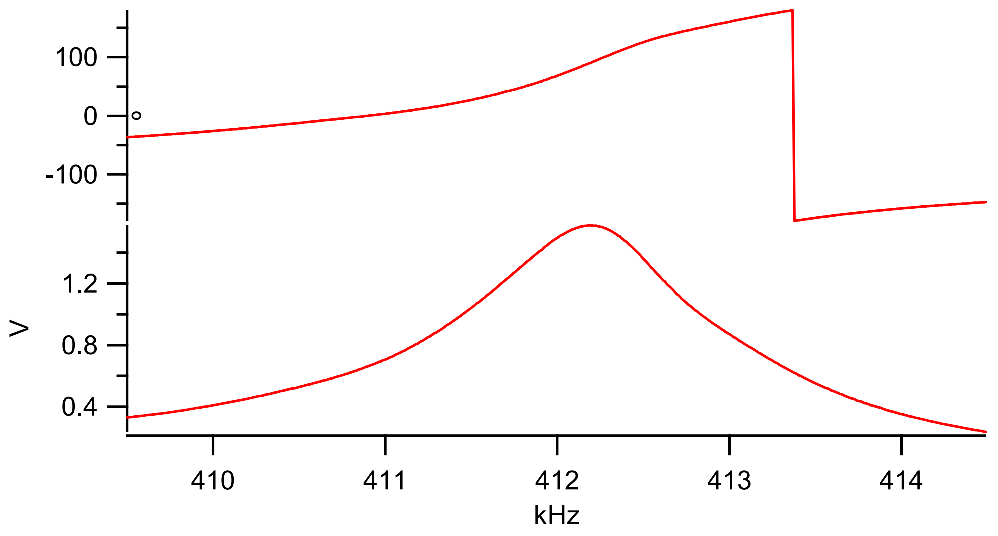
OK, so you can put your data up on separate axes, but how do you make a graph that has stacked axes? You can also do this from the Append Traces To Graph menu. First off let's remove the phase from the graph you have made in the previous steps. Right click on the phase trace and select Remove Phase. There is also a menu in the Graph menu labeled Remove from Graph that gives you a more complicated dialog to remove items from the Graph. Both of these methods will print out the command used in the command window. OK, so now you just have amplitude on your graph. Go to the Append Traces To Graph menu. Select phase for the Y data and Frequency for the X data. Now for the Y axis select New. You get a input dialog asking for a new name, let's call it PhaseAxes. Now click do it. Your graph should look like this:



Kinda hard to read hu? Yeah Igor does not like to help you out much with "free" axes like the one you just created (PhaseAxes). But you can fix it up.

Double click on the PhaseAxes, that brings up a dialog with a whole bunch of tabs. You want the first one, Axis. In order to stack the axes on top of each other you want the Draw between controls in the middle. This is percentage of the graph that this axes will be drawn between. So let's set the phase to go between 50% and 100%. You also have noticed that the PhaseAxes is offset laterally to the Left Axes. This is controlled with the Free position controls to the lower left of the Axes tab. Let's set the Free position distance to zero. OK click do it. Now you have set the PhaseAxes, let's fix up the Left axes. Double click on the Let axes (with the amp wave on it). You get the same dialog, but now the popup in the upper left corner says left, where before it says PhaseAxes. For the left axes you want to set the Draw Between to 0 to 49%, and click DoIt.

Now your graph should look like this:



Much better hu?
And I am sure you were paying attention to the commands that were being spit out in the command window as you modified the graphs. Don't worry too much about those. ModifyGraph is one of the nastier functions to use, since it has so many different things it does.

Updating your Graph code

OK, so you have changed your graph, and you want to update the code to recreate the graph. Hit Ctrl + Y again. Now the checkbox is labeled Update Window Macro. Select that and click doit. Now when you go back to the procedure window (Ctrl + M) your code has been changed and should look something like this:

```
Window Graph2() : Graph
	PauseUpdate; Silent 1		// building window...
	String fldrSav= GetDataFolder(1)
	SetDataFolder root:packages:MFP3D:Tune:
	Display /W=(34.8,120.2,430.2,327.8)/K=1  Amp vs Frequency
	AppendToGraph/L=PhaseAxes Phase vs Frequency
	SetDataFolder fldrSav
	ModifyGraph lblPos(left)=46
	ModifyGraph freePos(PhaseAxes)=0
	ModifyGraph axisEnab(left)={0,0.49}
	ModifyGraph axisEnab(PhaseAxes)={0.5,1}
EndMacro
```

Saving your work

So you have done some custom work, putting your code in the procedure window, and you want to go home and eat something before you pass out with exhaustion. Time to save your work. Igor can save what it calls experiments. Experiments are basically everything in Igor's memory, all the graphs, panels, waves, variables, just about everything. There are basically 4 types of saved experiments.
Packed, this is the one you want to use, it contains all the data in one file.
Unpacked, this one actually will not work with the 3D software. There is a data folder called Variables, which conflicts with how Igor saved Unpacked experiments. But if it did work, it would build a folder on the hard drive for each data folder in the experiment, and save each wave in a separate file, and all the strings and variables in separate files. Huge pain in the ass.
Packed template, this is what the MFP3D Template.pxt is. It is the same deal as a packed experiment, but when you go to save the experiment you have to work to save over it.

UnPacked template, same deal as Unpacked experiment, lots of files, same deal as a packed template, you have to work to write over your experiment when you load it.
So go ahead and go to file save experiment, this will bring up the save as dialog (since you opened a packed template to start this whole thing up). Make sure the file type is Packed Experiment, and write the whole Igor workspace to the harddrive. Now you can kill Igor and get on with your life, because you know that you will be able to open it back up later.

Loading your work

Most direct method is to double click on the saved experiment you want to open. But if this file is buried it might be faster to open up Igor (either the bare Igor, or the MFP3D Template.pxt), and then go through the File -> Recent Experiments menu to select the experiment you recently saved. Yeah, you are back where you left off.

Creating a User Folder

So one of the things that we have incorporated into the AR software are user folders. These are designed to allow you to save your preferences as to scan settings, and the multitude of parameters that run the MFP3D. The first step is to set up all the parameters the way you want them. Then go to the programming menu and select Set User Path. Then you will get a dialog to select a pre-existing user, the last option is to create a new user, select that and click continue. This closes all the windows that are open and saved all the parameters in a new folder on the harddrive
..\Program Files\WaveMetrics\Igor Pro Folder\AsylumResearch\Code3D\UserParms\"whatever your User name was"
Now you can restart the MFP3D Template.pxt (See loading the template), and then go back to the Programming menu and select set user path. Select the user you just created. Wow, everything is set up just the way you want it. Now if you want to overwrite a user path to incorporate a different setup, right now there is no interface for it, but it is still possible. Load up your user path just like normal. Change it to your new likings. Then type into the command line:

```
  SaveParms()
```

This is the function that saves all the parameters to the user path (which you had set with you selected your user name). So now when you load up your user parameters it will be in the updated layout.

Notebooks

Notebooks are a great little invention by Wavemetrics. They are built in notepads that can have graphics. So you can open up a notebook, record experimental parameters and put in some interesting data, basically an electronic lab notebook.
Step 1) open up a new notebook
Go to the Igor menu Windows -> New -> Notebook
Give it a new and go with formatted text
Plain text is old school, and who wants that?
OK, so you have your notebook open

There are a couple tricks to notebooks that are not easy to find in the Igor help.

Tricks
1), Graphs can be copied and pasted into notebooks. Just make the graph topmost and hit Ctrl + C, and go to your notebook and hit Ctrl +V and bam, your graph is in your notebook. These graphs in the notebook are now in some graphic format, such as png or the like, so they do not update when the data on the graph updates.

2) Commands can be executed from a notebook
if you select a set of code that is written in a notebook and right click it you have an option of execute selection. The short cut for that is Ctrl + Enter. The commands are executed just as if they were typed into the command window, with one nice little bonus. All the print commands are printed out in the

notebook. For example I am writing this help file in a Igor Notebook, so I type in
```
   Print DateTime
```
select it and hit Ctrl + Enter
and
3.14621e+09
is printed out just below the line of code.
So a nice way to timestamp your electronic lab notebook:
```
   Print Date()
```
Fri, Sep 12, 2003
```
   Print Time()
```
11:07:56 AM
```
   Print Date(),Time()
```
Fri, Sep 12, 2003 11:08:06 AM
And now you know exactly when I was typing this help file up.

Save your notebook:
When your notebook is topmost you have some extra menu items in the File menu. Save Notebook, SaveNotebook as, Save Notebook Copy. All the standard file saving options you have come to expect. There are a couple formats you can save ifn (Igor formatted text), being the one you want it you want to bring it back into Igor and edit it again. There is also RTF (rich text format), which allows you to open it in wordpad or a standard word processor.

To load a Saved notebook
Go to the file -> open file -> notebook menu, and hunt down your saved notebook. It is also probably listed in the File -> recent Files menu.

Dependencies

So we have mentioned dependencies before, and there is a nice example of them in the guided tour which you have gone through. So how do you set up a dependency to calculate the DeflectionVolts from the deflection and the Invols?

First off, let's do a force pull so we have some data to look at.

Second, where the hell is the InVOLS?
Go to The Programming menu, select global variable and master.
The Invols is one of the parameters in the table. This wave is in the root:Packages:MFP3D:Main:Variables: folder, in the MasterVariablesWave with the row label Invols and the column Label Value
i.e.
```
   print root:Packages:MFP3D:Main:Variables:MasterVariablesWave[%Invols][%Value]
```
9.5e-08

OK, we know where to get the InVOLS.
Now, let's make our DeflectionVolts wave in the root:packages:MFP3D:Force: folder
so go to that folder
```
   Duplicate/O Deflection DeflectionVolts
```
and give it the right data scale.
```
   SetScale d,0,0,"V",DeflectionVolts
```

The plot your new wave.
```
   Display/K=1 DeflectionVolts vs LVDT
```

Now we are set to make our dependency
```
   DeflectionVolts := Deflection/root:Packages:MFP3D:Main:Variables:MasterVariablesWave[%Invols][%Va
```
Now DeflectionVolts will update with the deflection wave. The problem with this dependency is that it will also update the wave every time the MasterVariablesWave updates, which is wasting a bit of

calculation time, but if you can live with that it is the easiest way. Another way is to make a global variable for the InVOLS, and every time the InVOLS updates to manually update the global variable, which defeats the purpose of the dependency somewhat. But since the deflection wave will probably update much more often than the Invols it would not be too bad.

BUT, an important thing to keep in mind is that the dependency does NOT resize your wave. So if Deflection changes size, such as you scan faster, then your deflectionVolt wave will have to be resized as well.

• Getting Started CurveFit tour

If you have not down it yet, you need to go through the Guided tour of Igor, Chapter 2, Analysis Tour #1 and #2. This section builds on the concepts you learn there.

Sub Toc

CurveFit Dialog

So you have gone through the Guided tour of Igor, Analysis tour #1 and #2, so you know about the curve fit Dialog window. You know what, that guided tour kicks so much ass, there really is no point in me trying to tell you more about it. let's just find a wave to fit in the MFP3D....

So do a force pull saving Deflection
, and go to root:packages:MFP3D:Force:
Now plot Deflection vs LVDT, no from the command line.

To get things rolling, fit the entire curve to a line.

So go to Analysis, Curve fitting...
Select line fit type
Select Ydata of deflection
Select Xdata of LVDT
Click DoIt

And you should have a line put on your graph of fit_deflection that is the best fit line to your deflection data.

Fitting Data Subsets

But you only want to fit a given region of the curve. So bring up your cursors and mark out some region of the curve to fit. The tricky part of this is that the deflection data wraps around, it starts at negative values, goes to higher values, and then goes back to where it started. So make sure both your cursors are on the same section of the trace. One way is to push the left or right key when both cursors are up there, if hey both go the same direction, then they are on the same section. If they move in opposite direction, then they are on different section.

OK
Go back to the analysis -> Curve Fitting... menu
And like in the guided tour, set the range in the data options tab to the cursors.

Click do it.

Extrapolating your fit

So you want to subtract this line from your entire data curve.

First set is to make your fit wave the same size as the data

```
  Redimension/N=(Numpnts(Deflection)) Fit_Deflection
```

Then find the line that says
fit_Deflection= W_coef[0]+W_coef[1]*x
Highlight that hit return
Change the x to LVDT

```
  fit_Deflection= W_coef[0]+W_coef[1]*LVDT
```

Now this looks like crap on the graph so remove the fit_deflection
and then put it back on, but with LVDT as the X data.
So now you can simple subtract out the fit_deflection from your deflection data.\
Deflection -= fit_deflection
Flat and zeroed, nice!

- **MFP3D Main Panel**

Scan Size

This is the size of the scan. The maximum size that this can be varies from machine to machine. This has a smaller maximum if the Scan Angle is not 0 or 90. It is also affected by the X and Y Offset, the more offset that you have the smaller the maximum scan is.

Scan Rate

This is the rate of the scan in hertz. This is a more important factor than the Scan Speed when scanning. As you scan a larger area the scan speed goes up with the same scan rate, but the resolution is also going down so that almost cancels out. At some point you do have to scan slower on bigger scans. The XY scanner has limitations that mostly relate to the scan rate and not the scan speed.

Scan Speed

This is the speed that the tip is moving across the surface. It is actually moving a little faster than it says, the software will probably reflect that some day. As mentioned under Scan Rate, that is actually the more useful way to keep track of how fast you can scan.

X Offset

This is the offset from the center of the scan range in X, or horizontal direction. This number is limited by the Scan Size and Scan Angle. As you scan larger the maximum offset gets smaller. If the scan angle is something other than 0 degrees, then figuring out exactly what direction X is gets harder. You can right click on a realtime graph, and set the X offset graphically. This will center the scan at the where you clicked.

Y Offset

This is the same as the X Offset except in the vertical direction.

Scan Angle

This is the angle of the scan. On the screen the fast direction of the scan is always horizontal irregardless of the scan angle. As you change the scan angle the topography that shows up an the screen will rotate. As the scan angle increases to 45 the maximum Scan Size decreases. If the scan size is at its max changing the scan angle can decrease the scan size automatically.

Scan Points

This is the number of points taken during one scan. The lower limit at this time is 32, but I would not use anything below 256. The maximum is not really firm, but I would not run more than 1024. The number of points have to be a multiple of 32. At this point it would be a good idea to stick to powers of 2, so that leaves the recommended numbers of 256, 512, and 1024.

Scan Lines

This is the number of scans taken during an image. The lower limit is 32 and the upper limit at this time is 512. 128 is a reasonable number for the recommended lower limit. The scan lines has to be a multiple of 8, but for now I would stick to powers of 2.

The ratio of points to lines can be whatever at the moment, but the scans will be square. This too will change at some point.

Set Point

This is what the feedback tries to hold the signal coming from the cantilever to. If the Imaging Mode is set to Contact, then this is done on the deflection of the cantilever. If the setting is NonContact, then the feedback system tries to hold the amplitude of the cantilever to the setpoint. This number is different depending on which mode you are in. If you are in NonContact then the setpoint can't be below 0.

Integral Gain

This controls how fast the feedback system reacts to the error signal. If this number is too low then the surface will not be tracked faithfully, if it is too high then the feedback loop will oscillate. It is usually best to run just below where it oscillates. At the present time useful gains are in the range of .2 to 20.

Proportional Gain

Since this is proportional gain it doesn't really matter too much what it is set at. If it is high enough it can make the feedback loop oscillate, but having it at 0 usually doesn't make much of a difference in tracking.

Secret Gain

This is actually double integral gain. It is mainly here for testing purpose. If I was trying to actually image something, I sure wouldn't turn it on.

Drive Amplitude

This is the amplitude of the drive signal that goes to the piezo that drives the cantilever. It is in volts and doesn't have any real connection with the amount of signal that comes out of the optical detection of the cantilever, other than as the drive amplitude goes up so should the signal coming out.

During Contact this is set to 0.

Drive Frequency

This is the frequency that the cantilever piezo is driven at.

Slow Scan Enabled

This is not hooked up yet. But I have a dream that it will be, someday.

Clear Image

This is in a state of transition. It used to fill waves with NaNs, but it now fills the wave with a magiacally generated number that works better.

Imaging Mode

This is mode that is currently being used. At the moment, there are only two modes, Contact and NonContact. In contact the deflection of the cantilever is used as the orror signal. In NonContact the cantilever is driven so that it oscillates and the amplitude is used as the error signal.

Do Scan

This starts the instrument scanning. Once it does this the button changes to Last Scan.

Last Scan

This is what the Do Scan button changes to once you start scanning. If you then hit this it will change to "Waiting...". Once it is in that state as soon as it finishes the next scan it will stop scanning and just sit there. If you like this button I would use it now as it might disappear in future versions of the software.

Stop!!!

This stops all input and output, turn off all feedback loops and retracts the piezo to 0 volts.

Frame Up

Starts a scan at the top of the image.

Frame Down

Starts a scan at the bottom of the image.

Image Name

This is the name that images are saved with, with the image suffix added on the end. At the moment this should start with a letter instead of a number.

Image Suffix

This is a four digit number that is attached to the image name when images are saved. It can't be negative.

Save Images

If this box is clicked then an image will be saved whenever a scan finishes without changes that would make the saved image weird. For example, changing the scan size halfway through would keep that image from being saved.

Save Image

This will save the next good image, but just the next one. If this is hit when the Save Status is displaying Save Next then it will force a save and the save status will change to Save Anyway. Whenever save status indicates a save will happen and you don't want to save this image, if you hit this button a couple of times it will change to Don't Save and that will stop the save if hit.

Don't Save

The Save Image button changes to this. If clicked it keeps the next image from being saved.

Save Status

This shows the current save status.
None that there is no save action at all
Save Current means that as soon as this image is through it will be saved. This will change to Save Next if certain parameters change.
Save Next means that the image after this one will be saved. A parameter changed that prevents the current image from being saved.
Save Anyway means that a parameter was changed in the current image, but then the Save Image button was hit so this image will be saved.

Show Box

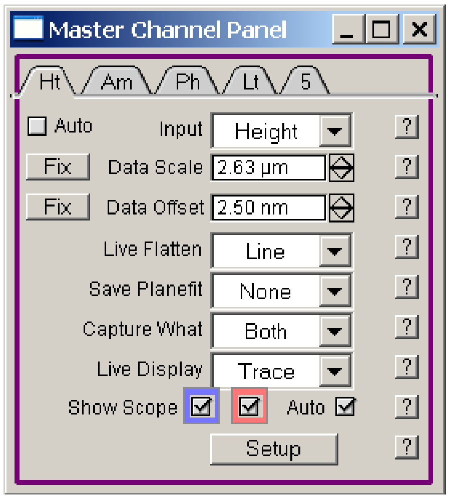
This adds an area at the bottom of the panel that shows the box that is being scanned and its relationship with the total available scan area.

Setup

This shows the same panel but with checkboxes which control which controls are shown.

- **MFP3D Channel Panel**

Sub Toc



Auto ☐ Input Height
Data Scale 2.63 µm
Data Offset 2.50 nm
Live Flatten Line
Save Planefit None
Capture What Both
Live Display Trace
Show Scope ☑ ☑ Auto ☑

Master Channel Panel Tab

This controls which of the 5 channels that you are dealing with. If a channel is not being used, then its number will be showing. If it is in use then two letters that describe the data type will show. See Data Type below for the list of data types and abbreviations.

Auto Scale

This sets the image scaling to automatic. With this on you can usually see something, but if there are big spikes in the image then the part of the image that is interesting can have contrast that is too washed out to be useful. If you use any of the other range scaling tools then this is turned off.

Data Type

Use this to select your input data type. All of the data types now work. Height or ZSensor are always on so you should pick one of them since it will be captured anyway. Lateral can only be captured if you have a production head. There are limits on how many channels can actually be captured at once, but they are complicated. If a channel can not be captured this is printed in the history after the scan is started.

Data Type	**Abbr.**	**Description**
Height	Ht	The Z voltage output scaled by the piezo sensitivity.
Amplitude	Am	The AC amplitude of the signal from the cantilever.
Deflection	Df	The DC deflection of the signal from the cantilever.
Phase	Ph	The Phase lag between the drive to the cantilever and the signal from the cantilever.
ZSensor	ZS	The height as measured by the sensor on the Z axis.
UserIn0	U0	The signal from the UserIn0 BNC on the front panel.
UserIn1	U1	The signal from the UserIn1 BNC on the front panel.
UserIn2	U2	The signal from the UserIn2 BNC on the front panel.
Inputi	Ii	This is the Amplitude and Phase plotted in rectangular coordinates
Inputq	Iq	The other rectangular coordinate.
Lateral	Lt	The signal from the cantilever that measures how much it is twisting.

Fix Scale and Offset

This is the Fix button next to Data Scale. It calculates what it thinks are reasonable values for the Data Scale and the Data Offset. The Data Offset it calculates is just the average value of the image. For the Data Scale it uses twice the difference of closest extreme from the average value. For example, suppose you have an image with an average of 2, a low of 1 and a high of 5. It would set the offset to 2, and the range to 2*(2-1) or 2. This means that everything that was 3 or above would be white, but with most AFM images there are points that would ruin the contrast of what you really want to see if try to accommodate them.

First Data Scale

The First Data scale is for the trace image, if both directions are displayed. if only one is displayed then this is the scale for that even if it is retrace. See Second Data Scale for the Retrace Image. This sets the range of the colormap. You have 4 options for this. Hit the Auto CheckBox to the upper left of the control, this will let Igor try and auto scale the image for you (often does a poor job, but great for auto updates). Then there is the fix button just to the left (see Fix Scale and Offset). This will calculate a decent range and offset for the scale. The third option is to manually enter a value in the setvar. The last option is to drag a rectangle on the image over the region of the image that has the feature you most care about, right click inside the rectangle, and select FixScale. This will calculate the best scale based on the data in that region. When changes are made to the data scale they will also affect the range on the scope trace, unless the auto checkbox is selected (which is to the right of the Show Scope checkboxes).

Fix Offset

This button will fix the colorMap offset to the mean of the image. It will also set the mid point on the scope trace, unless the auto checkbox is selected (which is to the right of the Show Scope checkboxes).

First Data Offset

The First Data offset is for the trace image, if both directions are displayed. if only one is displayed then this is the scale for that even if it is retrace. See Second Data Offset for the Retrace Image. This sets the offset of the colormap. You have 5 options for this. Hit the Auto CheckBox to the upper left of the control, this will let Igor try and auto scale the image for you (often does a poor job, but great for auto updates). Then there is the fix button Up and to the left (see Fix Scale and Offset). This will calculate a decent range and offset for the scale. The third option is the hit the Fix Offset button, which is just to the left of this setvar. The fourth option is to manually enter a value in the setvar. The last option is to drag a rectangle on the image over the region of the image that has the feature you most care about, right click inside the rectangle, and select FixScale. This will calculate the best scale based on the data in that region. When changes are made to the data scale they will also affect the range on the scope trace, unless the auto checkbox is selected (which is to the right of the Show Scope checkboxes).

Live Flatten

This popup has 4 options.
This will modify the displayed data, but will in no way alter the data saved to the disk.
None, leaves you with your raw data.
Offset, takes the average value of the scan line, and subtracts that from the data.
Line, Fits each scan line to a straight line, subtracts this line out of the scan line.
Plane, Not fully implemented just yet, only does a line for now.

Save Planefit

This popup has 3 options.
This will modify the Saved data.
None, leaves you with your raw data.
Offset, takes the average value of the Image, and subtracts that from the data.
Plane, subtracts a first order XY Plane from the Image.

These parameters are stored in the wave note.
They are stored according to the layer of the data in the matrix (see some help file that has not been written yet as to how the files are stored or just call us up 1-805-685-7077).

So you have a wave called Image0, that has 3 layers, Height = 0, Amplitude = 1, and Phase = 2
All of them Trace Direction.

EX:
```
String NoteString = Note(Image0)                    //pulls out the wave note.
Variable Layer = 0                                  //lets get the height
String LayerString = GetDimLabel(Image0,2,Layer)          //this contains the string HeightTrace
String PlaneFitType = StringByKey("Planefit "+num2str(Layer), NoteString,":",num2char(13))
//contains string of Save planefit type
Variable Offset = NumberByKey("Planefit Offset "+num2str(Layer),NoteString,":",num2char(13))
Variable Xslope, Yslope
if (StringMatch(PlaneFitType,"Full"))               //we did a full plane fit
      Xslope = NumberByKey("Planefit X Slope "+num2str(Layer),NoteString,":",num2char(13))
      Yslope  = NumberByKey("Planefit Y Slope "+num2str(Layer),NoteString,":",num2char(13))
endif
```

then if layer = 1 you would end up with the amplitude info (for this example).

Capture What

This popup lets you capture Trace, Retrace or Both.

Live Display

This popup lets you display the image you are collecting (see Capture What). If you are collecting trace, it puts that image up, if you are collecting retrace, it puts that image up, if you are collecting Both, you have the option of putting up one of them or both.

Show Scope

The popup underneath the blue background with put on the Trace Scope (blue line on the scope axes). The checkbox underneath the red background is the Retrace Scope (red line on the scope axes). The scope is the current scan line being collected. Now what you need to understand is that this is updated about once a second. So if you are scanning at 2 Hz, you are only seeing about every other scan line. The auto check box to the right of these 2 checkboxes auto scales the scope axes for you.

Second Data Scale

These controls are typically hidden from you. You need to go into the setup (for each individual channel), and set the show controls to display the second data scale setvars. These controls will adjust the scale for the Retrace Image if both images are displayed. They work in the same manner as the First Data Scale.

Second Data Offset

These controls are typically hidden from you. You need to go into the setup (for each individual channel), and set the show controls to display the second data offset setvars. These controls will adjust the offset for the Retrace Image if both images are displayed. They work in the same manner as the First Data Offset.

Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

• MFP3D Thermal Panel

Thermal DC

This is one of four parameters that control what the Simple Harmonic Oscillator (SHO) fit looks like. It is how much the cantilever moves at DC from the drive that is applied to it. In this case the drive is air or fluid (I know, air is a fluid) molecules hitting it in a random fashion

Thermal Q

This is the Quality of the SHO peak. If you divide the resonant frequnency by the width of the peak at 1/2 amplitude you get the Q.

Thermal Frequency

This is the resonant frequency of the cantilever. This is actually higher than the frequency where the cantilever resonates with the greatest amplitude, but the difference is only noticeable with a low Q.

Thermal White Noise

This is the white noise that is built into the whole detection system. If the cantilever is not moving this is the amount that it will appear to be moving because of noise in the detection system. This parameter does not affect the spring constant calculation, but it helps the fit especially at higher frequencies.

Fit Width

This is the width around the Frequency that is used when trying to fit the Thermal PSD with the SHO fit. Note that it is the value of the Frequency at the start of the fit attempt, not the value of Frequency at the end. This width is also used by the Fit Guess button.

InvOLS

This stands for Inverse Optical Lever Sensitivity. We express this as nanometers/Volts as then it works out that it is usually in the range of 25-200 nm/V. If it was expressed as Volts/nanometer it would be .04-.005 V/nm. This is just harder to deal with as a human. We call it Inverse because as the system gets more sensitive the number gets smaller.

Spring Constant

This is calculated when you hit the Try Fit button. It is calculated from the Thermal DC, Thermal Q, and Thermal Frequency after a successful fitting of the SHO equation.

Fit Guess

This guesses at what the SHO parameters should be based on what the Thermal PSD looks like centered around the Zoom Center within the Fit Width. If the Show fit box is not already checked it also does that.

Try Fit

Trys to fit the thermal PSD with the SHO fit using the parameters above. Calculates the spring constant after fitting.

Show fit

Shows the SHO fit.

Show Thermal

Shows the Thermal PSD. If this has not been done yet then the graph shows up wihthout anything interesting on it.

Graph Log

This sets whether the axes are set to log or linear. The default is Log/Log, which is usually the only way to look at the full range and actually see something.

Zoom Graph

This zooms into the region of interest centered around the Zoom Center using the width of Zoom Width. When this is done the left axis is not autoscaled, and it updates only every five data captures, so it looks strange at the start of a run.

Zoom Center

This is the center if you have the Zoom Graph checkbox checked. It is also the drive frequency.

Zoom Width

The width of the thermal graph if you have Zoom Graph checked.

Max Samples

This is the maximum samples that will be averaged together to make the thermal PSD. This number may as well be big unless you are looking at something that is changing quickly.

Current Samples

How many samples have been taken currently.

Thermal Resolution

How many points are captured. Better resolution is needed for the higher frequency cantilevers. The lower the relolution the quicker the noise settles out.

Do Thermal

This starts capturing thermal data and then changes to Stop Thermal. If the Stop Thermal button has just been hit then this button doesn't do anything for a couple of seconds.

Stop Thermal

This stops capturing thermal data. It can be hard to get to actually work. There is a timer so that if you hit Stop Thermal and get it to work you have two seconds to stop clicking on Do Thermal so it will not start again. Also closing the Thermal Graph window will stop capturing.

Setup

This allows you to adjust the controls that show up on the panel.

- **MFP3D Master Litho Panel**

These panels allows the user to Draw, Save and Import a path for the tip to follow.

TOC

Lithography Glossary

Some of the basic concepts of the Asylum Lithography interface that are most often confused by new users.

Litho Groups: One or more Litho Sections. This collections is often treated as 1 entity from the viewpoint of the user interface. However, the individual parts are treated in the same manner by the controller as the Litho Sections.

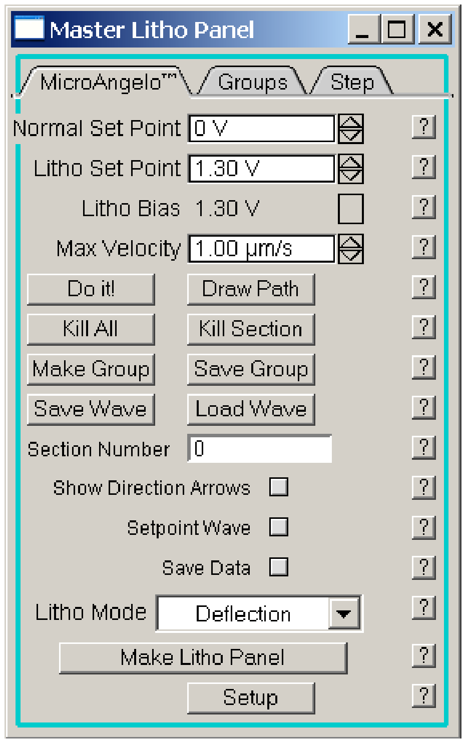
Litho Sections: A continuous path describing the XY co-ordinates the tip is to follow.

Edit mode: When you first draw a path or display a group, the line(s) will be shown in red. You can edit then, by either moving individual points, or moving the group around. When in this mode the line(s) are not recognized by the Lithography feedback. You need to set them, if in draw mode, you need to click on the image again to set it as a section, or if you are working on a group you need to click the add group button. This will change your line(s) to blue, and they are locked in for the feedback.

- **Litho Panel**

Main Litho Panel, used to draw freehand a path for the tip.

Sub Toc



Normal Set Point 0 V
Litho Set Point 1.30 V
Litho Bias 1.30 V
Max Velocity 1.00 µm/s
Section Number 0
Show Direction Arrows ☐
Setpoint Wave ☐
Save Data ☐
Litho Mode Deflection

Setpoint

This is the setpoint used when the tip moves between the litho Sections. It depends on the litho mode you are working in. When in Deflection mode the tip is working in contact mode, so this setvar is the Deflection Setpoint Volts. While in Contact Voltage mode, again, the tip is working in contact mode. However, when in AC voltage mode, the tip is working in intermittent contact mode, so that the Setpoint is the Amplitude Setpoint Volts.

Litho Setpoint

This is the setpoint that tip will feedback at when drawing the sections. It is only used in the Deflection Litho mode (so the tip is working in contact mode). When in other modes the Litho Bias is used.

Litho Bias

This is the bias applied to the tip when drawing the individual litho sections when working in the voltage litho modes. This is only available on production model heads, the beta heads can not handle the new tip holders to apply the voltage.

Litho Max Velocity

This is a rough guide as to how fast you want the tip to move when drawing the Lithography sections. It looks at the largest separation between any 2 points in a given section, and then from that calculates the constant time spacing between the points. So if you draw something with inconsistent X-Y point spacing, when the tip gets to the big gaps, it will move at the specified velocity. At the smaller gaps it will move at a correspondingly slower velocity.

Do Lithography

This button does just that, once you have your litho sections set up, you have chosen your Litho Mode,

and double checked your Setpoint, and Litho Setpoint or Litho Bias (depending on Litho Mode), then you are all set. Click on this and watch the little red dot follow your path. The red dot is getting it's data straight from the LVDTs on X and Y, and is a background task. Stay away from the meter panel while doing lithography, bad things can happen to the Background task. If the background task is messed up, it will continue to do it's work, there is just nothing to watch.
If you really want the background task back, the following 3 lines called from the command line should get the background task running again...

```
        KillBackground
        SetBackground LithoBackground()
        CtrlBackground start,period = 1, noBurst=1
```

While working, this button will tell you which section it is working on (clicking on the button does nothing just yet). The button Draw Path will change to let you stop the work early.

Draw Path

This button will allow you to draw on the RealTime Images to define a path for your tip to follow. Click on the button, one of the realtime images will be brought forward, you can then draw your path. It then goes into some Igor edit mode. You can drag the individual points around. This is generally bad, if you drag the points so that it is making very sharp turns with only a few points to do it, the feedback will not be able to accurately follow your path. But anyway, after you have finished your editing of points, just click on the image somewhere else, the red line you just drew turns blue, and becomes a section. You can get rid of that section by clicking Kill Section. When you are all done with your paths, then click on this button (which will be called Stop Draw at that point).

Kill All

This button will erase all the sections and groups that you have up on your images.

Kill Section

This will erase the last drawn section or all the sections of the last group you have drawn. The title of the button will change to reflect what action it will perform.

Make Group

This converts the last batch of sections into a group. This can be a little confusing. Say you draw 2 sections, make that to a group, then draw 3 more sections, and click on the make group button again. You now have 2 groups. Clicking on save group will make a group if the last batch is not yet a group. So you can have sections and groups together, but all the sections have to be at the end. Right now there is no way to go back from a group to a collection of sections.

Save Group

Saves a copy of the group in memory. This way you can take your freehand groups to the Litho Group Panel, and replicate them if you want. Will call make group if the last drawn paths is a section.

Save Wave

Saves the last group to the harddrive, so that you can load them later into a different experiment. If the last path drawn is a section it will call Make group.

Load Wave

Opens a dialog to load your previously saved waved waves, see Save Wave.

Litho Number

This is some guild as to how many groups and sections you have up on the images. It is mostly used for debuggin. If you are in draw mode it will show you 1 more than then number of groups and sections up (the section number of the next drawn section).

Show Direction Arrows

If this checkbox is selected arrow markers will be placed on the Blue litho sections pointing in the direction the writing will be done in.

Use Setpoint Wave

This checkbox will bring up a graph with some controls on it. This will generate a wave to be used as the setpoint. There are 2 modes, linear and Staircase. Another note, this panel has no concept of the groups, and so it will treat a group as the individual sections.

Linear will ramp from Start Voltage to EndVoltage. Here voltage depends on Litho Mode, can be the deflection setpoint volts (Deflection mode), or can be the bias applied to the tip (voltage modes). If the Full Slope checkbox is selected, then the ramp will cover all the sections on the image. So if you don't have the full slope checkbox selected, then it will ramp from start voltage to end voltage for each section.

Staircase will take the StartVoltage and increments the voltage by Voltage Step, for each subsequent section.

The X axis of the plot is the point number of the total collection of sections. The Segment number setvar will zoom in on the selected section. Setting it to 0 will show all the sections.

Litho Save Wave

Duplicates the deflection and Zsensor (ZLVDT) collected during the Lithography. These are stored in the root:SavedLitho: folder, which you can get to easy from the Data Folders menu. They are saved based on the BaseName.

Litho Mode

Feedback mode of the lithography. You have 3 options.
Deflection: Works in contact mode, the tip will maintain Deflection Setpoint volts between sections as it travels across the surface, and then when it is drawing the section will maintain Litho Setpoint volts in contact mode.
Contact voltage: Works in contact mode, the tip will always maintain Deflection Setpoint volts as it travels across the surface as well as drawing the sections. Additionally, when the tip is drawing the sections, Litho bias volts will be applied to the tip (if you have a production model head and a conductive tip).
AC voltage: This is NOT alternating current, it is applying a DC bias to the tip while in intermittent (AC) feedback. The tip works in AC mode, always using the Amplitude Setpoint Volts as the setpoint, and when drawing the individual sections, it will apply Litho Bias volts to the tip (if you have a production model head and a conductive tip).

See also Setpoint, Litho Setpoint and Litho Bias.

Make Other Litho Panel

This button makes the Litho panel or the Master Litho Panel, depending on which panel it is called from.

Litho Panel Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

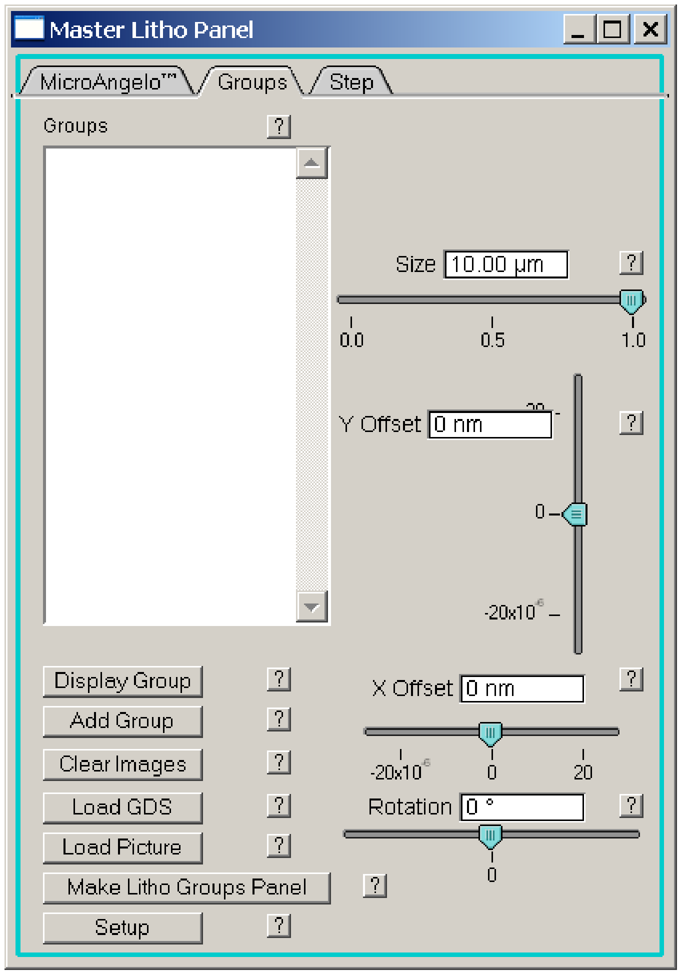
1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- **Litho Group Panel**

Panel to warp imported Lithography groups.

Sub Toc



Group List

This list box lists the available groups you have. To display a group, select it from the listbox and then click the Display group button.

Litho Size

This slider and setvar set will make your displayed group larger or smaller.

Litho Y Offset

This slider and setvar set will move your displayed group left and right.

Litho X Offset

This slider and setvar set will move your displayed group up and down.

Display Group

This places the currently selected group from the Group List in edit mode on the topmost real time image.

Append Group

This button sets the currently editing group. See Edit mode in the lithograph Glossary above. If you do not have any group in edit mode, clicking this button is the same as clicking display group, and then Append Group.

Clear Images

This button removes the litho groups that are in edit mode from the real time images.

Litho Angle

This slider and setvar set will rotate your displayed group clockwise and counter clock wise.

Load GDS

This button will ask you for a GDS II file. It will then try to read that file using simple converters (not very extensively tested). Complicated things in GDS format are not fully supported, but it will load all the X-Y data sets and sort them according to the specified layer. It will then convert that to Asylums litho format, and make sure that the point spacing does not change more than a factor of 2 within a given section. After that it will ask you if you want to clean up the individual waves. If you loaded a big file, with a couple thousand sections, igor can get pretty slow (first off, kill the data browser). These waves are not needed by the litho interface anymore, and so they should be killed to make things faster. But if you really wanted them around, we give you that option. They have to be killed if you load another GDS file. If you want this button to do more, you need to call or email asylum and let us know.

Load Picture

This will load up a picture (tiffs are the most tested), and then convert them into a set of contours, which can then be taken into the litho group panel.

Make Other Litho group Panel

This button makes the Litho Group panel or the Master Litho Panel, depending on which panel it is called from.

Litho Group Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

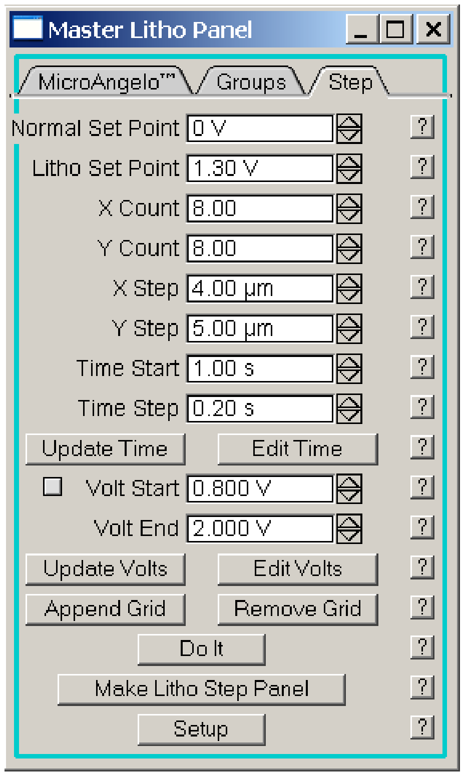
1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- ## Litho Step Panel

This is a new panel (more bugs then usual). It will let you build lithograph grids to ram your tip into the surface in lots of places.

Sub Toc



Normal Set Point 0 V
Litho Set Point 1.30 V
X Count 8.00
Y Count 8.00
X Step 4.00 µm
Y Step 5.00 µm
Time Start 1.00 s
Time Step 0.20 s
Update Time | Edit Time
☐ Volt Start 0.800 V
Volt End 2.000 V
Update Volts | Edit Volts
Append Grid | Remove Grid
Do It
Make Litho Step Panel
Setup

Setpoint

This is the setpoint used when the tip moves between the litho Sections. Right now Step litho only works in contact feedback, and so this setvar is the same as the Deflection Setpoint Volts setvar from the main panel.

Litho Setpoint

This is the deflection setpoint that tip will feedback at when drawing the points.

LithoXCount
This is the number of grid points you want in the X axis.

LithoYCount
This is the number of grid points you want in the Y axis.

Litho Xstep
This is the step size between the grid points in the X axis. The grid will be centered in the middle of the image.

Litho Ystep
This is the step size between the grid points in the Y axis. The grid will be centered in the middle of the image.

LithoTimeStart
The time wave is how long the tip will remain in contact with the surface at each point. It will make a wave for this based on the start time (how long the tip is in contact with point 1), and then increments that time by LithoTimeStep, for each of the subsequent points. Point 0 is the lower left corner.

See also Update Time and Edit Time.

LithoTimeStep
This is how much time spent with the tip in contact with the surface increases between points. See LithoTimeStart above.

Update Time
This will redo the calculation of the time for you. The code does not auto update this for you since you can edit the values yourself. See also LithoTimeStart.

Edit Time
Brings up a table of the times to spend on the surface at each grid point. See also LithoTimeStart.

Use Voltage Wave
If this checkbox is selected then the LithoSetpoint is ignored. The setpoint for the deflection is based on the wave calculated from the LithoStartVolts and LithoEndVolts. This is very similar to the Use Setpoint Wave from the LithoPanel. Point 0 is the lower left corner.

LithoStartVolts
This defines the start of the setpoint ramp, see Use Voltage Wave above.

LithoEndVolts
This defines the end of the setpoint ramp, see Use Voltage Wave above.

Update Volts

This will redo the calculation of the setpoint for you. The code does not auto update this for you since you can edit the values yourself. See also Use Voltage wave.

Edit Volts

Brings up a table of the setpoints for each grid point. See also Use Voltage Wave.

Append Grid

Puts your grid of points on the topmost realtime Image.

Remove Grid

Removes the grid of points on the topmost realtime image.

Do Step Litho

Just what the button describes, this starts the feedback and the tip moves to each point in the grid, waiting there the specified time from the Time wave (See LithoTimeStart), and using either the wave or the single value of the setpoint (see Use Voltage Wave). You can follow along the progress of the controller with bongo the bouncing red dot of joy. This has the same types of problems with the background as Do Lithography has from the LithoPanel.

Make Other Litho Step Panel

This button makes the Litho Step panel or the Master Litho Panel, depending on which panel it is called from.

Litho Step Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- **MFP3D Cursor Panel**

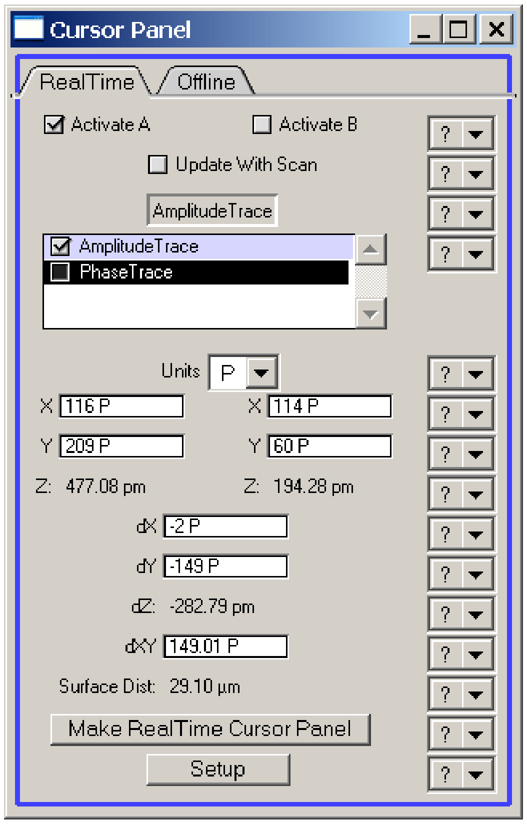
  This panel allows the user to manage the cursors on the RealTime and Offline Images. You can synchronize the cursors on multiple images, as well as pull out slices from the images and calculate various distances along the surface.

TOC

- **Real Time Cursor Panel**

  Panel to synchronize the cursors on the RealTime Images.

Sub Toc



Activate A
This "activates" the A cursor. This means that the A cursor will move with the arrow keys when the graph is topmost. If Both the A and B cursors are not active they will be removed from the graph. When entering values in dX, dY, or dXY the default cursor to move is the B cursor, but if the A cursor is NOT active, then the A cursor will be moved.

See also ShowInfo, Cursor.

Activate B
This "activates" the B cursor. This means that the B cursor will move with the arrow keys when the graph is topmost. If Both the A and B cursors are not active they will be removed from the graph.

See also ShowInfo, Cursor.

Update With Scan
When this checkbox is selected the profiles plotted from the Graph List will be updated, as well as the various distances when the Image is updated. This may make your machine run a fair amount slower, particularly with many 1K x 1K images. Killing both the RealTime Cursor panel and the Cursor Panel will set this to 0, so if you find yourself bogged down, try killing the panels.

Which Image
This Titlebox simply lists the title of the graph that the distances are calculated from.

Graph List

This Listbox shows the displayed RealTime Images. If the checkbox is selected for an image a profile will be pulled out of that image and displayed on a separate graph. The profile is the Image data between the 2 cursors, much like the section analysis tools in the Section Panel.

Units

This PopUpMenu will allow you to switch between distance (nm) and Point (P) units for the distance setvars.

X

These setvars show the X position of the A and B cursors. New values can be entered here in units determined by the Units Popup.

Y

These setvars show the Y position of the A and B cursors. New values can be entered here in units determined by the Units Popup.

Z

These setvars show the Z position of the A and B cursors. New values can NOT be entered here. The units are determined the topmost Image.

dX

This SetVar shows the difference in the X values between the 2 cursors. New values can be entered here in units determined by the Units Popup. If the A cursor is active (see Activate A), then the B cursor will be moved, conversely, if the A cursor is not active, the A cursor will be moved.

dY

This SetVar shows the difference in the Y values between the 2 cursors. New values can be entered here in units determined by the Units Popup. If the A cursor is active (see Activate A), then the B cursor will be moved, conversely, if the A cursor is not active, the A cursor will be moved.

dZ

This SetVar shows the difference in the Z values between the 2 cursors. New values can NOT be entered here. The units are determined the topmost Image.

dXY

This SetVar shows the distance in the XY plane between the 2 cursors. New values can be entered here in units determined by the Units Popup. If the A cursor is active (see Activate A), then the B cursor will be moved, conversely, if the A cursor is not active, the A cursor will be moved.

Surface Dist

This SetVar shows the distance traveled if you walked along the surface from point A to point B. This value is slow to calculate, so it is only done if you have selected to plot the profile (Graph List) of the topmost Image listed in the Which Image Titlebox (as that data is needed for the calculation). Also if the units of the topmost image are not in meters, it will not bother doing the work, since the value will have useless units.

RealCursor Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- **Offline Cursor Panel**

Panel to Synchronize the cursors on the Offline Images.
Coming soon to a version near you.

Sub Toc

- ## MFP3D Image Display

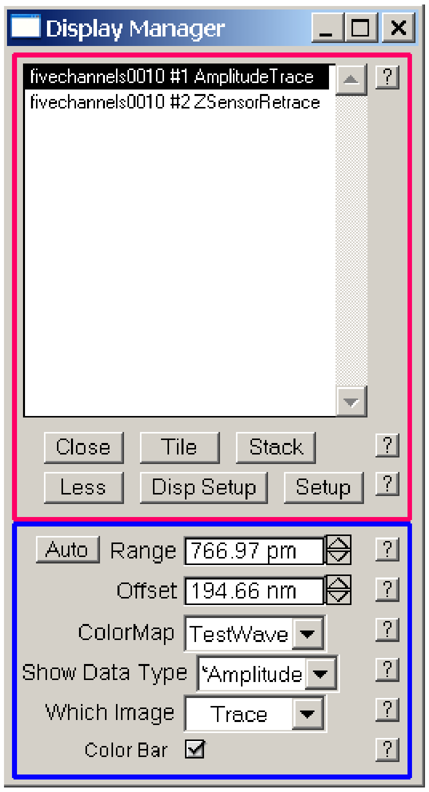
These Controls will allow you to Load and Display your Image Data.

TOC

- ## MFP3D Display Manager

Also referred to as the Display Panel (Historical reasons). This panel lists all of the Images that are being displayed.

TOC



fivechannels0010 #1 AmplitudeTrace
fivechannels0010 #2 ZSensorRetrace

Range 766.97 pm
Offset 194.66 nm
ColorMap TestWave
Show Data Type 'Amplitude
Which Image Trace
Color Bar ✓

From this panel you can manage your multitude of offline images. Changing display attributes of some or all of them as well as set up defaults for how they are opened.

Global Display List

This ListBox contains the list of the displayed images. Double click on one of them to bring to front. Select multiple Displays (shift click) to perform actions on the selected displays.

Global Display Close

The Close button closes all the graphs selected in the Global Display List.

Global Display Tile

The Tile button tiles all the graphs selected in the Global Display List.

Global Display Stack

The Stack button stacks all the graphs selected in the Global Display List.

Global Display More

The More / Less button Shows / Hides all the controls shown in the lower blue box.

Global Display Disp Setup

The Disp Setup button brings up a panel that lest you select which controls are going to be shown on the individual graphs. This will not update the existing graphs, only graphs that are opened subsequently.

Display Auto

The Auto Button will calculate a "nice" range and offset for all the graphs selected in the Global Display List. This will put the values in the wave notes as well.

Display Range

The Range SetVar will change the range for all the graphs selected in the Global Display List. This is only enabled if all the graphs have the same Data Type. This will put the value in the wave notes as well.

See also Display Data Type.

Display Offset

The Offset SetVar will change the Offset for all the graphs selected in the Global Display List. This is only enabled if all the graphs have the same Data Type. This will put the value in the wave notes as well.

See also Display Data Type.

Display Color Map

The ColorMap Popup menu will set the colormap for all the graphs selected in the Global Display List. This will put the value in the wave notes as well.

Display Data Type

The Show Data Type popup menu lists the available Data types based on the images selected in the Global Display List. The current value displayed in the popup does not necessarily correspond to the layer displayed in the selected images, because the selected value in the popup defines the default data type to display when opening an image. The data types that are common to all the selected images have a star (*) in front of them. If you select a channel that is NOT common to all selected images, the images that do not have that image will be set to a "Bogus" layer. What it is actually does is set the image to the last layer, re-titles the graph, and sets the scaling so that there is no contrast for the last layer.

Display Which

The Which Image popup menu lets you select if you want to see trace, or retrace images and works in conjunction with Display Data Type. If you select Either, it will try to put up trace first, then if it can't it will try to put up retrace.

Global Color Bar

The Color Bar checkBox sets up the default behavior of the graphs when they are opened. If it is selected the graphs will be opened with a color Bar. If this control is selected or unselected it will force the graphs selected in the Global Display List to have (or not have) a colorbar.

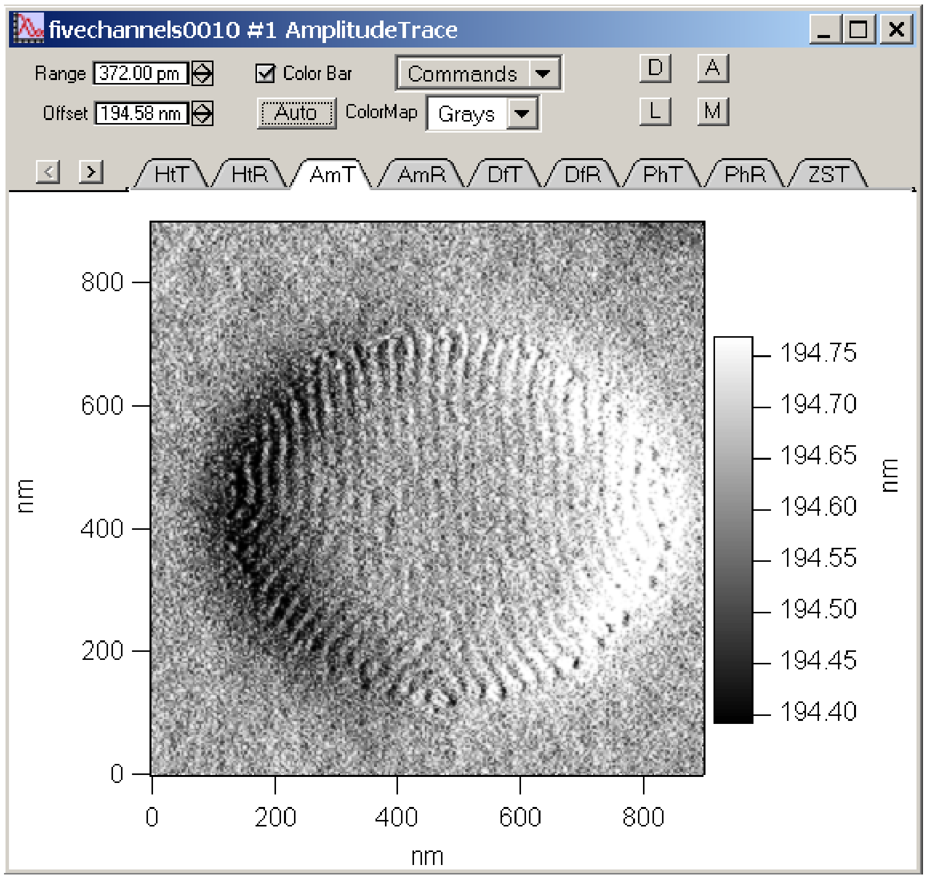
Border Color

The Display Manager has 2 colors for the border. One for the lower "Global Display More" region, and one for the upper controls.

• MFP3D Display Graph

Individual Image graphs, and the controls associated with them. These controls are referred to as local, as they affect only the individual image. The Controls, ColorBar and Axes labels will only appear if size permits. As you make the graph smaller these items will be removed. As you make the graph larger, they will be added back on.

TOC



Local Range

The Range SetVar will change the range for the graph. Changing the range alters the amount the clicker changes the range and offset. The range clicker is 10% of the range, and the Offset clicker is 5% of the range. This will put the value in the wave notes as well.

Local Offset

The Offset SetVar will change the Offset for the graph. This will put the value in the wave notes as well.
See also Local Range above.

Local Color Bar

The Color Bar checkBox puts a colorbar on the graph. Note that resizing the image can take the colorbar off if the plot gets too small, but resizing the graph to make it larger will put it back on.

Local Auto

The Auto Button will calculate a "nice" range and offset for the graph. This will put the values in the wave notes as well. See also Local Range and Local Offset.

Local Commands

The Command Popup menu has a list of useful commands:

Display Panel, Brings up the MFP3D Display Manager, see also Local "D".
Modify Panel, brings up the MFP3D Modification Panel, see also Local "M".
Analyze Panel, brings up the analyze panel, see also Local "A".
List Panel, bring up the list panel, see also Local "L".
Extract Layer, this will pull the currently displayed layer into the wave LayerData in the Root:Images folder, and put you in that folder. This Wave will be displayed in the List Panel, and can be viewed as a normal Image. You can then do custom work on this image, modifying / analyzing it as you see fit.
Advanced note: see ExtractLayerDF in temp2.ipf. Calling this function from the command line will extract the topmost image layer into the wave LayerData in the current data folder. This way you can change your data folder and have multiple LayerData Waves.
Inset Layer, This takes an extracted layer and puts it back into the Wave it came from. You can insert it overwriting the original layer, or as a new modified layer. When you insert it as a modified layer, you can overwrite one of the previously inserted modified layers (of the same data channel), or as a new Modified layer. When you insert the layer it will bring up a listbox where you can enter a note for that layer into the wave note for the image.
Advanced note: see InsertLayerDF in temp2.ipf. Calling this function from the command line will insert a LayerData wave from the Current Data Folder into the wave it came from. Using this you can have multiple LayerData Waves in multiple data folders. It automatically overwrites the original data layer.
Show Note, This brings up a listbox with the wave note for the Image. It is still in it infancy, and so there is not much to do with it right now. It will not update on it's own, if you change the note, you need to close this panel and open it back up. You can change the displayed note, but there is no way to put it back into the wave note just yet. There are some String finding controls to help you find parameters. You can double click on the right column to edit the text in there.
Advanced note: The text wave that is displayed in the listbox is stored in root:packages:MFP3D:Main:Display: folder, the wave is named from the Image Name with the extension "_HD" at the end of it.
Save Image, Saves the image to the hard drive.
Save then Kill image, does just that, saved to hard drive and then removes it from the memory.
Kill Image, removes the image from the memory.
Set as Surface Plot Data, takes a copy of the displayed layer (only the visible region, so you can zoom in). and sets it up for use with the Gizmo Interface. See also Surface Image.
Set as Color for surface plot, is similar to Set as Surface Plot data, but uses the shown data as the color data for the Gizmo interface. See also Color Image.
Setup, brings up another panel to set which controls are displayed on the graphs. See also Global Display Disp Setup.
Help, Brings up this document, but you already knew that didn't you.

Local Color Map

The ColorMap Popup menu will set the colormap for the graph. This will put the value in the wave notes as well.

Local "D"

Brings up the MFP3D Display Manager.

Local "A"

Brings up the Analyze Panel.

Local "L"

Brings up the MFP3D List Panel.

Local "M"

Brings up the MFP3D Modification Panel.

Local "<"

If you have more layers in your image than you have space on the graph to display, then you will not be able to see all the tabs at once. The "<" and ">" buttons will scroll through the tabs for you so that you can find the tab you want.

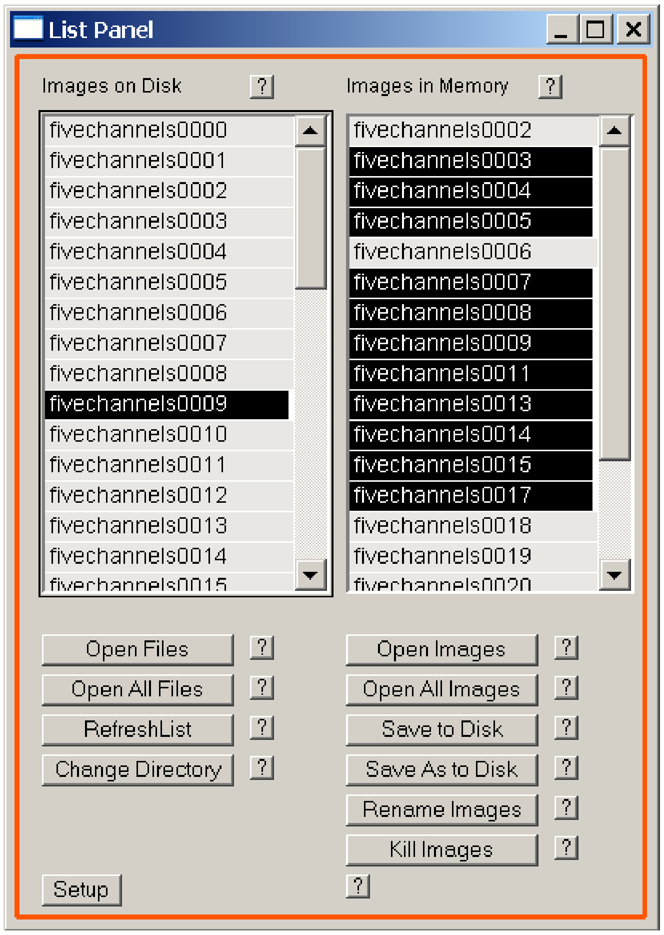
Local Layer Tab

The tabs show which layers are present in the image, in abbreviated format. The Data Channel is abbreviated with the first 2 characters (the full channel name is in the title bar). The T or R represents Trace or Retrace, and a *num, denotes a modified layer (which is not implemented as of the writing of this).

- ## MFP3D List Panel

Lists the Image Files in the current Browse Folder as well as all the Images in memory (root:Images:)

TOC



From this panel you can manage your multitude of offline images. Some default properties of the images are defined in the MFP3D Display Manager, such as which channel is displayed, if there is a colorbar or not, or which colormap (if data is not in the note).

Disk List

This ListBox contains the list of the "Images" in the folder selected from the MFP IP -> Browse menu. Double click on one to load it and the display it. The definition of an Image is that it contains the chars "MFP3D" as the last 5 bytes of the file. There are cases when this Browse Footer is lost in some images, in which case the image will not be seen by the browse. To get around this, you need to load the file into the root:images: data folder, then rebuild the list panel, and then save the image again. The resave function will rebuild the browse footer, which will then allow the browse function to realize that image is indeed an image. If this happens you should definitely let Asylum know, we want to make sure this never happens. email: Support@Rasy.Com

Memory List

This ListBox contains all the images in the Root:Images: data folder. Double click on one to display it.

Open Files

This button loads and displayed the selected images from the Disk List.

Open Images
This button displays the selected images from the memory list.

Open All Files
Same as the Open Files button, just selects all the files for you.

Open All Images
Same as the Open Images button, just selects all the files for you.

Refresh List
This button will refresh the list of Images on the Harddrive. It uses the same path that was defined previously.

Save Images
This button will save the image(s) to the hard drive auto overwriting it.

Save As Images
This button will first rename the image(s), and the save it.

Change Directory
This button is the same as clicking on the MFP IP - > Browse menu

Rename Images
This button will rename the image(s).

Kill Images
This button will remove the selected (Memory List) images from the memory as well as clean up the various other associated waves, such as the mask, backups, restore waves, etc.

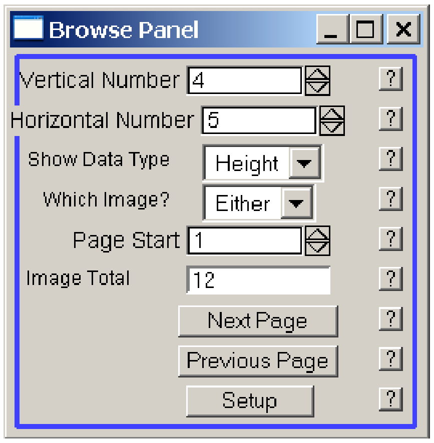
Border Color
This popupMenu will allow you to alter the color in your life. Select it to change the border color for the List Panel.

• MFP3D Browse Panel

This panel allows you to display small copies of your MFP3D Images on the hard drive. The smaller images are shown on the browseGraph. These images are obtained by reading a 128 x 128 image at the end of the file on the hard drive, so that the entire file does not have to be loaded (and is not loaded), so that things run faster. You can double click on the smaller images to load the full wave into the offline image analysis software.

You can load MFP3D Images directly by double clicking on them from the OS. This skips the browse step.
Igor has to be associated with the ibw extension, and igor has to be running the AR software. The image also has to be > 128 scan points (so that it will contain a browse footer).

TOC


Browse Panel

Vertical Number	4
Horizontal Number	5
Show Data Type	Height
Which Image?	Either
Page Start	1
Image Total	12

Vertical Number
This is the maximum number of column you want on your BrowseGraph.

Horizontal Number
This is the maximum number of rows on the BrowseGraph.

Data Type
This popup lists the possible data types available. So you can browse through the phase images if you want. If an image does not contain this particular channel is appears as a white image. You can still double click the non-image to load it from the hard drive.

Which Image
This popup controls which direction is plotted. Options are Trace, Retrace, or either, where either tried to load the trace data first, if that is not there, it tries for the Retrace.

Page Start
This setvar is an offset to the first page of images. Clicking on next page simply increments this control by the number of images shown on a page.

Image Total

This lists the total number of images in the selected hard drive directory.

Next Page

This sets the page start to show the next set of images.

Previous Page

This button backs up 1 page so that the page start is decreased by the number of imags on the BrowseGraph.

Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- **MFP3D Analyze Panel**

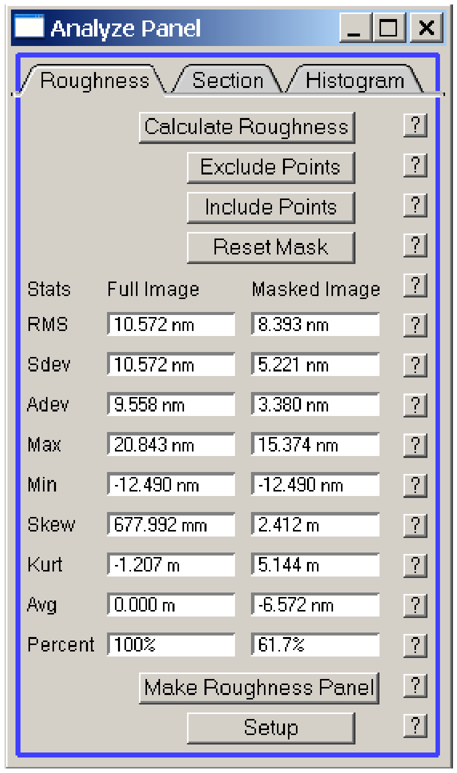
  This panel allows the user to Analyze offline (previously saved) Images.

TOC

- **Roughness Panel**

  Panel to take roughness information from images.

Sub Toc



Stats	Full Image	Masked Image
RMS	10.572 nm	8.393 nm
Sdev	10.572 nm	5.221 nm
Adev	9.558 nm	3.380 nm
Max	20.843 nm	15.374 nm
Min	-12.490 nm	-12.490 nm
Skew	677.992 mm	2.412 m
Kurt	-1.207 m	5.144 m
Avg	0.000 m	-6.572 nm
Percent	100%	61.7%

Calculate Roughness

This recalculates the roughness data for you. The roughness is calculated for both the full image as well as the UnMasked region of the Image. The image data is from the topmost visible Image Layer.

Exclude Points

This allows you to draw an oval on the surface, the data points in this oval will not be included in the fit to the Nth order polynomial. After you click this button, the Button's name becomes make mask, go to the top most image, and draw your oval around the region you want to exclude from the fit, then click make mask.

Include Points

This is very similar to exclude points, just that the rectangle you drag on your image will be used to remove regions of any masks that you had placed there before.

Reset Mask

This clears the mask on the image.

Full Image

These are the title boxes listing the 2 columns of values. The left column are the values for the full image. The right column are for the masked image. It takes the points which are not masked to obtain the Image statistics.

RMS

Root mean square of the image data.
See also WaveStats.

Sdev

Standard deviation of the image data.
See also WaveStats.

Adev

Average deviation of the image data.
See also WaveStats.

Max

Maximum value of the image data.
See also WaveStats.

Min

Minimum value of the image data.
See also WaveStats.

Skew

Skewness of the image data.
See also WaveStats.

Kurt

Kurtosis of the image data.
See also WaveStats.

Avg

Mean value of the image data.
See also WaveStats.

Percent

the percentage of the image that is analyzed, always 100% of the Full Image column, but tells you the fraction of the image that is not masked.

Roughness Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

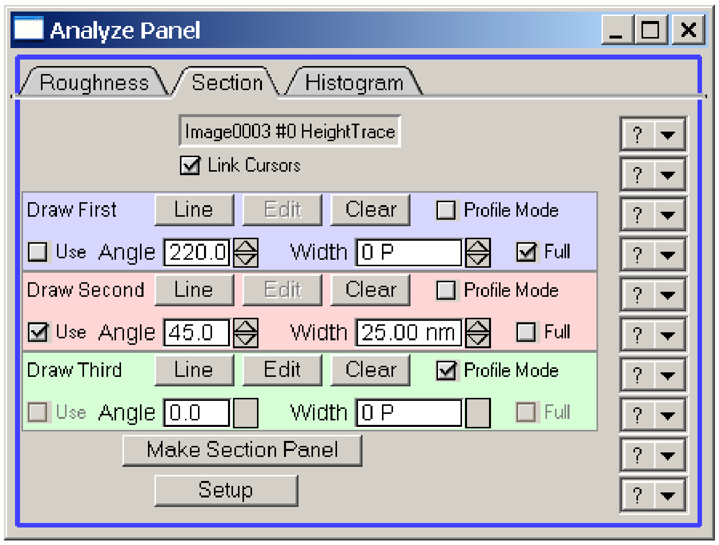
1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- **Section Panel**

  Panel to cut out lines from your images.
  Now with Profile Mode!

Sub Toc


Analyze Panel

Roughness | Section | Histogram

Image0003 #0 HeightTrace

☑ Link Cursors

Draw First — Line — Edit — Clear — ☐ Profile Mode
☐ Use — Angle 220.0 — Width 0 P — ☑ Full

Draw Second — Line — Edit — Clear — ☐ Profile Mode
☑ Use — Angle 45.0 — Width 25.00 nm — ☐ Full

Draw Third — Line — Edit — Clear — ☑ Profile Mode
☐ Use — Angle 0.0 — Width 0 P — ☐ Full

Make Section Panel

Setup

Link Cursors

This checkbox will force the cursors on the Image to corespond to the cursors on the Section. It only works when the width is 0.

Line

This button allows you to draw a line across the topmost displayed image. The manner in which you draw the line is goverened by the Profile Mode checkbox.

Edit

You can move the indivual points of your profile around. This button is only enabled in Profile Mode. However you can draw a straight line (non - Profile Mode), then switch to Profile mode and edit the points in the straight line.

Clear

Clears the drawn line.

Profile Mode

If this checkbox is selected you can freehand draw your profile. If it is not the line will be a straight line. When in Profile mode you can not change the angle or set to full scale.
See also Edit.

Use

This checkbox will use the angle in the Angle Setvar. The angle is used so that it rotates the section around the point you first clicked on the image. Can only be use when NOT in Profile Mode.

Angle

This is the angle of the line you want drawn. You can set it after you have drawn the line. This angle is only used if the "Use" checkbox is selected. Can only be use when NOT in Profile Mode.

Width

This is the width of the line you want to draw. It can have units of distance or points. To switch between units type the first letter of the units after the number. EX: width setvar reads 25 nm, you type in 1 P, which switches to 1 point width. then you type in 50 n, which switches the width to 50 nm. The section will be averaged through the line width to give you the resulting section. Can only be use when NOT in Profile Mode. Also the Link cursors do not work with the a width > 0.

Full

If this checkbox is selected, then the line will be drawn over the entire range of the image. If not selected then the line will be only drawn over the range that you draw. Can only be use when NOT in Profile Mode.

Section Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

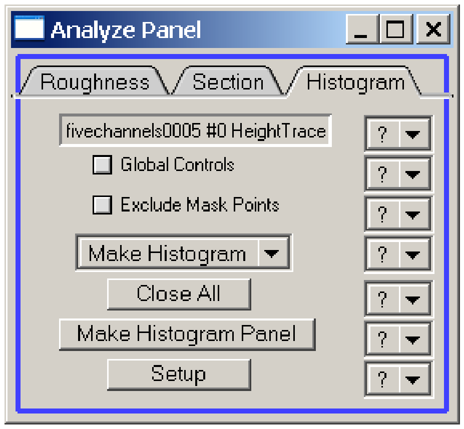
1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

• Histogram Panel

This panel will let you make histograms of your image data

Sub Toc



Which Image

This Titlebox simply lists the title of the graph that the Histogram will be made from.

Global Controls

When this checkbox is selected, the Make Histogram popup will make a histogram for each Image selected in the Global Display List. If you select New, each image gets it's own Graph, if you select Append they are all on the same graph.

Exclude Mask Points

When this checkbox is selected, the mask will be applied to the Image. Masked points are not included in the data to the histogram. This takes a bit of time, particularly for larger images, so be patient.

Make Histogram

This popup sends the data from the topmost displayed layer to the histogram. There are 3 options,
New - will not overwrite any existing histograms.
Append - will add the data to the Last Made histogram (the graph names ImageHist)
Overwrite - will overwrite the last made histogram with the new data.

See also Global Controls

Close All

This button closes all the Histograms opened with the Histogram Panel.

Histogram Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

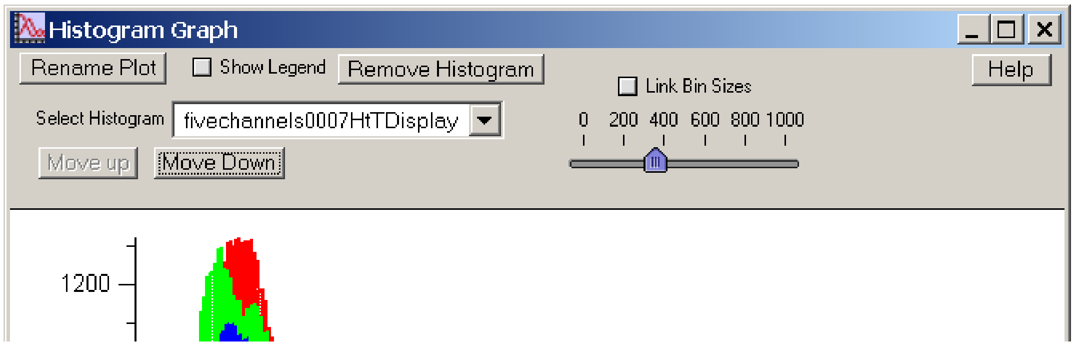
1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- **Histogram Graph**

This graph displays your Histograms.

Sub Toc



Rename Plot

This button (on the Histogram graphs) will rename the plot. This way you can keep that plot separate from the Histogram panel. The Histogram panel only operates on graphs that have the string ImageHist* in their names. This operation is NOT the same as DoWindow/C, the graph's name will tell the controls where their data is.

Show Legend

Puts up a simple legend for your histograms.

Remove Histogram

Removes one of the Histograms from your graph.

Select Histogram

This popup selects the active histogram on the graph, the slider will only work on the active histogram.

Move Up

Moves the Active Histogram up in the display layers, See Select Histogram.

Move Down

Moves the Active Histogram down in the display layers, See Select Histogram.

Bin Slider

This adjust the number of bins for the active histogram, see Select Histogram.

Link Bin Size

When this checkbox is selected, the Bin slider will adjust the number of bins for all displayed histograms. It takes the Bin size of the active histogram, and then calculates the number of bins for each of the other distributions which would have the same bin size.

Help

Brings up this document.

- ## MFP3D Modification Panel

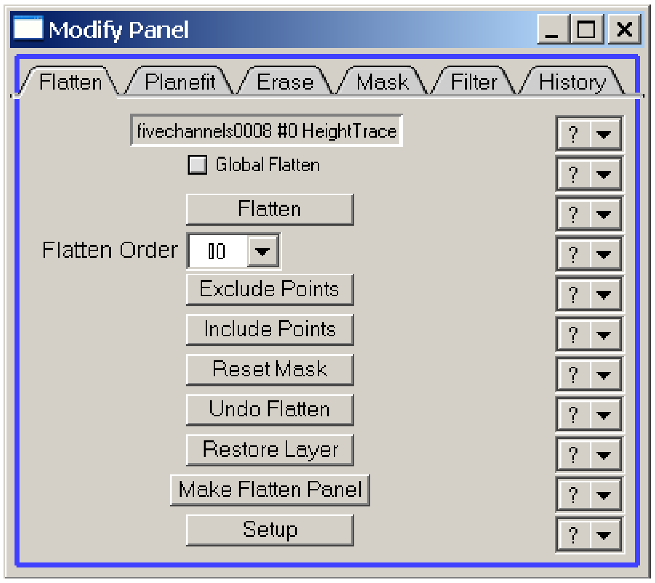
This panel allows the user to Modify offline (previously saved) Images.
Each Tab on this panel can be created in it's own panel by pushing the make "XXX" Panel button that is on each tab.

TOC

- ## Flatten Panel

Panel to Flatten image(s).

Sub Toc



Which Image

This Titlebox simply lists the title of the graph that the actions will be taken on. If the Global Flatten Checkbox is selected it will be grayed out, since the graphs are defined in the Global Display List.

Global Flatten

When this checkbox is selected the controls on this tab will work on all the images selected in the Global Display List. That is except for the Exclude Points and the Include Points buttons.

Flatten

This is the "do" button. It will fit each scan line with a polynomial line and subtract that from the data. The order of the polynomial is defined in Flatten Order. This is particularly useful in removing the streaky lines in an image. One very nice method is to do a 0 order flatten, then go to the Mask Panel, do an Iterative mask (calc method). and then you can do a higher order flattening (back in the Flatten Panel).

Flatten Order

This is the order of the polynomial fit to the scan line. It is generally safe to apply 0 and first order polynomials to the image, but care should be taken to ensure that higher order flattening is not altering the data you care about.

Exclude Points

This allows you to draw an oval on the surface, the data points in this oval will not be included in the fit to the Nth order polynomial. After you click this button, the Button's name becomes make mask, go to the top most image, and draw your oval around the region you want to exclude from the fit, then click make mask.

Include Points

This is very similar to exclude points, just that the rectangle you drag on your image will be used to remove regions of any masks that you had placed there before.

Reset Mask

This clears the mask on the image(s).

Undo Flatten

This undoes the all of the Flattening steps done to the Image(s). Doing different modifications to the data will clear this option.

Restore Flatten

This button reverts the image to the hard drive state. If this is the first time that the layer is being restored it will load from the hard drive. If the layer is one that was inserted as a modify layer, it will be reverted to the state when it was first inserted. The note is also restored for the shown layer. If you want to restore all the layers, load your data from the hard drive using the MFP3D List Panel.

Flatten Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

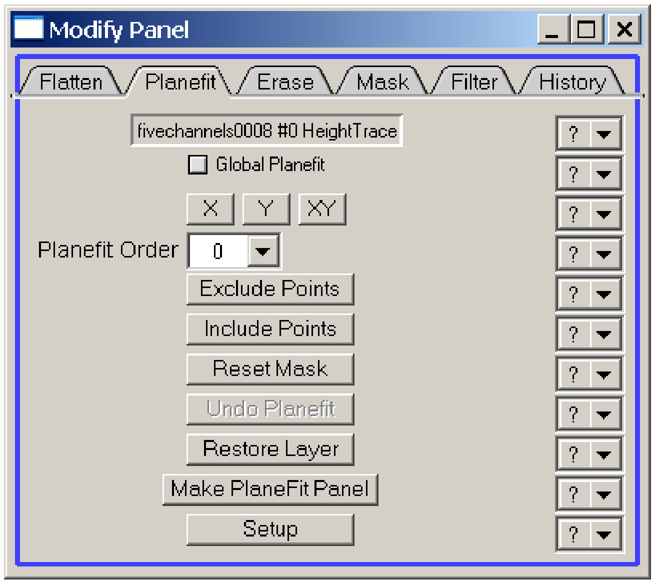
1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- **PlaneFit Panel**

Panel to Planefit image(s).

Sub Toc



Which Image

This Titlebox simply lists the title of the graph that the actions will be taken on. If the Global PlaneFit Checkbox is selected it will be grayed out, since the graphs are defined in the Global Display List.

Global PlaneFit

When this checkbox is selected the controls on this tab will work on all the images selected in the Global Display List. That is except for the Exclude Points and the Include Points buttons.

X Y XY

These 3 buttons are the do buttons for the plane fit panel. Plane fitting takes the best fit Nth order polynomial surface and subtracts that from the image. Nth being defined by the PlaneFit order. The X and the Y buttons do this process, but for only one axis, locking the fitting parameters of the other axis to zero. The exception is for zero order fits, which is simply taking the average of the data and subtracting it.

PlaneFit Order

This is the order of the plane to be removed from the image. Details of the fitting are given above in X Y XY

Exclude Points

This allows you to draw an oval on the surface, the data points in this oval will not be included in the fit to the Nth order polynomial. After you click this button, the Button's name becomes make mask, go to the top most image, and draw your oval around the region you want to exclude from the fit, then click make mask.

Include Points

This is very similar to exclude points, just that the rectangle you drag on your image will be used to remove regions of any masks that you had placed there before.

Reset Mask

This clears the mask on the image(s).

Undo Planefit

This undoes the all of the planefit steps done to the Image(s). Doing different modifications to the data will clear this option.

Restore Planefit

This button reverts the image to the hard drive state. If this is the first time that the layer is being restored it will load from the hard drive. If the layer is one that was inserted as a modify layer, it will be reverted to the state when it was first inserted. The note is also restored for the shown layer. If you want to restore all the layers, load your data from the hard drive using the MFP3D List Panel.

Planefit Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

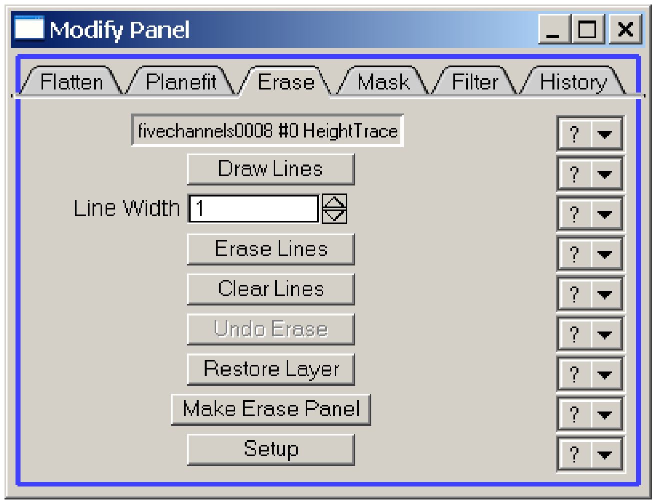
What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- **Erase Panel**

This tab allows you to erase scan lines from an image.

Sub Toc

Erase Setup



Which Image

This Titlebox simply lists the title of the graph that the actions will be taken on.

Draw Lines

By clicking here you can now draw horizontal lines on your image, which will then be averaged out of your image. To get back to the normal axis scaling mouse function you will need to either Clear Lines or Erase Lines.

Line Width

This is the width of the line to remove in scan lines. Note that the exact diameter of the drawn line is only accurate at normal scaling. When you zoom into the image the lines will apear to be much thinner than the number of lines which they will average out.

Erase Lines

This the is the do button for the Erase Panel. It executes the averaging of the scan lines marked by the red horizontal lines that were drawn onto the image with the Draw Lines button.

Clear Lines

This button removes any red horizontal marker lines that were drawn on the image.

Undo Erase

This undoes the all of the Scan line erasing that you have done to the Image(s). Doing different modifications to the data will clear this option.

Restore Erase

This button reverts the image to the hard drive state. If this is the first time that the layer is being restored it will load from the hard drive. If the layer is one that was inserted as a modify layer, it will be reverted to the state when it was first inserted. The note is also restored for the shown layer. If you want to restore all the layers, load your data from the hard drive using the MFP3D List Panel.

Erase Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

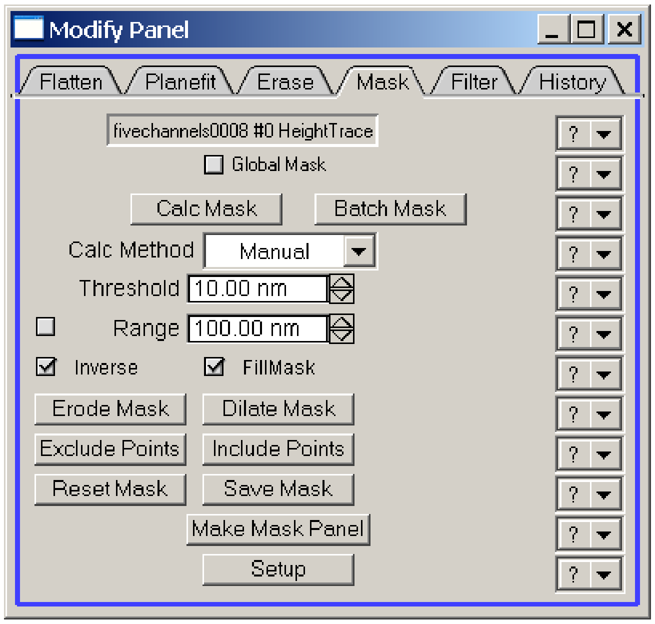
1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

• Mask Panel

Panel to make mask to apply to the Flatten Panel and PlaneFit Panel
See also Wavemetrics ImageThreshold function.

Sub Toc



Which Image

This Titlebox simply lists the title of the graph that the actions will be taken on. If the Global Mask Checkbox is selected it will be grayed out, since the graphs are defined in the Global Display List.

Global Mask

When this checkbox is selected the controls on this tab will work on all the images selected in the Global Display List. That is except for the Exclude Points, Include Points, and the Batch Mask buttons.

Calc Mask

This is the do button for the Mask panel, it will execute the mask calculation based on the parameters entered and render it on the topmost image.

Batch Mask

The Batch Mask is tied to the MFP3D Display Manager. It will take the *process* applied to the current (topmost) Display graph, and then apply that same process to the images listed in the Global Display List. So if you do an adaptive mask on the topmost image, and then Dilate it twice, invert it, and then Erode it. It will apply all 5 steps to each of the images listed. For custom masks, such as Exclude Points and Include Points, it will scale the current mask to fit the images listed in the Global Display List.

Calc Method

The masks are calculated using Wavemetrics ImageThreshold function. The details of the various methods are detailed in that function.

Mask Threshold

This is only used as an input for the Manual Mask. It is the threshold of the sharp edge on the surface to draw the mask around. In that if the surface feature goes through a step > threshold the mask edge will be drawn through those pixels.

Mask Range

If the checkbox to the left of this control is selected then the Range will be applied to the mask.
What this does is calculates a second mask at a threshold of threshold + range. It then compares the 2 masks so that only values of the surface from threshold to threshold + range are in the mask.
Note that adaptive masks do not use the inputted value for the range.

Inverse Mask

This flips the mask so that excluded points become the included points.

FillMask

This is an easy way to see which points are being excluded. I highly recommend it!

Erode Mask

This makes the mask slightly smaller to cover fewer points.

Dilate Mask

This expands the mask slightly to cover more points.

Exclude Points

This allows you to draw an oval on the surface which will include points into the mask. After you click this button, the Button's name becomes make mask, go to the top most image, and draw your oval around the region you want to exclude from the fits, then click make mask.

Include Points

This is very similar to exclude points, just that the rectangle you drag on your image will be used to remove regions of any masks that you had placed there before.

Reset Mask

This clears the mask on the image(s).

Save Mask

This puts a copy of the mask wave in the root:Images:Masks: data folder

Mask Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

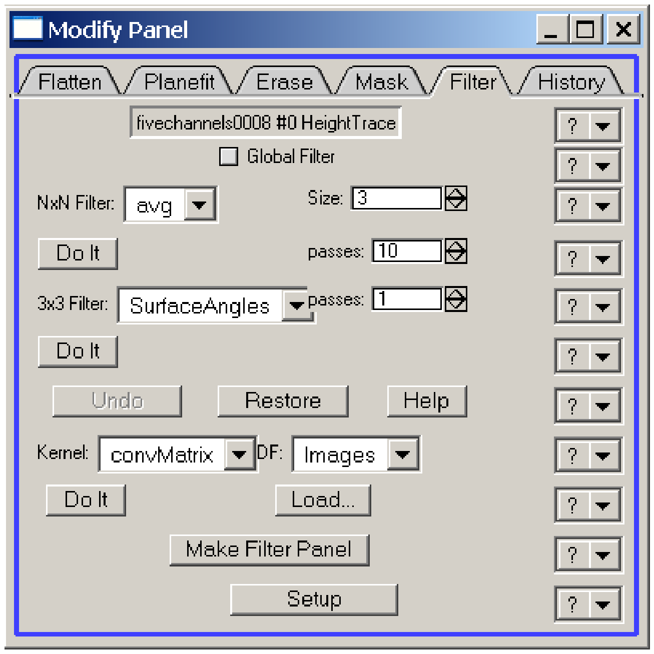
What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- **Filter Panel**

Panel to apply various wavemetrics image filtering to the image(s).
NOTE: The masks are Finally incorporated into this data processing.

See also Wavetrics Image Processing.
Wavemetrics MatrixFilter.

Sub Toc



fivechannels0008 #0 HeightTrace
Global Filter
NxN Filter: avg
Size: 3
passes: 10
3x3 Filter: SurfaceAngles
passes: 1
Kernel: convMatrix
DF: Images

Control Sets

There are 3 separate control sets on this panel.
NxN Filters, are filters of variable width. They basically go through every pixel of the image, look at the

nearest neighbors of that pixel, and apply an operation on those neighboring pixels to calculate the new pixel.

The 3x3 Filters are similar to those of the variable width operations, exect that the width of these are fixed at 3 pixel wide square. These are simple operations that find the gradient of the image.

Kernel controls
There is a set of controls to run the Kernel (the NxN matrix applied to each pixel). These controls are hidden by default, as they are not very necessary to the usefulness of this panel.

The Kernel is the MatrixConvole Matrix.

Example:
```
  SetDataFolder Root:
  Make/O ConvMatrix={{1,1,1},{0,0,0},{-1,-1,-1}}
  Edit/k=1 ConvMatrix          //show us our Convolution Matrix
```

Then from the Kernel DataFolder, select current data folder (which we just set to the root)
Then from the Filter Kernel, you can select your 2D ConvMatrix that you just made.
Now you are locked and loaded, and you can apply your newly made convolution matrix to the Image listed in the which Image titlebox, or if you have the Global Filters selected, it will work on the Image(s) selected in the Global Display List.

Which Image

This Titlebox simply lists the title of the graph that the actions will be taken on. If the Global Filter Checkbox is selected it will be grayed out, since the graphs are defined in the Global Display List.

Global Filter

When this checkbox is selected the controls on this tab will work on all the images selected in the Global Display List.

NxN Filter

This is basically just a switch for the methods used in the matrixfilter function.

Filter Size

This defines how many neighbors that the filtering uses to calculate the filtered data points.

Passes NxN

This defines how many iterations the NxN Filter is applied.

Filter Do It

These buttons are the do buttons for the panel. The control set they are closest to are the ones they will operate on. The control sets are outlined in the subtopic for the Filter Panel.

Passes 3x3

This defined how many iterations the 3x3 Filter is applied.

Filters 3x3

This is basically just a switch for the methods used in the matrixfilter function.

Filter Undo

This undoes the all of the Filtering steps done to the Image(s). Doing different modifications to the data will clear this option.

Filter Restore

This button reverts the image to the hard drive state. If this is the first time that the layer is being restored it will load from the hard drive. If the layer is one that was inserted as a modify layer, it will be reverted to the state when it was first inserted. The note is also restored for the shown layer. If you want to restore all the layers, load your data from the hard drive using the MFP3D List Panel.

Filter Kernel

This is the name of the wave used as the Kernel. See example in Control Sets.

Kernel DataFolder

This is the data folder of the Kernel wave. See example in Control Sets.

Load Kernel

This allows you to load your own kernel waves for custom filtering. See example in Control Sets.

Filter Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

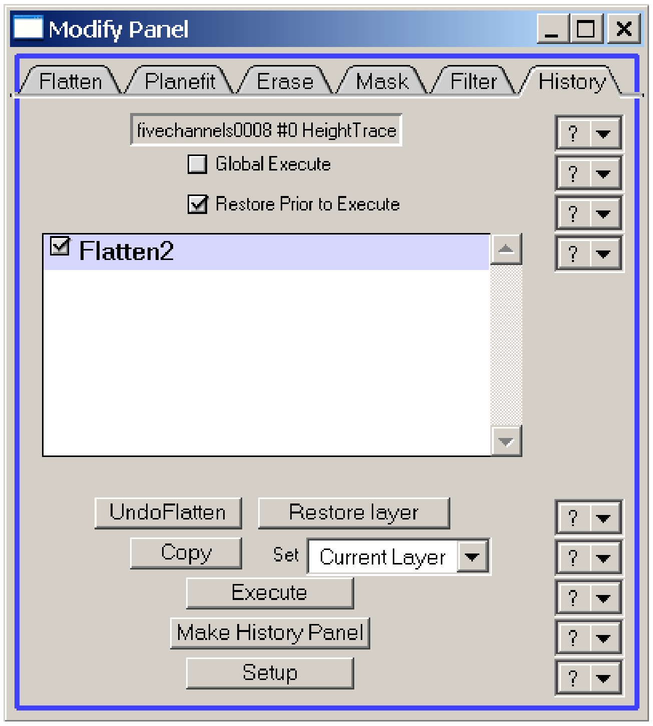
1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- ## History Panel

Panel to view and edit the modification history of your image(s).
Read the History Execute first.

Sub Toc



Note: Read the History Execute First.

Which Image

This Titlebox is a little different on this image. It lists the image from which the history is being taken, when the Set History popup is set to the current layer. If Global Execute is NOT selected it is also the image which will have the modifications applied to. If the Global Execute Checkbox IS selected the images are defined in the Global Display List.

Global Execute

When this checkbox is selected the controls on this tab will work on all the images selected in the Global Display List.

Restore First

When this checkbox is selected each of the images selected will be restored before any modifications are applied. If this checkbox is not selected, then the modifications listed in the History List will be added to the modifications already done on the image(s).

History List

This Listbox lists the modifications that have been applied. If the Set History popup is set to current layer this list box lists the modifications applied to the image and layer listed in which image. If you un mark the checkboxes in the list those modifications will not be applied when a History Execute is called.

History Undo

This button will undo the last N steps of the specified type. Say you ran 2 flattens and the a planefit. The Undo button would read Undo planefit, and undo the last step. Say you ran a planefit, then 3 flattens, the undo button would read Undo Flatten and undo the last 3 steps. It is only enabled if the Global Execute checkbox is NOT selected.

History Restore

This button restores the visisble layer of the topmost image, or if the Global Execute checkbox is selected, all of the images listed in the Global Display List.

Copy History

This button is enabled when the Set History is set to current layer. It copies the current layer's history to a safe place, and then you can use that copy to apply the modifications to other images. If you call a History Execute, with the Set History set to current layer it will automatically call this button.

Set History

This popup lists the available History copies. The first option is Current layer, and when selected the History List will show the modification history of the topmost graph's visible layer, which is listed in Which Image. If you have made a "Copy History", then you can select that copy from this popup.

History Execute

OK, so you have done some modifications to a image. You then want to see what that Image would look like if you did not have the Erase Lines done in the second step. So you go to your History List, Un-select the Erase checkbox, and then hit Execute (that is with the Global Execute NOT selected, and the Restore First IS selected). Then it makes a copy of that history (Copy History), It restores the Image to it's original state, and then goes through the list of modifications, applying each Selected modification to the Image. You can then set the Set History popup Back to Current Layer and see that it is missing the second step.

Or, you have a set of modifications you want to apply to a whole bunch of images. You tweak out your execution order, and what you want done to the image, by modifying it as you normally do. Then you go to the History Panel. You make a copy of the history you want to use with the Copy History button. You select Restore First, and Global Execute, you then select all the Images you want to work on from the Global Display List. And then you hit the Execute Button, and watch the modifications fly, or grab lunch depending on how many images you have to run. Hell, you could even go home and grab some much needed sleep, come back in when it is done.

History Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- **MFP3D Master Force Panel**

  These panels allows the user to Draw, Save and Import a path for the tip to follow.

TOC

Force Display Glossary

Some of the basic concepts of the Asylum Force interface that are most often confused by new users.
Force Plot Name: The Force plot name is broken unto 4 parts. The BaseName, which is entered on the Main Panel when collecting the data. The Suffix, which is the 4 character number of the force plot. The DataType, which is an abbreviation of the data type, such as defl for Deflection, or amp for Amplitude. And the Section which denotes which part of the force plot the data is from. For example BaseName0003Defl_Ret is the Deflection from the retract portion of force plot BaseName number 0003.
if there is no section part of the name, then the wave corresponds to the entire pull.

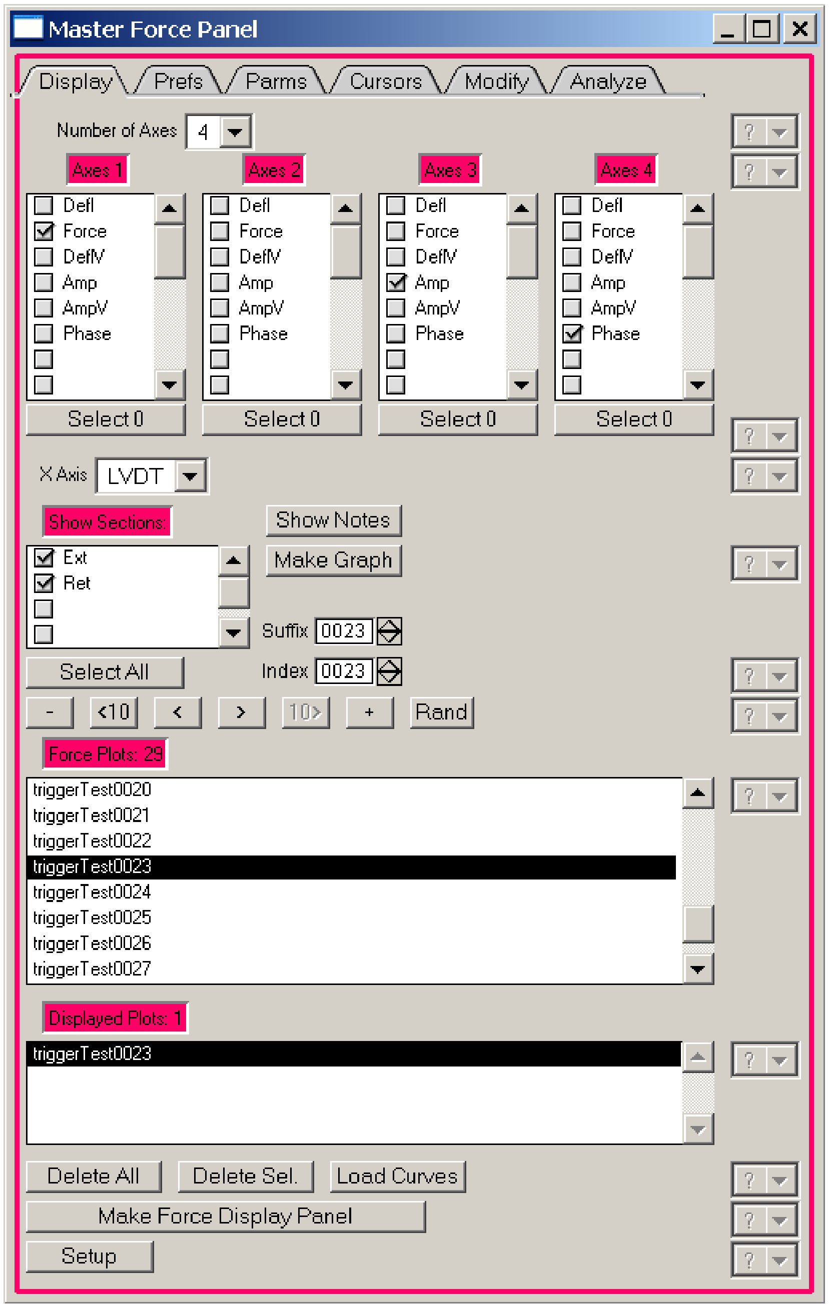
Raw vs. LVDT: Raw is the data collected on the Z LVDT sensor. This data is typically noisy and often not monotonic. To clean it up and make it easier to work with we fit it to a 7th order polynomial and that fit is then used to create the LVDT. This was called Zposition on the 1D's.

- **MFP3D Force Display Panel**

  Main Force Display Panel, used to control which force plots are displayed on the Force Display Graph.

Sub Toc

Force Display List
Delete All
Delete Selected
Load Curves
Make Force Display Panel
Setup



Number of Axes 4

Axes 1: [✓ Force]
Axes 2: none
Axes 3: [✓ Amp]
Axes 4: [✓ Phase]

X Axis LVDT

Show Sections: [✓ Ext] [✓ Ret]

Suffix 0023

Index 0023

Force Plots: 29

```
triggerTest0020
triggerTest0021
triggerTest0022
triggerTest0023
triggerTest0024
triggerTest0025
triggerTest0026
triggerTest0027
```

Displayed Plots: 1

```
triggerTest0023
```

Number of Force Axes

This popup controls how many axes you want to have stacked on top of each other. A left new left axes will not appear until on of the available data types have been selected from the AxesData Types list.

Axes Data Types

These list boxes list all the data types that the Force software knows about. If the data type shows up in the list box as Black text, then all of the selected force plots (in the Force Plot List) have that data type. If the Text for a particular data type is grey, than only some of the force plots have that data type available. If the text is white, then none. You can hold down the mouse on a data type to be able to read the data types that are not available. All of the available data types selected will show up in the specified axes. For example, if you have defl and amp selected in axes 1, and one force plot selected in the Force Selection List (which has both deflection and amplitude waves saved), then the bottom Left axes on the Force Display Graph will contain both deflection and amplitude.

Select None

These buttons just below the Axes Data Types List boxes will un-select all the data types for that axes (which will then remove the axes from the graph).

Force X axis

This popup controls which data type is used for the X axis of all of the Left axes.

Show Note

This button will bring up the ForceHeaderWave in a table.
*****NOTE***** parameters should NOT be changed in this table. These are the parameters the software uses to interpret the force plots and bad things will happen if you change the wrong parameters. ALL changes to the parameters should be done through the MFP3D Force Parms Panel.

See also the Advanced Topic ForceHeaderWave in
MFP3D Force DataFolders -> Parameters

Force Section List

This listbox lists the available sections of the force plots. It is typically Ext (Extend) and Ret (Retract). If the Force plots selected in the Force Plot List have dwell saved, then those options will also be visible. This listbox executes in a very similar fashion to that of the Axes Data Type.

Make Graph

This button will create the Force Display Graph. It will also bring the graph forward if it has already been built. In the MFP3D Force Prefs Panel you can set it to Auto Make Force Graph.

Force Suffix

This will offset the first selected force plot in the Force Plot List to the closest suffix entered.
See also Force Index.

Select All Sections

This button will select all of the Section in the Force Section List.

Force Index

This will offset the first selected force plot in the Force Plot List to select the entered index. The Index is an index to the force plot list, not the Suffix.
See also Force Suffix.

-

The Minus Button. Unselects the last selected force plot from the Force Plot List.

<10

Shifts the First selected force plot in the Force Plot List back 10 Indexes.
See also Force Index.

<

Shifts the First selected force plot in the Force Plot List back 1 Index.
See also Force Index.

>

Shifts the First selected force plot in the Force Plot List forward 1 Index.
See also Force Index.

10>

Shifts the First selected force plot in the Force Plot List forward 10 Indexes.
See also Force Index.

+

The Plus Button. Selects the next un-selected force plot from the Force Plot List. This is typically the first unselected force plot after the first selected one, but if all of the force plots after the first selected are also selected, then the plus button will select the closest force plot to the first selected.

Rand

Shifts the first selected force plot in the Force Plot List by a random number. This can clip the number of force plots selected (until we take to time to write a better algorithm).

Force Plot List

This Listbox lists all of the force plots in the saved force folder. Selecting one or more of them will put those plots up on the Force Display Graph. The selected force plots are shown below in Force Display List.

Force Display List

This list box lists all of the force plots displayed on the Force Display graph. The coloring of the listbox corresponds the color table coloring of the Force Plots. See also Use Color Table.

Delete All

This button will delete all of the force plots listed in the Force Plot List from Igor's Memory. It leaves the files on the Hard drive (if any).

Delete Selected

This button deleted the selected force plots in the Force Plot List from Igor's Memory. It leaves the files on the Hard drive (if any).

Load Curves

This button will ask you for a new directory to load Force Curves from.

Make Force Display Panel

This button will make the Stand alone Force Display Panel (it pushed on the tabbed MasterForcePanel). It pushed from the Stand alone Force Display Panel it will make the MasterForcePanel.

Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

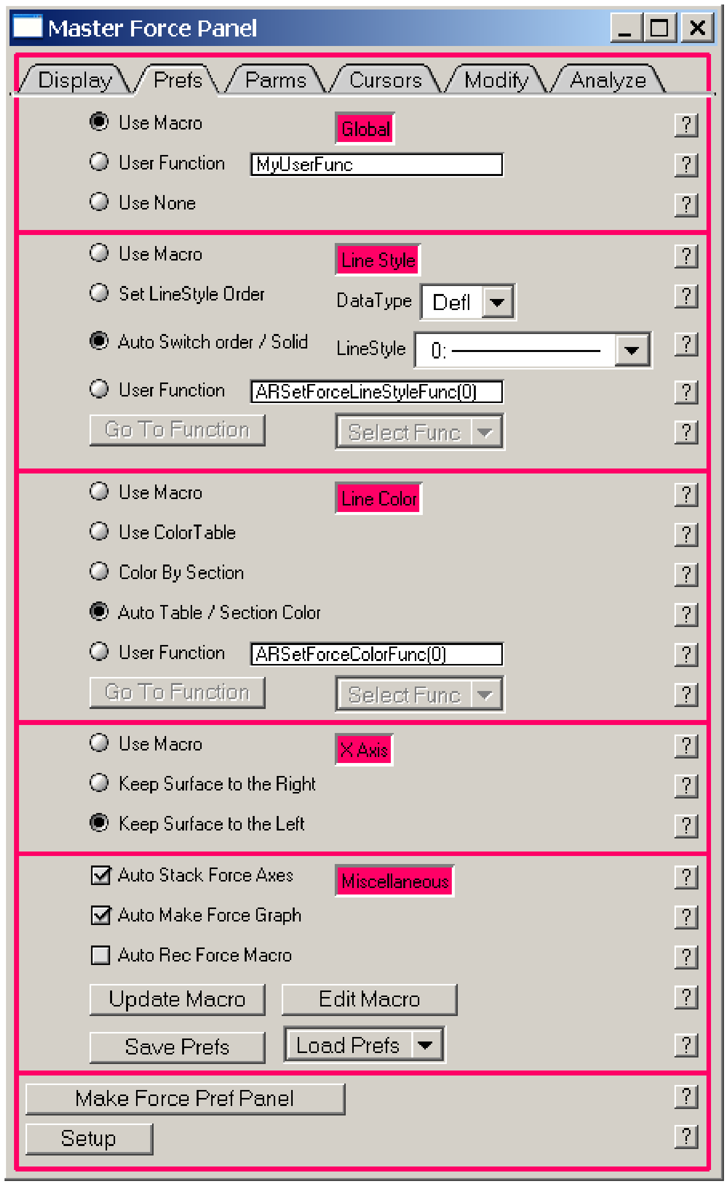
What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- ## MFP3D Force Pref Panel

a panel to customize the Force Display Graph to your liking.
Executing Order:
The recreation Macro or User Function is executed.
Then if selected the Left Axes are stacked.
Then the selected LineStyle operation is preformed (if Use Macro is selected then none is preformed)
Then the Selected Line Color operation is preformed (if Use Macro is selected then none is preformed).
And finally the X axis is Updated as per the control's setting.

In this way you can have linestyle and line color data in the recreation macro, which will be overridden
by
the subsequent operations.

Sub Toc


Master Force Panel

Display / Prefs / Parms / Cursors / Modify / Analyze

Global
- [● Use Macro]
- ○ User Function: MyUserFunc
- ○ Use None

Line Style
- ○ Use Macro
- ○ Set LineStyle Order — DataType: Defl
- [● Auto Switch order / Solid] — LineStyle: 0:
- ○ User Function: ARSetForceLineStyleFunc(0)
- Go To Function — Select Func

Line Color
- ○ Use Macro
- ○ Use ColorTable
- ○ Color By Section
- [● Auto Table / Section Color]
- ○ User Function: ARSetForceColorFunc(0)
- Go To Function — Select Func

X Axis
- ○ Use Macro
- ○ Keep Surface to the Right
- [● Keep Surface to the Left]

Miscellaneous
- [✓] Auto Stack Force Axes
- [✓] Auto Make Force Graph
- [ ] Auto Rec Force Macro
- Update Macro — Edit Macro
- Save Prefs — Load Prefs

Make Force Pref Panel

Setup

Use Macro
This radio button will use the recreation macro to stylize your Force Display Graph. See also Update Macro and Edit Macro. Also Advanced Topic MFP3D Force DataFolders -> ForceGraphMacro.

Use Func
This radio button (if selected) will make it so that the user function specified in the setvar to the right will be executed instead of the recreation macro built. The default MyUserFunc, does not exist.

Use None
When this radio button is selected, then it will not use either of the recreation macro or the user function to stylize the Force Display Graph.

Use LineStyle Macro

This Radio button will basically the same as do nothing. It assumes that the recreation macro has lineStyle commands in it that you want to have executed. If that is the case, then have Use Macro selected and then The Use LineStyle Macro radio button selected. See Execution order above the sub toc.

Set LineStyle Order

When this radio button is selected then the lines will have a line style as defined by the Data type. This is controlled with the Data Type and LineStyle popups.

Data Type

This popup switches between the known data types to allow you to change the lineStyle for that data type with the LineStyle popup. Slaved data Types use the same linestyle as their parent. Slaved data types are directly calculable from a collected data type, for example Force is a slave to Deflection. For convenience we have called deflection the parent and DeflV a slave, even though technically it is the other way around.

Auto Switch LineStyle

When this radio button is selected then the linestyle will switch between Solid and the lineStyle defined by the Set LineStyle Order. It will be solid of only one set of traces is on a particular left axes.

LineStyle

This popup is the lineStyle for the data type defined in the Data Type popup. Changing this popup will change the lineStyle for that data type when Set LineStyle Order is being used.

Use LineStyle Func

This Radio button will execute a user function instead of the previous options for the lineStyle. The function that will be executed is listed in the SetVar to the right (LineStyle UserFunction). The Default Setting, ARSetForceLineStyleFunc(0), is actually what is called for the Auto Switch LineStyle. The Select Func popup below the setvar lists all the functions that end in "*ForceLineStyleFunc". The GoTo button will bring up the function specified in the SetVar.

LineStyle UserFunction

This setVar lists the function that will be executed if the Use LineStyle Func radio button is selected. The select Function Popup will swap out the function listed in this SetVar.

Use Color Macro

This is the same deal as the Use LineStyle Macro, but applies to the Line colors.

Use Color Table

If this radio button is selected then the lines will be colored according to the color displayed in the Force Display List. So that all the traces from a given BaseName+Suffix combination will have the same color. A more detailed description of how this works is given in the advanced topic MFP3D Force DataFolders -> ForceColorWave.

Use Section Color

When this radio button is selected then the Force plot with be Colored by the section. Red for Extend, Blue for Retrace and some dark magenta for dwells.

Auto Switch Color

When this radio button is selected then it will switch between the Color Table and he Section color options. If only one force plot is selected then it colors by section, if multiple Force plots are selected it colors by Color Table.

Use Color Func

This the the same deal as the Use LineStyle Func....
This Radio button will execute a user function instead of the previous options for the Line Color. The function that will be executed is listed in the SetVar to the right (Color UserFunction). The Default Setting, ARSetForceColorFunc(0), is actually what is called for the Auto Switch Color. The Select Func popup below the setvar lists all the functions that end in "*ForceColorFunc". The GoTo button will bring up the function specified in the SetVar.

Color UserFunction

This setVar lists the function that will be executed if the Use Color Func radio button is selected. The select function Popup will swap out the function listed in this SetVar.

Use XAxis Macro

This uses the macro (see Use Macro), to handle the X axis.

Use XAxis Right

When this radio button is selected then it will try to keep the surface towards the right hand side of the axis limits.

Use XAxis Left

When this radio button is selected then it will try to keep the surface towards the left hand side of the axis limits.

Auto Stack Force Axes

When this CheckBox is selected then the left axes will be stacked on top of each other with the fifth axes at the top and the first axes on the bottom. If this is not selected it is up to the "use macro" or "Use Func" to handle the Left axes mess.

Auto Make Force Graph

When this CheckBox is selected then the Graph will be made again any time one of the display parameters are changed on the Force Display Graph.

Auto Rec Force Macro

When this CheckBox is selected then the style macro that is run by "Use Macro" will be updated every time the graph is killed.

Edit Macro

This Button will bring up the style macro up in a table. You can see the commands that it will execute, as well as enter new ones, and alter existing ones.

Update Macro

When this button is hit it will rebuild the style macro. See also Auto Rec Force Macro.

Save Prefs

This button will prompt you for a file name to save the state of the Force Preference Panel. It primarily consists of the style macro wave, but also the state of the checkboxes and radio buttons. The wave that it builds will be saved in the User path.

Load Prefs

This popup lists all the Force Preference states that have been saved in the User path. selecting one will load that state, and rebuild the panel with the new state.

Make Force Pref Panel

This button will make the Stand alone Force Prefs Panel (it pushed on the tabbed MasterForcePanel). It pushed from the Stand alone Force Prefs Panel it will make the MasterForcePanel.

Force Pref Panel Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

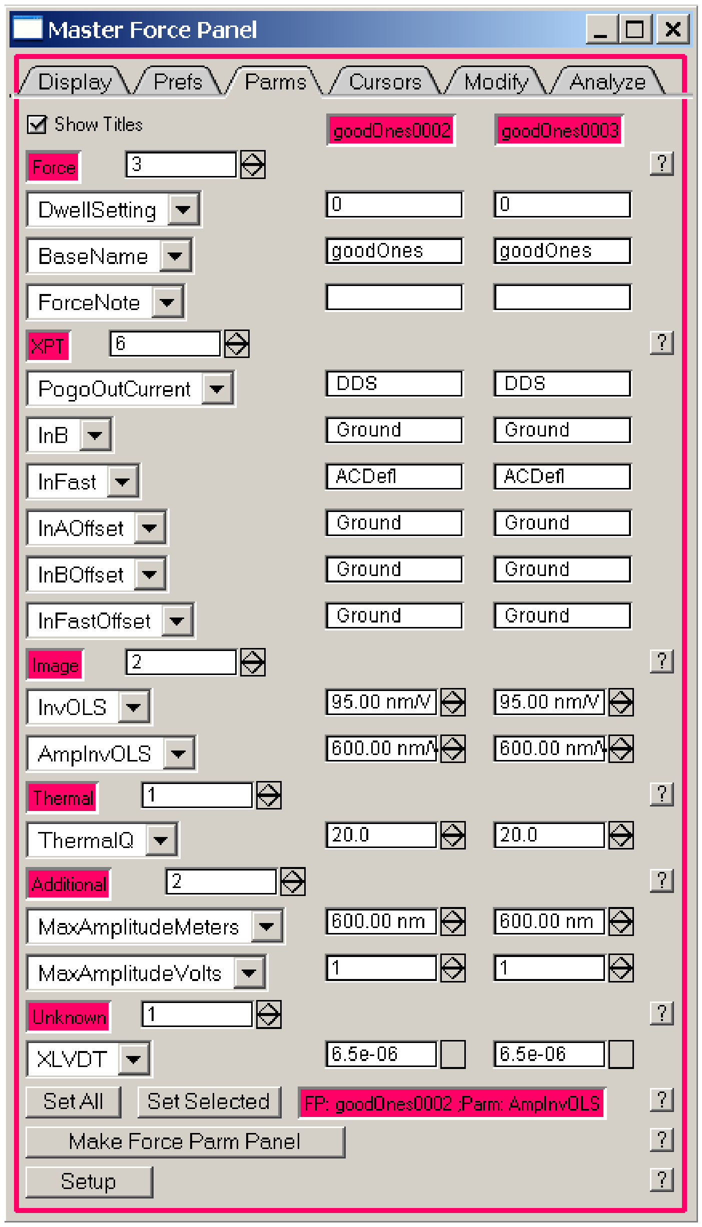
• MFP3D Force Parms Panel

This panel will let you see and edit the various header parameters associated with the saved force plots.
The force plots selected in the Force Plot List are listed in the columns of this panel, and can be read from the titleboxes on the top of the panel. The header wave is broken into 9 sections, which are listed in the subtoc from Force Parms to Unknown Parms. The popups on the left side will list all the parameters for that section, and the setvar on the right the parameter titlebox (shown if Show Titles is selected) determines how many parameter popups will be displayed. Then there are the setvars that actually show the parameters for each selected parameter and for each force plot selected. Most of these setvars can not be changed, which means the software does not know what to do when that parameter is changed. The parameters that can be changed with the setvars are listed by the function GetEditableForceParms(), which as of this writing are:
Invols
AmpInvols
SpringConstant
MaxAmplitudeVolts
MaxAmplitudeMeters
ThermalFrequency

DriveFrequency
ThermalQ
ForceNote

Many of these parameters can be changed just for the dissipation wave (such as MaxAmplitude).

Sub Toc



☑ Show Titles		goodOnes0002	goodOnes0003
Force	3		
DwellSetting		0	0
BaseName		goodOnes	goodOnes
ForceNote			
XPT	6		
PogoOutCurrent		DDS	DDS
InB		Ground	Ground
InFast		ACDefl	ACDefl
InAOffset		Ground	Ground
InBOffset		Ground	Ground
InFastOffset		Ground	Ground
Image	2		
InvOLS		95.00 nm/V	95.00 nm/V
AmpInvOLS		600.00 nm/V	600.00 nm/V
Thermal	1		
ThermalQ		20.0	20.0
Additional	2		
MaxAmplitudeMeters		600.00 nm	600.00 nm
MaxAmplitudeVolts		1	1
Unknown	1		
XLVDT		6.5e-06	6.5e-06

Set All | Set Selected | FP: goodOnes0002 ;Parm: AmpInvOLS

Make Force Parm Panel

Setup

Show Titles

This Checkbox, when selected will show the sub-section titleboxes and the setvars that determine the number of parameters for that sub-section. If it is not selected then just a red line will be drawn across the panel to separate the various parameters. When you leave the setup you can have a very tall panel, to the point that you can't see the Top and bottom at the same time. By hiding the titles you can gain a little bit or room on your panel.

Force Parms

This subset of parameters is from the ForceVariablesWave. It includes parms such as Velocity, and DwellSetting. This section also has a couple parameters which do not belong to other waves, such as DDSamplitude, and BaseSuffix.

SubForce Parms

This subset of parameters are from the ForceVariablesWave, but are deemed less important than the Force Parms. Parameters such as Hamster sensitivity, number of points per nm, and continuous Force.

XPT Parms

This section contains the cross point panel parms for the force plot.

Image Parms

This section contains the parameters from the Master Variables Wave. Some of the important parameters being InVOLS, AmpInVOLS, and SpringConstant.

SubImage Parms

This section also holds parameters from the Master Variables Wave, but the parameters are ones that do not seem to be as important to force measurements. Examples of which include: ScanSize, XIGain, and LastScan.

Thermal Parms

This section includes the important parameters from the Thermal Variables Wave. Editable parameters include: ThermalQ and ThermalFrequency.

SubThermal Parms

This section contains all the useless info from the Thermal Variables Wave.

Additional Parms

This is where some new parameters are added. Logically the parameters which do not have an original home that ended up in Force Parms should have been added to this section. However we put them up front so they were easier to find. Right now this section only contains MaxAmplitudeMeters and MaxAmplitudeVolts. These parameters are both editable, but only effect the dissipation wave. OK, I took my own advice and put some of the unknown parms into here, such as Time, and EndHeadTemp.

Unknown Parms

This is where parameters are found in the notes of the waves, but the template wave does not know about them. The archaic ForceSuffix parameter is in this category, as the parameter is no longer used, but is often found in old wave notes.

Set other parms

Here we have 2 buttons and a titlebox. The 2 buttons will take the last entered value and apply it to either the force plots selected in the Force Plot List (Set Selected), or all of the force plots in memory (Set All). The title box tells you what the last entered parameter was (force plot name and which parameter).

Make Force Parm Panel

This button will make the stand alone Force Parm Panel if click on the Master (Tabbed) Force panel, or vice verse.

Export Parm Table

This button will export the parameters listed on the panel into a table. They are not in the same order, the variables are listed first and then the strings are added after that.

Force Parms Panel Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- ## MFP3D Force Cursors Panel

This panel will let you link the cursors between force traces.

Sub Toc

A topic

Oh go ahead

Another Topic

and try to format a new igor topic.

Force Cursors Panel Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

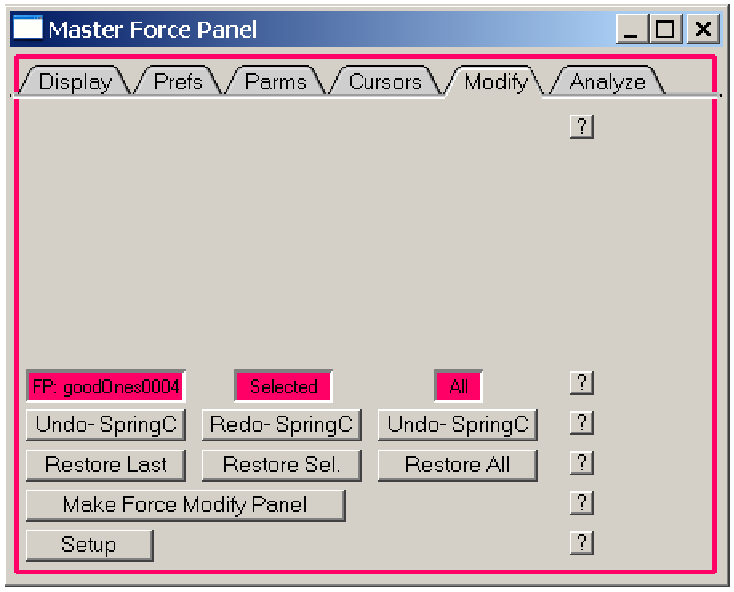
What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- ## MFP3D Force Modify Panel

This panel will let you Modify your force data. Right now it is still very much a work in progress...

Sub Toc

Force Restore Buttons
Force Modify Panel Setup



Force Undo Titles

These titleBoxes describe the actions of the Force Undo Buttons and Force Restore Buttons. There are 3 columns, from left to right, Last one, selected, and All. The last one acts on the last modified force plot (which is not changed when multiple force plots are modified at once). The titlebox lists the force plot that will be modified. The Selected buttons act on the force plots selected in the Force Plot List. The All column acts on all the force plots in memory.

Force Undo Buttons

These buttons will toggle between undo and redo. They undo all of the last TYPE of change. i.e. you change the InVOLS on a force plot 20 times. Clicking undo last, will undo all the InVOLS changes and revert the force plot to it state before any InVOLS changes have been made. The button will then change to Redo. Clicking on that will put your force plot back to where it was after the 20 InVOLS changes. The three different undo / redo buttons are described in Force Undo Titles. When working on multiple force plots, and some of them have been undone, and some redone, the button title will say Undo, and when going through the force plots will not alter the force plots that have already been undone. At that point all the force plots that button acts on will have been Undone, and the button title will state Redo.

Force Restore Buttons

These 3 buttons will restore the specified (see Force Undo Titles) force plots to their "Original" state. Original is defined in this context as the state on the hard drive. However if the force plots were saved directly to memory, then when you first modify that force plot a copy is stored in the root:ForceCurves:Restore folder, and that is considered the Original state.

another topic

It might look good

Force Modify Panel Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

• MFP3D Force Analyze Panel

This panel will let you Extract parameters from the force curves. It still is not really well planned out just how far this tab will go....

Sub Toc

A topic

And then look for it in the Help Browser

Another Topic

So I say again, cut and paste your topic and sub topics

Force Analyze Panel Setup

This button allows you to select which control lines will be shown, and allows you to pick a color for the border. These parameters will not be saved unless you have:

1) Set your user path (this would have to be done before the changes were made)
2) Then called SaveParms() from the command line.

What that does is saves the parameters in your user directory, so that the next time you load up your user parameters, and then rebuild the panel, it will be the way you like it.

- **Gizmo Control Panel**

NOTE: READ THIS!!!!!!!!!!!!!!!!!!!!! (IGNORE AT OWN RISK)

The Gizmo Surface renderer allows you to render data as a 3D surface. It also lets you rotate this surface, and when you let up the mouse if there is any lateral movement of the mouse it will continue to spin the surface indefinitely. This will Rail your CPU and make it **VERY, VERY** difficult to even close Igor. What you **MUST** do is to be very patient, Right click on the Gizmo Plot, and wait for the menu to appear (this can take some time), then select the stop rotation option, and you will have Igor back in your control. See also Mouse Up Stops Rotation.

Getting Started:

Setting the Image, from the Listpanel (MFP IP menu, or the Gizmo Control Panel Surface Image button), select the image you want to see. Then from the popup menu on the displayed image you can set the Surface Image or the Color Image. This takes a copy of the currently displayed image layer, so do all your mods, axes zoom, and layer selection first. Once you have your data set then you can click on the Plot Surface button of the Gizmo Control Panel to actually render the Surface. You need to be aware of live update as you change parameters on the Gizmo Control panel if you have plotted your surface, this can greatly slow down your machine and eat up lots of memory that you may not get back. If the memory usage gets absurd, save the experiment as a packed experiment, close igor, then open up the packed experiment you just saved. You memory that Gizmo was waisting will have been recailmed.
See also Demo

Note: When you set the Surface Plot data from the Offline images, only the visible range is used. The data beyond the axis limits is cropped off. So to zoom into a section of your image with the gizmo plot simply select that region of the image before clicking on the Set Surface Data command. You should then keep those axis limits if you are going to use a different data channel for the colormap.

COLORBAR BUG

Despite our best efforts, the colorbar in the Gizmo plot has failed MANY more times than it has worked, and as such as been disabled.
The alpha version of Gizmo that this interface was written for does not display text correctly (and I can not figure out the igor command around it). To get the text to show when a colorbar is plotted, simply double click on lowLimtString (from the GizmoInfo panel), click on font, and then select any size, click OK, and then click on again. you my have to rotate the image a little to get all the text to show.

Surface Image

This is the image that will be used to generate the surface map. It will be interpolated down to the number of points defined by Interpolate X and Interpolate Y. The Image is selected from the Display window and will use the data in it's modified form at the point it is selected, as a copy of the wave is stored in root:Images:Gizmo Gizmo is the wavemetrics term for the openGL interface to the surface plot renderer used here.

Clicking on the Surface Image button brings up the List panel (also accessed from the MFP IP menu).

Color Image

You can use a second image to define a colormap. For example you can use a phase image to define the colormap of your height data. There is a very nice example of this shown by the demo button.

ColorMap

If you do not use a second image as the colormap you can select from Igor standard colormaps.

ColorBar

NOTE: this has been disabled as it screws up the axis limits on the Surface 99% of the time. We will revisit this to see if we can't work around the problems with Gizmo in the future.
This puts a non-rotating colorbar off to the right of the plot. There is a bug with the gizmo.xop that we are working on, but as of now the text will not initially render with the colorbar. There are more details at the top of this file. In order to get the text to render, you must double click on LowLimitString in the display list of the Surface Info panel. Then click on font..., select any font size, click on OK, and then click on OK on the next window. This will then render the colorbar strings and (if plotted) the axis labels.

Interpolate X

This defines the number of points allowed in the X axis of the displayed Image. Right now it is limited to 128 as the openGL renderer is VERY slow at higher number of points.

Interpolate Y

This is the same as the X interpolate except in the vertical direction.

Background Color

This is the Backgroundcolor for the Surface Plot, not the background color of the panel. If you want to change the background Color of the panel you can do that through the setup checkBox.

Plot Surface

This does the actual Surface plot. Once this has been initialized it will then look to see if Live Update is selected, to see if it will update the Surface plot every time a parameter is changed. The Light controls work a bit differently with Live update, see Edit Gizmo Lights.

Surface Info

Opens the openGL interface to the Surface Plot. When the figure is made, there is a new menu labeled Gizmo. You can open the same interface by selecting GizmoInfo. This interface has all of the controls to the openGL renderer, and so you can customize your surface plots even further. If this window ever gets maximized in Igor, you have to close it with Ctrl + F4.

Demo

Plots a sample of the Surface Plot controls. See Demo 1 and Demo 2 below.

Demo 1

This data was collected by Rachel Segalman and Alexander Hexemer at UCSB in the labs of Professor Ed Kramer. It is an A-B coblock polymer showing very nice phase segregation. The surface data is the Height data, while the colormap is defined by the Phase image, showing that the worm-like ridges have a clear phase contrast.

Demo 2

This data was collected at the University of Pittsburgh in the Walker Labs. It is a composite material of poly-dimethyl siloxane with a filler of calcium carbonate. The height features correspond to the harder calcium carbonate, which show a marked decrease in adhesion force (colormap).

Axes

If you want axes to be plotted for your surface, then select this checkbox.

Aspect ratio 1

Selection this checkBox forces the surface to have an aspect ratio of 1. That is the Z axes is in proportion to the X and Y axes.

Z scale

Setting this value less than one compresses the Z scale of the Surface and axes, setting it greater than one expands it.

Grid

Selecting this checkbox puts a grid of the selected Grid Color on the plotted axes.

Grid Color

Defines the Color of the grid for the Axes

Fix Labels

This uses an in-exact algorithm to rotate the text to look better. Not perfect, but it gets you pretty close. Simply click it after you rotate the Surface to get your labels visible again.

Labels

If you want tickmarks and text labels for the axes then this is what you want to select. Note that the openGL treats the text as 3D objects, and so when you rotate the Surface, the text is also rotated. You can then rotate the text labels relative to the axes by using the Text Rotate tools (X, Y and Z).

See Angles

This button will rotate the label text through 180 degrees in 10 degree increments. The way the openGL interface is set up the text is rotated through the 3 axes in proportions to the angles defined in X rotate, Y rotate and Z rotate. That is if X rotate is 50, Y rotate is 100, and Z rotate is 0, This button will rotate the text twice as much in the Y axes then the X axes, while not rotating in the Z axes.

Text Rotate

These 3 controls (X, Y, and Z rotate) will rotate the label text in 3 dimensions. Units are in degrees.

Flip X

There are 4 axes the X ticks can appear on, 2 on the top of the cubic axes, and 2 on the bottom. This control will flip which of the lower axes the ticks and labels appear on. If you want ticks and labels on more of the axes you can do this by means of the Surface Info button, then right clicking on the axes in the middle column, and selecting edit.

Flip Y

Same as Flip X, but works the Y axes.

Z Tick

This is the same principle as the Flip X, but allows you to select which of 4 vertical edges of the cubic axes to place the Z ticks on.

Add Light on Initialization

If this checkbox is selected then the light that is described by the controls will be created when the gizmo plot is made.

Add Light

This adds the light described by the controls to the Current Gizmo Plot

Delete Light

This removes the light selected in the Edit Gizmo Lights.
Minor Bug: The gizmo renderer will not always remove the Light until after the gizmo panel is made the topmost panel.

New Light

The Lights listed in the PopupMenu are templates, storing the various parameters for lights. You can create a new one by entering a name in the box to the left and clicking this button. The light can be edited (if write protect is NOT selected). You can also add / edit / remove Lights manually by using the Programming>Global Variables>Surface Lights menu. These waves are saved every time the panel is closed in the symbolic path User (defined using the Programming menu). So if you are using multiple Users, you have separate light temples for each user.

Delete Template

This removes the currently selected light template from the database.

Light Template

This popup selects the predefined light template and sets the light parameters to that template. See New Light (above).

Write Protect

Selecting this makes it so that the changes to the light's properties does not alter the template. It is intended to give Users a single level of protection from other Users altering Light templates. If many Users are using this you may want to set up separate users in the programming menus. See New Light (above)

Ambient Light

This defines the color of the Ambient Light of the Light. Ambient light is a directionless light that shines uniformly on the objects.
See also Edit Gizmo Lights

Diffuse Light

This defines the color of the Diffuse (dull) Light of the Light.
See also Edit Gizmo Lights

Specular Light

This defines the color of the Specular (shiny) Light of the Light. You need a little ambient light to see any Specular or diffuse light.
See also Edit Gizmo Lights

Light Dir. Z

This is the angle of the light direction in the Z (Elevation) axes.
See also Edit Gizmo Lights

Light Dir X-Y

This is the angle of the light direction in the X-Y plane (Azimuth). Currently this is forced to 0 inside the code (wire is not connected yet).
See also Edit Gizmo Lights

Spot Light

NOTE: There is a bug in the current XOP, so that values from 0.5 < 1 are errors, so they are all treated as 0.49
You can limit the amount of the surface area covered by the light object. This value corresponds to the fraction of the total surface area that is illuminated. If you do not want a spot-light effect, this value should be 1. A value of 1 corresponds to a value of 180 for the light cuttoff in the Gizmo info panel (Surface Info).
Try using a very small value for the spot light, such as 0.03, and a Light Dir. Z of about 10 degrees, with a blur edges of about 1.
See also Edit Gizmo Lights

Blur edges

This controls how sharp the edges of the spot light are. Higher numbers make for a sharper spot light. This value directly corresponds to the exponent value for the light in the Gizmo info panel (Surface Info).
See also Edit Gizmo Lights

Edit Gizmo Lights

Setting this control to Build Templates will have the light color controls (Ambient Light, Diffuse Light, and Specular Light), as well as the Light Dir. Z, Light Dir X-Y, Spot Light, and Blur edges controls working as described above. However if one of the rendered lights are selected then those controls will be set to the values of the rendered light. If live update is set then the controls will change the properties of the rendered light.

Preset Views

This has a set of view angles stored for easy viewing. It can also remember the current view by selecting store user views (this is only stored for this experiment). That view can be recalled by selecting the corresponding user view menu that was created in this popup. These user views can be cleared by selecting the clear user views popup menu.
Advanced note:
The view data is stored in Root:Images:Gizmo:UserViews, you can save this wave to the hard drive and then load it back into the Root:Images:Gizmo dir. to keep old views around. The easiest way to update the controls to see the loaded views is to kill the Gizmo Control Panel and then rebuild it.

Gizmo X View

This sets the view angle for the "X" axis, the X-Y plane. See also Show Axis Cues.

Gizmo Y View

This sets the view angle for the "Y" axis, the Y-Z plane. See also Show Axis Cues.

Gizmo Z view

This sets the view angle for the "Z" axis, the Z-X plane. See also Show Axis Cues.

Show Axis Cues

This renders a set of axes markers in the middle of the gizmo plot so that you can see which axis is which. Very helpful for keeping your orientation in mind, but alters the way Gizmo renders Lights.

Stop Rotation

This will make it so that mouse can not rotate the surface.

Fix Labels After Rotation

When this checkbox is selected, after the surface is rotated the axes labels will be rotated to correspond to provide easy viewing. This can take up a fair amount of CPU.
See also Fix Labels

Gizmo Zoom

This lets you zoom into the Gizmo panel, 0 is normal view, -10 is Zoomed in, +10 is zoomed out.

Mouse Up Stops Rotation

When this checkbox is selected it attempts to stop the rotation when the mouse button is let up. Does not work perfectly, but it stops some of the annoying autorotations of the Gizmo Surface plot, which can pin your CPU and make it extremely Difficult to do anything with Igor.

Zero Plane

If this is selected a flat plane (X-Y) will be rendered. This may be useful in highlighting certain height features. Minor Bug: The gizmo renderer will not remove the zero plane after this checkbox is deselected until after the gizmo panel is made the topmost panel.

Opaque

This will define if how transparent the zero plane is. A value of 1 is opaque, while a value of 0 is hardly visible.

Offset

This will apply an offset to the zero plane in the z axes.

Live Update

After the Surface Plot has been initially made, and this checkbox is selected, then changes to the various parameters will automatically update on the Surface Plot. This may take an enormous amount of CPU.

Getting Started

This button simply opens up this help file at the beginning.

Setup

Selecting this will allow you to choose which controls will show up under the selected tab (it can be done individually for each tab). You can also select the backgroundcolor of the panel under this control.

MFP3D.xop - Interface for the Asylum Research MFP-3D

Build 21 up1, 2003-07-18

[Please note: If you have old code that no longer runs because I've deleted some functions, you can probably make it work again by selecting "Deprecate" from the Asylum Research logo button in the status bar. This item will insert Igor functions into the Main Procedure Window that will emulate most of the deleted functions. This backwards compatibility is provided so that you may run older saved experiments without having to completely rewrite everything to the new specification. However, any new code you write should be written to the new spec as there is no future guarantee of deprecation functions.]

The MFP-3D XOP contains drivers for the MFP-3D controller plus support for the ATI Radeon All-In-Wonder video display and capture card. Other video capture cards may or may not work.

MFP-3D Controller Introduction

The MFP-3D controller connects to the computer with a USB cable. If the controller is on and connected when Igor is booted, the MFP-3D Xop will reset it and download DSP code to it. After this download process, the controller is fully initialized, and the status bar in Igor will reflect this. If the controller is not found, the status bar will indicate "Disconnected". The controller may be connected and disconnected or power cycled at any time and the MFP-3D Xop will again reset and download the code when appropriate.

All MFP-3D Xop functions begin with "td_", which is an abbreviation for 3D (functions cannot start with numbers). These two letters also happen to be my initials.

Device, Group, and Parameter Reference

Previous versions of this Xop had many different functions for manipulating device parameters. These functions would be different for each type of device. This system was quickly getting out of control as more devices were added. So a new system has been created that provides a consistent interface to all parameters on all devices.

Before I continue, some definitions are required. A **parameter** is simply a value that has a name. These parameters can stand for an input (A/D on the controller), an output (D/A on the controller), or a control (a gain in a feedback loop). A **group** is simply a collection of parameters. A **device** is a piece of hardware that is controlled by the Xop (the controller or the head).

The idea behind the new system is that you can use the same functions to access a parameter no matter which group it is in or what device it is on. Each access function takes an "address" as an argument. The addresses look like email addresses - **"Parameter%Group@Device"**. For example, to specify the laser sum in the head, one would use "Sum%Laser@Head". The X sensor input on the controller becomes "X%Input@Controller".

To prevent this from becoming tedious, there are some shortcuts. If you do not specify the device in your address, the root device is assumed. The controller is our "root" device - i.e., all other devices like the scanner and head are connected to the controller. Therefore, "X%Input@Controller" can be shortened to "X%Input". Another shortcut is that each device has a "default" group which is unnamed. This means that it is actually not possible to specify the group by name when refering to a parameter in the default group. For example, to get the temperature from the head, use "Temperature@Head". To read the default group from the scanner, use "@Scanner".

There is another small quirk that appears when more than one group or device has the same name. The first one encountered when scanning for devices gets its name as normal. The second one will get a '1' appended to the name. The third gets a '2', and so on. So in a particular device chain, you might see "Head", "XYLVDT", "Heater", "Heater1".

Device Names

Here is a list of the devices you are likely to encounter.

Controller - The MFP-3D controller. This is the "root" device. All other devices are linked to the Xop via this device. Contains all of the high-speed inputs and outputs.

XYLVDT - This board contains the electronics that drive the X and Y LVDT sensors in the Scanner. It also controls video magnification in Stand-Alone bases.

Scanner - This board contains a Flash memory device for recording the scanner sensitivities and a temperature sensor to help track thermal drift.

Head - A combination of both XYLVDT and Scanner, but only for one axis, this board provides A/Ds for tracking the laser target and a method for discovering which type of fluid cell has been snapped in.

Unknown - This device only appears when an unprogrammed device is found, or a device that is not compatible with this particular Xop.

Group Names

Each of the devices above have a default group, which is unnamed. The following is a list of some additional groups.

Input@Controller - Contains all of the A/Ds and some parameters that control these A/Ds. It also contains some "calculated" data, like **r** and **theta** which come from **i** and **q**, as well as a digital input registers. All of the data in this group passes through a bandwidth filter (see below).

Raw@Controller - The unfiltered version of the above signals.

Output@Controller - Contains all of the D/As, as well as a digital output register.

DDS@Controller - Contains parameters that control the DDS output.

InputBandwidth@Controller - Controls the bandwidth of the **Input** group signals.

PISLoop@Controller - Controls the PIS feedback loop. There are three PIS loops, and thus three groups, named "PISLoop", "PISLoop1", and "PISLoop2".

Crosspoint@Controller - Controls the crosspoint switch.

Event@Controller - Provides user triggerable events.

Hamster@Controller - Interface to Hamster device.

Laser@Head - Contains three values that correspond to the laser target.

InfoBlock@AnyDevice - All devices have FLASH programmable EPROM in which parameters are stored for later retrieval. All info blocks contain hardware version information, but some might also include sensitivities, voltage limits, LVDT drive information, and other factory calibration data. Unless otherwise noted, most parameters default to zero if the InfoBlock contains invalid or unknown data.

Parameter Names

Here is a brief overview of what each parameter means. They will be listed under the corresponding Device and Group names. The following designators might appear in parenthesis to further qualify the parameter.

(RO)	Read Only - the parameter only makes sense being read, not written.
($)	String - the parameter only makes sense with strings - don't use numeric waves or **td_ReadValue**, **td_WriteValue**, etc. Instead, use textWaves or td_ReadString, td_WriteString, etc.

Controller

Default

ConnectionStatus (RO) - Returns the connection status of the controller.

0	Connected
1	Not Connected
2	Firmware Failure

ConnectionStatusCallback ($) - This parameter will cause a user-supplied function to be called whenever the connection state of the controller changes. Use this parameter in conjunction with **ConnectionStatus** to reset your software whenever the user resets the controller. An expression within the supplied string will be executed whenever the controller status changes. This will typically be the name of a user function (e.g., "MyCallback()"). Of course, if the controller state never changes, the callback function will not be called. If you want to discontinue notification, simply set this parameter to an empty string (e.g., "").

STFCTriggerCount (RO) - Returns the number of 100kHz sample count where the STFC Trigger went off. See td_SetSTFC.

Input

X, **Y**, **Z**, **A**, **B**, **Fast**, **i**, **q** (RO) - Correspond to the bandwidth-limited (filtered) A/D inputs (see the **InputBandwidth** group).

X	Sensor on X-axis
Y	Sensor on Y-axis
Z	Sensor on Z-axis
A	A/D A
B	A/D B
Fast	Fast A/D
i	Down-converted in-phase component
q	Down-converted quadrature component

r, **theta** (RO) - Calculated from **i** and **q**, and thus inherit any bandwidth limit applied to those parameters.

r	Magnitude developed from filtered **i** and **q**
theta	Phase developed from filtered **i** and **q**

FastGain, **FastOffset** - Control the Fast A/D.

FastGain	Hardware gain applied to the **Fast** A/D input, specified in dB. Allowed gains are 0dB (1x), 3dB (1.41x), 6dB (2x), 12dB (4x), 15dB (5.62x), 18dB (8x), and 20dB (10x). If you change this while collecting data from the Fast A/D, you might notice a slight glitch in the middle of the wave. Note that **i**, **q**, and **r** are also affected since they are all ultimately calculated from data collected by the Fast A/D.
FastOffset	16bit signed offset which is added to the Fast A/D input. Used mostly for zeroing residual offsets to improve the accuracy of the **i** and **q** quadrature inputs.

Digital - User digital inputs. Currently, there is no way to use these without opening up the controller.

Raw

X, **Y**, **Z**, **A**, **B**, **Fast**, **i**, **q** (RO) - Correspond to the unfiltered A/D inputs. See the **Input**

group for more details.

Output

X, **Y**, **Z**, **A**, **B**, **C** - Correspond to the D/A outputs.

X	Piezo drive on X-axis
Y	Piezo drive on Y-axis
Z	Piezo drive on Z-axis
A	D/A A, filtered to 52kHz
B	D/A B, filtered to 52kHz
C	D/A C, filtered to 78Hz (yes, that is Hz, not kHz)

DDS

Freq, **FreqOffset**, **Phase**, **PhaseOffset**, **Amp**, **AmpOffset** - Control the DDS output.

Freq	Center frequency - the main output frequency.
FreqOffset	Offset frequency - this plus the center frequency add to the output frequency. Useful for frequency modulation. Should be set to zero when not used.
Phase	Phase of output frequency relative to reference frequency. Only useful when looking at **i**/**q** or **r**/**theta** from the lock-in downconverter.
PhaseOffset	This plus the phase parameter add to define the phase of the output signal relative to the reference signal for the lock-in. Useful for phase modulation. Should be set to zero when not used.
Amp	Peak amplitude of the main DDS output.
AmpOffset	This plus the amplitude parameter add to make up the peak amplitude of the output signal. Useful for amplitude modulation. Should be set to zero when not used.

InputBandwidth

X, **Y**, **Z**, **A**, **B**, **Fast**, **i**, **q** - Correspond to the bandwidth parameter for the named channel. The value for the parameter is the frequency in Hz of a two-pole IIR lowpass filter. There is no way to defeat the filter. For unfiltered data, see the **Raw** group. Every filtered channel defaults to 100 kHz upon controller reset. Note that in order for **r** and **theta** to make sense, **i** and **q** should always be set to the same bandwidth.

PISLoop[n]

Setpoint, **PGain**, **IGain**, **SGain** - Correspond to the same parameters that are used in td_SetPISLoop. Basically, they allow you to update the feedback loop parameters after the feedback loop has already started.

Setpoint	Subtracted from the input signal to get the error signal. Since the units of this parameter are derived from the input signal, setting this parameter will not work correctly until the input is defined by td_SetPISLoop.
PGain	Proportional gain. You'll find this mostly useless for controlling piezos.
IGain	Integral gain
SGain	Secret gain... *shhhh!* Also known as double integral gain.

SetpointGain and **SetpointOffset** - These are modifiers to **Setpoint**. They are only useful if you are driving **Setpoint** with a wave. These values are set to 1 and 0, resepectively, whenever td_SetPISLoop is called.

SetpointGain	Multiplied by the setpoint to get the actual setpoint subtracted from the reference signal.
SetpointOffset	Added to the setpoint after the above gain is applied to get the actual setpoint subtracted from the reference signal. Since the units of this parameter are derived from the input signal, setting this parameter will not work correctly until the input is defined by td_SetPISLoop.

MungeCount, **MungeLength**, and **MungeAlpha** require Yoda skill level. And perhaps a Vulcan mind meld.

Damping NEW! - Controls an active damping loop based on the FastADC and DDS. Ask Dan for more info.

Gain - Ask Dan.

Angle - Ask Dan.

Crosspoint - Each of these parameters corresponds to one of sixteen crosspoint switch outputs. Each parameter can be set to one of fifteen different values, which correspond to the crosspoint switch inputs. You can set the crosspoint switch one parameter at a time using td_WriteString, but since each individual update forces an update of the entire crosspoint switch, it is recommended that you set up an entire text wave and then use td_WriteGroup to update the entire crosspoint switch once.

InA	Connected to **A**%Input
InB	Connected to **B**%Input
InFast	Connected to **Fast**%Input
InAOffset	Summing junction attached to **A**%Input
InBOffset	Summing junction attached to **B**%Input
InFastOffset	Summing junction attached to **Fast**%Input
OutXMod	Summing junction attached to **X**%Output
OutYMod	Summing junction attached to **Y**%Output
OutZMod	Summing junction attached to **Z**%Output
FilterIn	Input to the 36kHz filter
Out0	Attached to BNC Output 0
Out1	Attached to BNC Output 1
Out2	Attached to BNC Output 2
PogoOutCurrent	High current drive (need better explanation)
PogoOut1	(need better explanation)
PogoOut2	(need better explanation)

Each of the above parameters can be set to one of the following values, which correspond to a crosspoint switch input.

OutA	Connected to **A**%Output
OutB	Connected to **B**%Output
OutC	Connected to **C**%Output
Defl	Deflection signal from Head
ACDefl	A/C coupled deflection signal from Head
Lateral	Lateral signal from Head
In0	Attached to BNC Input 0
In1	Attached to BNC Input 1
In2	Attached to BNC Input 2
Ground	Take a guess
FilterOut	Attached to output of the 36kHz filter
PogoIn0	(need better explanation)
PogoIn1	(need better explanation)
DDS	Connected to the DDS output
Off	Turns off the crosspoint output by killing buffer

Event - Each of these parameters correspond to a different event. These events can be used to trigger waves, feedback loops, etc. They are handy for synchronization of different functions.

Always, **Never** (RO) - Constants.

Always	Always set.
Never	Always clear.

0, **1**, **2**, **3**, **4**, **5** - User programmable. Can be set to one of the following values.

Clear	Clear the trigger. Will stop any wave any wave dependent on it from starting again, but *will not stop any waves already in progress*. Use td_StopInWaveBank, td_StopOutWaveBank, or td_Stop to stop waves already in progress.

Set	Set the trigger. Will allow any wave dependent on it to start running.
Once	Sets the trigger, allows one data collection cycle to pass, then clears the trigger. This should allow any waves dependent on the event to run only once.

Hamster

StatusCallback ($) - This parameter will cause a user-supplied function to be called whenever the hamster wheel is rotated or hamster button is pressed. Use this function in conjunction with **Degrees** and **Switch** to respond to user manipulation of the hamster. An expression within the supplied string will be executed whenever the hamster status changes. This will typically be the name of a user function (e.g., "MyCallback()"). Of course, if the hamster state never changes, the callback function will not be called. If you want to discontinue notification, simply set this parameter to an empty string (e.g., "").

Degrees (RO) - This parameter returns the internal relative degree counter. Every time this function is called, the current running total is cleared.

Switch (RO) - Returns the status of the hamster function switch.

0	Open
1	Down
2	Up

InfoBlock

DateWritten (RO, $) - Date that this InfoBlock was last written.

SerialNumber - Serial number of the Controller.

DeviceRevision ($) - Revision code of the Controller as an entire unit.

BoardRevision ($) - Revision code of the Controller motherboard.

DateBuilt ($) - Build date of the Controller as an entire unit.

Note ($) - Random ramblings of the test engineer.

BackPanelRevision ($) - Revision code of the Controller's back panel board.

FrontPanelRevision ($) - Revision code of the Controller's front panel board.

MinimumVoltage - Smallest or most negative voltage possible on the X, Y, or Z axes.

MaximumVoltage - Largest or most positive voltage possible on the X, Y, or Z axes.

Has1xACDeflGain - Early controllers have an 8x gain applied to the A/C Deflection signal. This was removed during later production.

0	No
1	Yes

HasMDBPullup - Early controllers did not have a pullup resistor on the MultiDrop Bus. This lead to "ghost" devices appearing on the bus.

0	No
1	Yes

HasAlteraHeatsink NEW! - Early controllers did not have a heatsink on the Altera FPGA. Subsequent Altera code causes Altera to get pretty damn hot without a heatsink.

0	No
1	Yes

Is1DPlus NEW! - This version of the controller does not support a scanner.

0	No
1	Yes

XYLVDT

Default

MDBSerialNumber, **MDBBus**, **MDBType** ($), **MDBTypeCode**, **MDBVersion** ($), **IsBlank**, **HasJumper** (All RO) - Internal parameters used by the Xop to manage the device. They are only exposed in this interface for debugging purposes.

XPhase and **YPhase** CHANGED! - Phase of the LVDT drive signal relative to the reference for the lock-in, for each particular axis. Not generally useful to change this after it has been calibrated at the factory. Setting this value will turn on the oscillator if it was off.

Oscillator - Can turn off the oscillator that drives the LVDT circuitry for the X and Y axes. Might be useful for an ultra-sensitive measurement if the scanner is run open loop.

0	Off
1	On

VideoMagnification - For Stand-Alone bases, selects between the two cameras set at different magnifications.

0	Low
1	High

LEDBlinking - Can disable the blinking light. Might be useful for an ultra-sensitive measurement if you see 1Hz noise creeping into your data.

0	Off
1	On

FLASHMemoryID and **TempSensorID** - (RO, $) Internal parameters used by the Xop to manage the device. They are only exposed in this interface for debugging purposes.

InfoBlock

DateWritten (RO, $) - Date that this InfoBlock was last written.

SerialNumber - Serial number of the XYLVDT.

DeviceRevision ($) - Revision code of the XYLVDT as an entire unit.

BoardRevision ($) - Revision code of the XYLVDT mainboard.

DateBuilt ($) - Build date of the XYLVDT as an entire unit.

Note ($) - Random ramblings of the test engineer.

HasZoom - Indicates whether or not the system using the XYLVDT takes advantage of the LowMag/HiMag video switching capability. For example, Stand-Alone bases use it, Inverted-Optical systems do not.

0	No
1	Yes

CameraType - Indicates the type of video camera if zoom is available.

0	NTSC
1	PAL

BaseType NEW, CHANGED! - Describes the type of base in which the XYLVDT board has been installed. "SA" means "Stand Alone base" and "IO" means "Inverted Optical microscope".

0	*None* - board has been sold to someone working on their own custom-built system
1	*SA - Bottom View*
2	*SA - Top View*
3	*SA - Dual View*
4	*SA - No View*
5	*IO - Unknown Microscope*
6	*IO - Nikon TE200/300*
7	*IO - Nikon TE2000*
8	*IO - Olympus 1x70/71*
9	*IO - Zeiss Axiovert 200*

Scanner

Default

Temperature (RO) - Returns the temperature in Celsius degrees as read from inside the Scanner. Please note that it takes approximately 3/4 of a second to make a temperature reading, so what you get in return is actually the temperature from the last time this call was made. After returning the last temperature, the sensor takes a new reading for the next time this parameter is read. Therefore, the very first time you read this parameter, the value returned should be discarded. The acuracy of the temperature readings will generally increase if you refrain from reading this parameter more than once per second.

InfoBlock

XPhase - Phase offset in the drive signal to the X-axis LVDT for best sensitivity (degrees).

YPhase - Phase offset in the drive signal to the Y-axis LVDT for best sensitivity (degrees).

XDiffAmpGain - Not currently used.

YDiffAmpGain - Not currently used.

DateWritten (RO, $) - Date that this InfoBlock was last written.

SerialNumber - Serial number of the Scanner.

DeviceRevision ($) - Revision code of the Scanner as an entire unit.

BoardRevision ($) - Revision code of the Scanner mainboard.

DateBuilt ($) - Build date of the Scanner as an entire unit.

Note ($) - Random ramblings of the test engineer.

ScanBoardRevision ($) - Revision code of the Scanner voltage distribution board.

XLVDTSens - Sensitivity factor for the X-axis LVDT (um/V).

YLVDTSens - Sensitivity factor for the Y-axis LVDT (um/V).

XPiezoSens - Open-loop sensitivity factor for the X-axis piezo (um/V).

YPiezoSens - Open-loop sensitivity factor for the Y-axis piezo (um/V).

XIGain - Optimum closed-loop integral gain factor for X-axis feedback.

YIGain - Optimum closed-loop integral gain factor for Y-axis feedback.

XSGain - Optimum closed-loop double-integral gain factor for X-axis feedback.

YSGain - Optimum closed-loop double-integral gain factor for Y-axis feedback.

PiezoType - Not currently used.

SafeMinimumVoltage - The minimum or most negative voltage that can be safely applied to either the X or Y piezos (V).

SafeMaximumVoltage - The maximum or most positive voltage that can be safely applied to either the X or Y piezos (V).

MaxScanSize - Approximately the largest non-rotated image scan possible (um).

HasDamping NEW! - Indicates whether or not the scanner has damping material applied.

0	No
1	Yes

Head

Default

MDBSerialNumber, **MDBBus**, **MDBType** ($), **MDBTypeCode**, **MDBVersion** ($), **IsBlank**, **HasJumper** (All RO) - Internal parameters used by the Xop to manage the device. They are only exposed in this interface for debugging purposes.

ZPhase CHANGED! - Phase of the LVDT drive signal relative to the reference for the lock-in. Not generally useful to change this after it has been calibrated at the factory. Setting this value will turn on the oscillator if it was off.

Oscillator - Can turn off the oscillator that drives the LVDT circuitry for the Z axis. Might be useful for an ultra-sensitive measurement if the head is run open loop.

0	Off
1	On

Relay - Even when the Z axis drive signal is set to zero using the Z D/A output, there is enough electronic noise in the amplifier stage such that the Z piezo will move around a little bit, thus making thermal cantilever measurements impossible. This relay will open the path between the drive signal and piezo. Note that it should be closed for normal operation.

0	Open
1	Closed

LEDBlinking - Can disable the blinking light. Might be useful for an ultra-sensitive measurement if you see 1Hz noise creeping into your data.

0	Off
1	On

PogoID (RO) - Used to identify the type of fluid cell module that has been attached. *These have not been defined yet.*

0	None (ie, nothing plugged in)
1-255	Unknown

Temperature (RO) - Returns the temperature in Celsius degrees as read from inside the Head. Please see the description of this parameter in the **Scanner** section for more information about making readings.

TempSensorID (RO, $) - Internal parameter used by the Xop to manage the device. This is only exposed in this interface for debugging purposes.

Laser - This section refers to a laser quadrant detector that looks like this. I apologize for the crude ASCII graphics.

```
+-+-+
|A|B|
+-+-+
|C|D|
+-+-+
```

Sum (RO) - The sum of the laser light hitting all four quadrants of the detector. Also known as A+B+C+D. This parameter is a good indicator of crude laser alignment. **Sum** is also used to normalize the **Difference** and **Lateral** signals.

Difference (RO) - The vertical component of the laser position. Also known as (A+B) - (C+D). This is not the same as Deflection, which is normalized by the **Sum**. That is to say, Deflection is **Difference** divided by **Sum**.

Lateral (RO) - The horizontal component of the laser position. Also known as (A+C) - (B+D). In this case, the parameter is poorly named, as Lateral is also used as the name of the **Sum** normalized version of this signal.

InfoBlock

ZPhase - Phase offset in the drive signal to the Z-axis LVDT for best sensitivity (degrees).

ZDiffAmpGain - Not currently used.

DateWritten (RO, $) - Date that this InfoBlock was last written.

SerialNumber - Serial number of the Head.

DeviceRevision ($) - Revision code of the Head as an entire unit.

BoardRevision ($) - Revision code of the Head mainboard.

DateBuilt ($) - Build date of the Head as an entire unit.

Note ($) - Random ramblings of the test engineer.

LaserBoardRevision ($) - Revision code of the laser control board.

ZLVDTSens - Sensitivity factor for the Z-axis LVDT (um/V).

ZPiezoSens - Open-loop sensitivity factor for the Z-axis piezo (um/V).

PiezoType - Not currently used.

SafeMinimumVoltage - The minimum or most negative voltage that can be safely applied to the Z piezo (V).

SafeMaximumVoltage - The maximum or most positive voltage that can be safely applied to the Z piezo (V).

ZMaxScanSize - Approximately the largest force curve scan possible (um).

LaserType - Not currently used.

ZIGain - Optimum closed-loop integral gain factor for Z-axis feedback.

ZSGain - Optimum closed-loop double-integral gain factor for Z-axis feedback.

• MFP-3D Controller Function Reference

Please read the Device, Group, and Parameter Reference before proceeding.

td_XopVersion()

Returns a string containing the version number of the MFP3D Xop. Useful for checking to see whether the correct Xop is loaded before proceeding with certain Igor code. This version number also appears in the Igor title bar.

Function Result

Current version string in the form of "NN" or "NNpre [date]", where
NN is the build number (might be more than two digits)
date is the date of the build of pre-release versions

Example

```
if (str2num(td_XopVersion()) < 20)
   Abort "MFP3D.xop version is stale!"
endif
```

td_ControllerReset()

Stops all controller activity, resets the controller, and unsets any associated Igor waves. Note that the Xop will automatically upload software to the controller after reset. This function should not be used unless communication with the controller appears to be lost. If this does not work, you will have to power cycle the controller.

Function Result

0	Successful
1	No controller found or controller is seriously hung

See Also td_Stop, td_SoftReset

td_ScanForDevices()

Causes the Controller to scan all of its busses to check for device changes.

Function Result

0	Successful
1	Controller not found

See Also td_GetDevices

td_GetDevices(textWave)

Returns a list of all devices currently connected in the MFP-3D system. Please note that this returns the results from the last scan. If devices have been unplugged since the last scan, the results of this function will be inaccurate. You probably want to call td_ScanForDevices before using this function.

Parameters

wave	A text wave in which you want to store the devices. The row label contains the name of the device as it is known on the bus and the string value contains the type of the device. These two are often the same unless you have multiple devices of the same type on the bus.

Function Result

0	Successful
1	Controller not found
6	A non-text wave was specified

See Also td_ScanForDevices

td_ReadValue(parameterName)

Returns the value of the specified parameter, making a trip to the device on which the parameter is located if need be.

Parameters

parameterName	String containing the parameter name. See Parameter Names for more information.

Function Result

Value of the parameter in its native units. These units may be discovered using a function which I have not yet made. A value of NaN will be returned if the parameter name is bad or the device on which it resides is disconnected.

See Also td_WriteValue, td_ReadString

td_RV(parameterName)

Alias for td_ReadValue.

td_WriteValue(parameterName, value)

Sets the value of the specified parameter, making a trip to the device on which the parameter is located if need be.

Parameters

parameterName	String containing the parameter name. See Parameter Names for more information.
value	Value of the parameter in its native units. These units may be discovered using a function which I have not yet made. The maximum and minimum

value possible can be found using more functions which I have not yet made.

Function Result

0	Successful
1	Device not connected
4	USB Error
5	Bad parameter name

See Also td_ReadValue, td_WriteString

td_WV(parameterName, value)

Alias for td_WriteValue.

td_ReadString(parameterName)

Returns the value of the specified parameter, making a trip to the device on which the parameter is located if need be. Note that in general, you will want to use td_ReadValue instead of this function whenever it makes the most sense to use values. String parsing is computationally expensive as well as not as accurate as using values.

Parameters

parameterName String containing the parameter name. See Parameter Names for more information.

Function Result

A string containing the value of the parameter with its native units appended (if any). A string of "*ERROR*" will be returned if the parameter name is bad or the device on which it resides is disconnected.

See Also td_WriteString, td_ReadValue

td_RS(parameterName)

Alias for td_ReadString.

td_WriteString(parameterName, valueString)

Sets the value of the specified parameter, making a trip to the device on which the parameter is located if need be. Note that in general, you will want to use td_WriteValue instead of this function whenever it makes the most sense to use values. String parsing is computationally expensive as well as not as accurate as using values.

Parameters

parameterName String containing the parameter name. See Parameter Names for more information.

valueString A string containing the value of the parameter. For now, any units appended to the end of the string are ignored, and the conversion uses the native units of the parameter. However, you should add units anyway, as one day, they won't be ignored.

Function Result

0	Successful
1	Device not connected
4	USB Error
5	Bad parameter name

See Also td_ReadString, td_WriteValue

td_WS(parameterName, valueString)

Alias for td_WriteString.

td_ReadGroup(groupName, wave)

Reads a wave full of values or strings (depending on supplied wave type) from the device on which the group resides. It is like calling td_ReadValue or td_ReadString for each parameter in the group. This function is extremely useful for debugging, especially if the resulting wave is then displayed in a table with its row labels.

Parameters

groupName — String containing the group name. See Group Names for more information.

wave — A wave of values or strings. Note that using waves with variable types other than doubles or floats may result in truncated data. For now, anything in the wave before the call is made is erased. In the future, this might not be true. So to "future proof" your code, you should empty the wave before passing it. After the call is made, the row labels of the wave will contain the parameter name and the value or string section will be filled in with appropriate data.

Function Result This function will try to read all the parameters, even if errors are encountered in the middle of updating. The last error encountered, if any, will be the one reported.

0	Successful
1	Device not found
3	Not implemented yet
4	USB Error
5	Group does not exist
22	FLASH memory does not have valid data

See Also td_WriteGroup

td_RG(groupName, wave)

Alias for td_ReadGroup.

td_WriteGroup(groupName, wave)

Writes a wave full of values or strings (depending on supplied wave type) to the device on which the group resides. It is like calling td_WriteValue or td_WriteString for each parameter specified by a row label. This function is extremely useful for debugging, especially if the wave is displayed in a table with its row labels. You can easily type in new values for the various parameters, then call this function.

Parameters

groupName — String containing the group name. See Group Names for more information.

wave	A wave of values or strings. Note that using waves with variable types other than doubles or floats may result in truncated data. Before the call is made, the row labels of the wave should contain the parameter names that you want updated and the value or string section should be filled in with appropriate data. You do not need to specify every parameter in a group. Any row labels which do not match a parameter in the group will be ignored. Any parameters which are read-only will also be ignored.

Function Result This function will try to read all the parameters, even if errors are encountered in the middle of updating. The last error encountered, if any, will be the one reported.

0	Successful
1	Device not found
4	USB Error
5	Group does not exist
23	FLASH memory is locked

See Also td_ReadGroup

td_WG(groupName, wave)

Alias for td_WriteGroup.

td_GetLabel(parameterName)

Returns the full label of the specified parameter. The label is often similar to the parameter name, but not always. The label always includes the spaces, so this function would be handy for dynamically creating dialog boxes for various parameters.

Parameters

parameterName	String containing the parameter name. See Parameter Names for more information.

Function Result

A string containing the label of the parameter. If the parameter doesn't exist, "*ERROR*" will be returned.

See Also td_GetGroupLabels

td_GetGroupLabels(groupName, textWave)

Returns the full labels for all parameters for which row labels have been specified. It is like calling td_GetLabel for every specified parameter in the wave.

Parameters

groupName	String containing the group name. See Group Names for more information.
textWave	A text wave containing row labels for each parameter of interest. After the call, the string will be filled in with the corresponding labels.

Function Result

0	Successful
5	Group does not exist
6	The wave must be a text wave

See Also td_GetLabel

td_GetRange(parameterName)

Returns the range of the specified parameter. This function would be handy for dynamically creating dialog boxes for various parameters.

Parameters

parameterName	String containing the parameter name. See Parameter Names for more information.

Function Result

A string containing the range of the parameter. If the parameter doesn't exist, "*ERROR*" will be returned. The string might look like one of the following examples. If it is empty, that means the parameter cannot be written.

#I32;*min*;*max*	Based on a 32bit integer
#I16;*min*;*max*	Based on a 16bit integer
#I8;*min*;*max*	Based on an 8bit integer
#F;*min*;*max*	Based on a 32bit floating point value
#V;*min*;*max*	Unknown value size
#D*len*	String of length *len* that contains a date
#RA*len*	String of length *len* that contains an alpha revision code (eg, 2.3C)
#RE*len*	String of length *len* that contains an ECO revision code (eg, 2.4.13)
#S*len*	String of length *len*
#X*num*;*str*;*str*...	Indexed strings, of amount *num*

See Also td_GetGroupRanges

td_GetGroupRanges(groupName, textWave)

Returns the ranges for all parameters for which row labels have been specified. It is like calling td_GetRange for every specified parameter in the wave.

Parameters

groupName	String containing the group name. See Group Names for more information.
textWave	A text wave containing row labels for each parameter of interest. After the call, the string will be filled in with the corresponding ranges. See td_GetRange for information on the format of the text value returned.

Function Result

0	Successful
5	Group does not exist
6	The wave must be a text wave

See Also td_GetRange

td_xSetInWave(whichBank, eventString, channelString, wave, callback, decimation)

The exact same function as td_xSetInWavePair, except that only one of the two pairs of the bank are sampled. Please see that function below for more information. Only differences will be noted here.

Bank 2 is a special bank that passes data over the USB bus using 32bit integers. As a result, bank 2 will only accept floating point waves. Banks 0 and 1 use 16bit shorts (even if you specify a floating point wave).

Parameters

whichBank	Same as td_xSetInWavePair.
eventString	Same as td_xSetInWavePair.
channelString	Same as channelStringA in td_xSetInWavePair.
wave	Same as waveA in td_xSetInWavePair.
callback	Same as callbackA in td_xSetInWavePair.
decimation	Same as td_xSetInWavePair.

Function Result

0	Successful
1	Controller not connected
4	USB error
5	Bad channel string
6	Wave must be composed of shorts or floats / Bank 2 can only be floats
7	Bad decimation factor
9	Wave length must be a non-zero multiple of 32
10	Bad event name
11	Bad bank number
12	Wave already in use

See Also td_xSetOutWavePair

td_xSetInWavePair(whichBank, eventString, channelStringA, waveA, channelStringB, waveB, callback, decimation)

Connects two input waves to two physical inputs (channels). Both waves *must* have the same length and that length must be a multiple of 32. This function will start filling both waves with the specified input channel data reduced by the given decimation factor as soon as the first event given in eventString is triggered. If a second event is given, the waves will once again begin for *each subsequent time* that event is triggered after the waves end.

The input waves may be either 16-bit integer or 32-bit floating point. If 16-bit integer, the user may scale the data to the particular channel's specifications which can be found using td_GetRange and td_GetUnits. Please note that the data that travels over the USB bus from the controller is only 16-bit on Banks 0 and 1, so that even if you are supplying 32-bit waves, the granularity of the underlying data is 16-bit. Floating point waves are allowed mostly as a convenience so that the scaling is taken care of automatically. If you would like 32-bit data, please use Bank 2 and read the instructions for td_xSetInWave.

Once a wave is used by this function, it is "locked" and its parameters cannot be modified. Use td_StopInWaveBank in order to "unlock" the waves from the bank. If this function notices that the wave is not being used in a repeating fashion, it will "unlock" the wave after the wave has completed, but this is mostly for convenience and should not be relied upon. Bank 0 has double the bandwidth of bank 1 and 2. Please see the section on *decimation* below.

Parameters

whichBank	There are three banks of input pairs, 0-2. However, Bank 2 may not be used by this function. See td_xSetInWave for more information.
eventString	Can contain one or two events, separated by commands and/or spaces. The first event will be used to determine whether or not to start collecting data. After an entire wave has been collected, all subsequent triggers (not just the second one) are based upon the second event. Events may be any of those listed in the *Event* section of Parameter Names. Note that you do not have to specify the group name, as Event will always be assumed. "0, Always" is an effective substitution for "0%Event, Always%Event".

"Never"	Never trigger. Not too useful as first event. If second event not specified, it defaults to "Never".
"Always"	Always trigger. When used as first event, input waves start immediately. When used as second event, input waves collect data repeatedly once they are triggered.

channelStringA	String containing the input channel name for the first wave of the pair on this particular bank. Channels can be almost any parameter listed in Parameter Names in the *Input*, *Raw*, or *Output* groups. Since inputs can come from many different groups, the group must be specified for each parameter.
waveA	The actual wave to connect to the first input channel. Must be the same length as waveB and that length must be a multiple of 32.
channelStringB	String containing the input channel name for the second wave of the pair on this particular bank. See *channelStringA* for additional guidance.
waveB	The actual wave to connect to the second output channel. Must be the same length as waveA and that length must be a multiple of 32.
callback	This function will be called once the wave has completed.
decimation	Amount to decimate the input waves by. A value of 1 will defeat decimation. Value must be integer and 1 or greater - fractional parts are ignored. You should probably use the bandwidth parameter to insure that aliasing will not occur when decimating. Please note that banks 1 and 2 are low bandwidth banks and can only support a minumum of 2 for the decimation factor.

Function Result

0	Successful
1	Controller not connected
4	USB error
5	Bad channel string
6	Wave must be composed of shorts or floats / Bank 2 not allowed
7	Bad decimation factor
8	Waves are not the same length
9	Wave length must be a non-zero multiple of 32

10	Bad event name
11	Bad bank number
12	Wave already in use

Example

```
// none yet
```

See Also td_xSetInWave

td_StopInWaveBank(whichBank)

Halts any wave output action on the named bank, regardless of event state. Waves will never restart no matter what the trigger is unless they are completely re-setup. Waves that are 16bit integers will be zero-filled in areas beyond which data was collected. Waves that are 32bit floating point will be NaN-filled in areas beyond which data was collected.

Parameters

whichBank	Specifies which bank (0-2) of wave outputs to shut off. Specifying -1 will stop all banks at once.

Function Result

0	Successful
1	Controller not connected
4	USB error
11	Bad bank number

td_xSetOutWave(whichBank, eventString, channelString, wave, interpolation)

The exact same function as td_xSetOutWavePair, except that only one of the two pairs of the bank are driven. Please see that function below for more information. Only differences will be noted here.

Parameters

whichBank	Same as td_xSetOutWavePair.
eventString	Same as td_xSetOutWavePair.
channelString	Same as channelStringA in td_xSetOutWavePair.
wave	The actual wave to connect to the first output channel. See the blurb on interpolation below for some hints on length and repeating waveforms.
interpolation	Same as td_xSetOutWavePair.

Function Result

0	Successful
1	Controller not connected
4	USB error
5	Bad channel string
6	Wave must be floating-point type
7	Bad interpolation factor
9	Wave too long or too short
10	Bad event name
11	Bad bank number

16	Corresponding PI Loop for the specified setpoint has not yet been specified. Call td_xSetPISLoop before setting the setpoint values.

See Also td_xSetOutWavePair

td_xSetOutWavePair(whichBank, eventString, channelStringA, waveA, channelStringB, waveB, interpolation)

Connects two output waves to two physical outputs (channels). Both waves *must* have the same length. This function will start driving both outputs with the specified output waves extended by the given interpolation factor as soon as the first event given in eventString is triggered. If a second event is given, the waves will once again begin for *each subsequent time* that event is triggered after the waves end. The output waves must be scaled to the particular channel's specifications which can be found using td_GetRange and td_GetUnits. Maximum wave length allowed is 87,380 points.

Parameters

whichBank	There are three banks of output pairs, 0-2.
eventString	Can contain one or two events, separated by commands and/or spaces. The first event will be used to determine whether or not to start collecting data. After an entire wave has been collected, all subsequent triggers (not just the second one) are based upon the second event. Events may be any of those listed in the *Event* section of Parameter Names. Note that you do not have to specify the group name, as Event will always be assumed. "0, Always" is an effective substitution for "0%Event, Always%Event".

"Never"	Never trigger. Not too useful as first event. If second event not specified, it defaults to "Never".
"Always"	Always trigger. When used as first event, output waves start immediately. When used as second event, output waves output repeatedly once they are triggered.

channelStringA	String containing the output channel name for the first wave of the pair on this particular bank. Channels can be almost any parameter listed in Parameter Names in the *Output* groups or the Setpoint parameter in the *PISLoop* groups. Since outputs can come from different groups, the group must be specified for each parameter. Please note that there are parameters that can be driven one point at a time using td_WriteValue that cannot be driven by waves.
waveA	The actual wave to connect to the first output channel. Must be the same length as waveB. Only 32-bit floating point waves are accepted. See the blurb on interpolation below for some hints on length and repeating waveforms.
channelStringB	String containing the output channel name for the second wave of the pair on this particular bank. See *channelStringA* for enlightenment.
waveB	The actual wave to connect to the second output channel. Must be the same length as waveA. Only 32-bit floating point waves are accepted. See the blurb on interpolation below for some hints on length and repeating waveforms.
interpolation	Amount to interpolate the output waves by. A value of 1 will defeat interpolation. Value must be integer and non-zero - fractional parts are

ignored. The method used depends on the sign of the interpolation value. A positive value will use cubic spline interpolation and a negative value results in simple linear interpolation.

Edge effects are always problematic for an interpolator. The strategy of this interpolator depends on whether the waveform is repeating or not. This function guesses at the user's intention by looking for a second eventString argument. If there is none, then the wave does not repeat. If there is one, then the wave probably does repeat.

For a non-repeating waveform, the last point is always a problem because the last of the "interpolation" points are really an extrapolation. For this case, this function will repeat the last point and extrapolate to that. This works well for linearly ramping the voltage from one value to another. Cubic spline is less optimal here because there will tend to be a bit of overshoot at the end as a spline curve and not a straight line will be drawn between these last two repeated points. The cubic spline will also end "one point short" of the last true value, where the simple linear method will not.

For a repeating waveform, the first point can be a problem as well. This function's cubic spline looks at the end of the given waveform to solve this problem. The last points are extrapolated by interpolating towards the first point of the wave. This insures smooth transistions between repeated waveform copies. Use cubic spline interpolation for the best results when driving a repeating waveform that is not a triangle wave.

Function Result

0	Successful
1	Controller not connected
4	USB error
5	Bad channel string
6	Wave must be floating-point type
7	Bad interpolation factor
8	Waves are not the same length
9	Wave too long or too short
10	Bad event name
11	Bad bank number
16	Corresponding PI Loop for the specified setpoint has not yet been specified. Call td_xSetPISLoop before setting the setpoint values.

Example

```
// repeating 100 point sine wave on Z output
// repeating 100 point cosine wave on Y output
make/o/n=100 test, test2

// Z & Y both range from -10V to 150V,
//  so add in a bit of offset to prevent clipping
// note that x ranges from 0 to 100 so last point is same as first
test = 50 + 50*sin(2*PI*x/100)
test2 = 50 + 50*cos(2*PI*x/100)

// interpolate these waves by 5
// make sure to examine return value
print td_xSetOutWavePair(0, "Always, Repeat", "Z", test, "Y", test2, 5)
```

See Also td_xSetOutWave

td_StopOutWaveBank(whichBank)

Halts any wave output action on the named bank, regardless of event state. Waves will never restart no matter what the trigger is unless they are completely re-setup.

Parameters

whichBank	Specifies which bank (0-2) of wave outputs to shut off. Specifying -1 will stop all banks at once.

Function Result

0	Successful
1	Controller not connected
4	USB error
11	Bad bank number

td_Stop()

Stops almost all controller activity, clears any active events, and unsets any associated Igor waves. Please note that this function will not stop thermal data collection. Use td_StopThermal for that.

Function Result

0	Successful
1	Controller not connected
4	USB error

See Also td_ControllerReset, td_SoftReset

td_SoftReset()

Calls td_Stop and then proceeds to establish conditions similar to those found after a full td_ControllerReset, without actually rebooting the controller.

Function Result

0	Successful
1	Controller not connected
4	USB error

See Also td_ControllerReset, td_Stop

td_xSetPISLoop(whichLoop, eventString, inChannelString, setpoint, pgain, igain, sgain, outChannelString)

Runs a proportional-integral (PI) feedback loop, with optional double integeral gain. This function will drive the output specified by outChannelString in such a manner that the value of the input specified by inChannelString will approach the value specified by the setpoint. The loop begins as soon as the event given in eventString is triggered. Please note that every time you call this function, the loop is halted and the integral error term is reset. Therefore, if you need to change the setpoint, pgain, igain, or sgain parameters, please use td_WriteValue. Use td_xStopPISLoop to halt the feedback loop.

Parameters

whichLoop	There are three feedback loops available, 0-2.
eventString	Can contain one event only. When this event has triggered, the feedback

loop will begin. Events may be any of those listed in the *Event* section of Parameter Names. Note that you do not have to specify the group name, as Event will always be assumed. "0" is an effective substitution for "0%Event".

"Never"	Never trigger. Not too useful.
"Always"	Always trigger. Feedback starts immediately.

inChannelString	String containing the input channel name for the feedback loop. Any channel listed for use by td_xSetInWavePair may be specified.
setpoint	Setpoint of the feedback loop. Specified in the units of the channel given by inChannelString.
pgain	Proportional gain of the feedback loop. Can be any real number. Note that you might have to use a negative value for all gains to make the loop stable, depending on the relative polarity of the specified input and output channels.
igain	Integral gain of the feedback loop. Can be any real number. Please read the note for *pgain*.
sgain	Double integral gain of the feedback loop. The name "sgain" is short for "special gain". It got this name because we have a natural aversion to the term "di". Can be any real number. Please read the note for *pgain*.
outChannelString	String containing the output channel name for the feedback loop. Any channel listed for use by td_xSetOutWavePair may be specified.

Function Result

0	Successful
1	Controller not connected
4	USB error
5	Bad channel string
10	Bad event name
11	Bad loop number

Example

```
// none yet
```

td_StopPISLoop(whichLoop)

Halts the specified feedback loop.

Parameters

whichLoop	Specifies which feedback loop (0-2) to shut stop. Specifying -1 will stop all feedback loops at once.

Function Result

0	Successful
1	Controller not connected
4	USB error
11	Bad loop number

td_StartThermal(wave, callback)

This function will collect data at 5MHz from the Fast A/D ("Fast%Input"). This function is a bit of a temporary hack until I can "do it right". As such, it has some specific requirements in its use as I haven't had time to fully integrate a checking system to prevent you from hurting yourself. The most basic rule is that once this function is called, no other waves may run or be running. You must call td_Stop before calling this function *the first time*. Once this function is called, you can call it as many times as you want to gather additional data. Before doing any other wave operation, you *must* call td_StopThermal. If you do not heed this advice, it is possible that Igor will crash.

Parameters

wave	Data collected will be stored here. Number of points must be a multiple of 32, and less than or equal to 2,097,152. You may use a 16bit integer wave or a 32bit floating point wave.
callback	This function will be called once the wave has completed.

Function Result

0	Successful
1	Controller not connected
4	USB error
6	Wave must be composed of shorts or floats
9	Wave length must be a non-zero multiple of 32 and less than or equal to 2,097,152.
12	Wave already in use

td_StopThermal()

Halts fast data collection started by td_StartThermal. Please read that function's definition for more information. This function takes the system out of the special "hack" mode needed to collection data at 5MHz and restores it to the normal operating state. It is imperative that you call this function only after starting thermal data collection. If you do not, it is possible that Igor will crash. But please don't call this function gratuitously. Best used sparingly.

Function Result

0	Successful
1	Controller not connected
4	USB error - time for a power cycle!

td_xSetCustom(eventString, inChannelString, outChannelString)

Note! This is an advanced function. You should ignore it until you are more experienced with using this driver. It is intended for internal development only.

Allows passing of arguments to customer DSP interrupt code. Whether the custom code pays any attention is up to the writer of the custom code.

Parameters

eventString — Can contain one event only. When this event has triggered, execution of the custom code will begin. Or not. Events may be any of those listed in the *Event* section of Parameter Names. Note that you do not have to specify the group name, as Event will always be assumed. "0" is an effective substitution for "0%Event".

"Never"	Never trigger. Not too useful.

"Always"	Always trigger. Feedback starts immediately.

inChannelString	String containing the input channel name for the custom code. Or not. Any channel listed for use by td_SetInWavePair may be specified.
outChannelString	String containing the output channel name for the custom code. Or not. Any channel listed for use by td_SetOutWavePair may be specified.

Function Result

0	Successful
1	Controller not connected
4	USB error
5	Bad channel string
10	Bad event name

Example

```
// none yet
```

td_SetCustomFloat(wave)

Note! This is an advanced function. You should ignore it until you are more experienced with using this driver. It is intended for internal development only.

Allows passing of arguments to customer DSP interrupt code. Whether the custom code pays any attention is up to the writer of the custom code.

Parameters

wave	Contains values that will be sent to the DSP and end up in the float section of the custom block. This size of this wave must be 256.

Function Result

0	Successful
1	Controller not connected
4	USB error
6	Wave must be floating-point type
9	Wave not 64 in length

Example

```
// none yet
```

td_StopCustom()

Note! This is an advanced function. You should ignore it until you are more experienced with using this driver. It is intended for internal development only.

Tries its best to halt execution of custom DSP interrupt code. Whether the custom code pays any attention is up to the writer of the custom code.

Function Result

0	Successful
1	Controller not connected
4	USB error

td_SetSTFC(eventString, zStart, zEnd, time, inChannelString, triggerValue, greaterOrLesser, callback)

STFC means Simple Triggered Force Curve. Will drive the Z output over the duration specified. If the specified input channel reaches the conditions set by the trigger parameters, the Z output will reverse direction. Use td_StopSTFC to halt the force curve.

Parameters

eventString	Can contain one event only. When this event has triggered, the feedback loop will begin. Events may be any of those listed in the *Event* section of Parameter Names. Note that you do not have to specify the group name, as Event will always be assumed. "0" is an effective substitution for "0%Event". "Never" — Never trigger. Not too useful. "Always" — Always trigger. Feedback starts immediately.
zStart	Where Z will begin its ramp from. To prevent clicking, the Z piezo should be ramped slowly to this position before calling this function. Units are volts.
zEnd	The farthest Z will be ramped to, assuming the trigger conditions are never hit. Units are volts.
time	Assuming that the trigger conditions are not met, the total time you would like the trip from zStart to zEnd to take. Note that the reverse ramp from zEnd to zStart is not included here.
inChannelString	String containing the name of the input channel to be monitor for trigger conditions. Any channel listed for use by td_xSetInWavePair may be specified.
triggerValue	When the monitored input channel hits this value, the z ramp will be reversed. Units are the units of the input channel.
greaterOrLesser	If this value is positive, "trigger when **greater than** triggerValue". If this value is negative, "trigger when **less than** triggerValue".
callback	This function will be called once the wave has completed. Currently does not work!

Function Result

0	Successful
1	Controller not connected
4	USB error
5	Bad channel string
10	Bad event name

See Also STFCTriggerCount%Controller

Example

```
// simple Z ramp from 150V to -5V that
//  triggers off its own value when crossing 80V
print td_WriteValue("Z%Output", 150)
print td_SetSTFC("0%Event", 150, -5, 0.5, "Z%Output", 80, -1, "")

// make a wave to capture this
make/n=120000 test
print td_xSetInWave(0, "0%Event", "Z%Output", test, "", 1)
```

```
// we'll look at the V-shape here
display test
print td_WriteString("0%Event", "Once")

// now let's check to see if it marked the trigger point
edit test
print td_ReadValue("STFCTriggerCount")
```

td_StopSTFC()

Halts the Simple Triggered Force Curve.

Function Result

0	Successful
1	Controller not connected
4	USB error

td_ShowVideo(source)

This function displays a window that contains a video image from a camera connected to the MFP. Please note that some video hardware may not support source selection capability using the *source* parameter. The source will have to be selected manual using the "Source" button on the resulting video dialog box.

Parameters

source — Selects the video source.

0	Off. Video dialog box is removed.
1	Composite video.
2	SVideo.
3	Tuner. A good source of snow, unless your lab is lucky enough to have cable. Plus there is no channel changer. If you'd like to be a couch potato, your best bet would be to use the stand alone application that comes with the video card.

Function Result

0	Successful
15	No video hardware found, or mode value not supported.

td_CaptureImage(waveName)

This function will capture the current video image into the specified wave name. An attempt is made to capture the image at a resolution of 640x480, but it is possible that your card may force a different resolution on the resulting image. A video window must be active, or this function will fail. See td_ShowVideo. If you press the "Capture" button on the video window, td_CaptureImage("capture") will be called.

Parameters

waveName — A string containing the name of a wave which will contain the captured image. This function will write over any data that might be in any existing wave of the same name without any warning.

Function Result

0	Successful
18	No video hardware found, or no video window currently displayed, or some other problem occurred within the video driver.

See Also td_ShowVideo

td_usTimer()

Returns the number of µs since the computer was last booted. You can find elapsed time in µs by calling this function before and after the Igor code you are trying to time and subtracting the two returned values.

Function Result

Current time in µs.

td_Date()

Returns a string containing the current date. Useful for generating date stamped filenames for saved data.

Function Result

Current date string in the form of "YYYY-MM-DD".

td_AdvancedFunction(advancedFunction)

Note! This is an advanced function. You should ignore it until you are more experienced with using this driver.

Executes an advanced function. This description intentionally left vague.

Parameters

advancedFunction	String containing the advanced function to execute. *Warning! This function becomes annoyed when advance functions are guessed at!*

Function Result

Returns a string containing the results of the advanced function. Do not display unless you have a sense of humor.

- ## MFP3D Force Review Programming

This help file is here to help you to Hack Asylum. We want you to tear apart our code and be able to build it up to do whatever you need it to. Before you start writing your code you should take a look at the User Demo, also see help topic [How To hack Asylum].

TOC

Force Review Glossary

Some conventions of the Asylum Force Display interface that are most often confused by new users.
Header: This is actually the wave note, which is NOT stored at the beginning of the ibw file. This wave note stores parameters for the collected data in a similar fashion as old style file headers.
Compressed: I have been calling some Force plot data as compressed or expended. The waves are saved in "Compressed" format, but once loaded, are expanded out. Compressed in this context does not have any influence on the size of the data, it is just that in compressed format all the data types of a given force plot are stored in a single 2D wave with the name BaseName+Suffix. The column labels are the DataType string. See MFP3D Master Force Panel glossary for an explanation of the waves name fragments. When expanded the 2D wave is broken up by the columns into BaseName+Suffix+DataType names.

- ## MFP3D Force DataFolders

The data folders used to store the Info for the force review.

Sub Toc

ForceCurves

[root:ForceCurves]
ARGetForceFolder("")
This folder stores the data from the harddrive. See MFP3D Force Data Format for more details on the format.

Parameters

[root:ForceCurves:Parameters]
ARGetForceFolder("Parameters")
General use Folder for the AR software.

Waves:

AxesListColor
(n+2,3) numerical wave, where n is the number of data types the software knows about.
Wave used to color the Axes List boxes. White for data types that are not there.
Black when all force plots have that data type. Degrees of grey when some force plots have that data type.

AxesListColor is Updated / Initialized with the function CalcDataTypeColor()

AxesListWave
(n,1) TextWave
Lists all the data types that the software knows about.
if is built from the list generated from

```
  String AllDataTypes = GetAvailableForceTypes(Inf)
  StrList2Wave2(AllDataTypes,AxesListWave,0)
```

AxesListBuddyX
(n,1,3) NumericalWave
there are GVH("NumOfForceAxes") copies of this wave,
AxesListBuddy0, AxesListWave1, ... AxesListBuddyMaxAxes
Where AxesListBuddy0 is the selected DataTypes for the bottoms Axes.

the first layer of this wave is the standard selection wave for ListBoxes
The second layer is there to be able to actually use the ColorWave (foreColors)
The third layer is to override the background color when using checkboxes.
See help for ListBox, DON'T try to read the info on the ColorWave, go down below and try their colored ListBox example to understand how stupid Igor is with the colorwave.

AxesListWave and AxesListBuddy are Initialized from the function UpdateDataTypeList()

SectionListWave
This Text wave lists the Sections as shown in the Section listbox on the Display panel

SectionListBuddy
This is the selection wave for the force plot sections. It is run in a very similar fashion as the AxesListBuddies.

SectionListColor
Again the color wave for the sections, very similar to AxesListColor

For the Section waves, see function CalcForceSectionColor()

ForceListWave
This is the text wave that lists all of the force plots in "BaseName+Suffix" format that can be found in root:ForceCurves:
Whenever this wave is updated, first a copy is stored in **OldForceListWave**, so that the changes can be determined and the data stored in the "data" folder can be cleared.
See also function UpdateForceList().

ForceListBuddy
This is the selection wave for the force plot selection ListBox. The Selection bit is bit 0, with no other bits being set. This way you can pull out the total number of force plots selected with sum.
Whenever this wave is updated, first a copy is stored in **OldForceListBuddy**, so that the changes can be determined and the data stored in the "data" folder can be cleared.

ForceDisplayList
This is the Text wave for the bottom Listbox on the ForceDisplayPanel. It lists the force plots that are displayed on the Force graph in "BaseName+Suffix" format

See function CalcForceListColor.

ForceDisplayBuddy
This is the selection wave for the bottom ListBox on the ForceDisplayPanel. It is a (n,1,3) dimension wave, where second layer defines the index to the ForceDisplayColor wave for the Listbox's Foreground colors (text), and the third layer defines the index to the ForceDisplayColor wave for the Background colors.

ForceDisplayColor
This wave is a (N+2,3) wave, where n is the number of force plots displayed on the force graph. The columns store the Red, Green and Blue colors to be used for the Listbox's colors. See also ForceDisplayBuddy and function CalcForceListColor.

ForceColorWave
This wave is a 101 by 3 matrix. It is used as a color table for the displayed waves, where the first column is the red color, the second column is the Green color, and the third column is the blue color. The row indexes for the colors are extrapolated so that if 2 force plots are overlaid, then it will use the colors from the first row and the color from the last row. If 3 force plots are overlaid, then it uses the first, middle, and last rows for the colors.

This wave is created with the function MakeMagicColor.

ForceGraphMacro
This text wave stores the recreation macro for the Force Display Graph. The recreation macro is built up from the style macro [see also function WinRecreation, where options is 3]. There are a couple tricks in this, mostly with the formatting of the execution strings passed from Igor. If all the axes are set to Log scale the command is ModifyGraph log = 1, however to keep things robust we need to expand that out to the command to modify each individual axes.

See also function RecordForceGraphMacro.

ForceHeaderWave
This is the Text wave that stores the headers for all the waves that have been displayed. The rows organize the parameters, while the columns separate the force plot names. The column Labels are "BaseName+Suffix". The first column label is "DefaultValue". This column established default values for the parameters for backward compliance. If a loaded force plot does not have a specified parameter, the value from column 0 is inserted into that column.
for example, say you are looking at a force plot that does not have the IsTriggered parameter in the header, when this force plot's header is extracted into the ForceHeaderWave, the defaultValue is used for IsTriggered (which happens to be 0)

If the header contains parameters that the ForceHeaderWave does not have, those parameters are inserted at the bottom of the ForceHeaderWave, after "//Start of Unknown Parms//"

See also function ExtractForceHeader.

ForceLoadDirs
This is the Text wave that stores the file path for the Force plots that have been loaded. The row labels correspond to the FPname (BaseName+Suffix). It is managed by the functions ARLoadFP, DeleteAllForceCurves, and DeleteSelectedForceCurves.

ParameterExport
This Text wave is the wave that is created when the user clicks on Export Parm Table. It contains the same data as is shown in the MFP3D Force Parms Panel. It is only created if the user has clicked on that button.

There are also IndexWave, PullDirWave, ZposCoef, AR_Coef, W_Sigma, and W_ParamConfidenceInterval waves. These waves hold transient data. They are not killed so as to speed up the code, but it is difficult to say what the values in these waves represent at any given state of the software.

Strings:

LastParm
This StringKey contains the last parameter changed. It is in the form:
FPName:Value;LastParm:Value;NewValue:Value;IsStr:Value;
It is maintained by AlterForceParameters, and used by the Set other Parms buttons / title.

LastMod
This StringKey contains the last modification to the last force plot. It is in the form:
FPName:Value;LastParm:Value;NewValue:Value;IsStr:Value;LastAction:Value;
Where LastAction describes the type of modification, i.e. "ParmChange"
This string is used by UpdateForceUndoButtons, to change the title of the undo and redo controls on the MFP3D Force Modify Panel.

Display

[root:ForceCurves:Display]
ARGetForceFolder("Display")
This folder ONLY stores the waves that are currently up on the Main Force Display Panel. They are broken up into the fragments of the waves stored in ForceCurves.
i.e. root:ForceCurves:Display:BaseName0002Defl_Ret stores the retrace data from root:ForceCurves:BaseName0002Defl

See also function ExtractForceData.

Data

[root:ForceCurves:Data]
ARGetForceFolder("Data")
As of yet, this folder hold nothing, but when the analyze and modify tabs get written they will store data here.

Undo

[root:ForceCurves:Undo]
ARGetForceFolder("Undo")
This is where all the Undo / Redo data is stored. Whenever a change is done, it looks in this folder. If the change is the same type as was last done (which is determined using the wave ForceUndoData), and if no redoes have been done, then the state of the force plot is stored in this folder before the mod is done. The Format of the wave is compressed (see glossary above). Undo and redo are basically the same operation from the standpoint of the software. They take the current state of the Force plot into a temp folder, copy the important data types from the Undo folder onto the data in ForceCurves, and then take those same data types from the temp dir and put them in the Undo folder. The main engine for this is SwapForceUndoData, which is run by ARForceUndo.

Waves:

ForceUndoData
This text wave contains the last modification to all of the force plots in memory. The column labels are:
LastAction, which is the modification type
LastParm, if the modification is ParmChange, then this contains the parameter that was changed.
OrgValue, this is the value of the parm before the first change of this type.
NewValue, this is the value of the parm after the last change.
IsStr, 1 if the parm changed is a string type variable (i.e. ForceNote)
DidUndo, 1 or 0, 1 if you have clicked on Undo, (so that the title will say Redo), 0 if you have not clicking on Undo Yet.

The row labels are the names of the ForcePlots in the form BaseName+Suffix

Restore

[root:ForceCurves:Restore]
ARGetForceFolder("Restore")
This folder contains all the restore data, data in its "Original" state. To try and cut down on memory usage, there are 2 steps when data goes into this folder. If the data was originally loaded from the hard drive, then the Wave ForceLoadDirs knows where that force plot came from, and will be able to load it when you ask for a restore. So the software waits for the user to ask for a restore, before going to the harddrive and loading the wave again (which it keeps in compressed format). But if the data was saved directly into memory, then it will not be able to load the file when asked, so when the first modification is done, first a copy of the wave is stored in this folder in compressed format. The main engine for this is SwapForceUndoData, which is run by ARForceRestore.

• MFP3D Force Examples

Some examples of how to craft your own force review....
This section assumes that you have a decent understanding of Igor Functions, their workspace, DataFolders, Local vs. Global parameters

If you feel a little weak with Igor, go through the MFP3D Getting Started help file
All of the example functions here are in the User Function, ForceReviewDemo.

Sub Toc

Get a parameter from a force plot

The most direct method is to address the wave note directly. However this is a bit tedious. For the 3D we have set up a single wave that stores the data in the wave note for all the waves that have been displayed.

But first the direct method
Say you have the wave BaseName0003Defl loaded into root:ForceCurves:

first you need to get the wave note into a string
```
   String NoteStr = Note(root:ForceCurves:BaseName0003Defl)
```
Now NoteStr is a string Key list (see StringByKey for more info on that). The Items are separated by carriage returns: num2char(13) or "\r"
The Keys are separated from the values by ":"
so from noteStr to get the StartDist
```
   Variable StartDist = NumberByKey("StartDist",NoteStr,":","\r")
```

A useful tool to see a wave note as you are programming is to open up a notebook
```
   NewNoteBook/K=1/F/N=ForceNoteBook
   NoteBook ForceNoteBook text=NoteStr
```
Will dump the string NoteStr into the notebook so that you can see what you are working on.

To get the parameters from the HeaderWave. You should first read Parameters -> ForceHeaderWave.
So this wave holds the text version of the wave note.

So if you want the StartDist from force plot BaseName0003Defl it is simply
```
   Variable StartDist = Str2num(root:ForceCurves:Parameters:ForceHeaderWave[%StartDist][%BaseName003
```
There are a couple of tricks you can do as well.
The wave will always be located in the data folder specified by ARGetForceFolder("Parameters")
So you can say
```
   String DataFolder = ARGetForceFolder("Parameters")
   Varable StarDist = str2num($DataFolder+"ForceHeaderWave"[%StartDist][%BaseName0003])
```
But it is good to first check to make sure that the force plot has been placed into the HeaderWave.
```
   Variable Index = FindDimLabel($DataFolder+"ForceHeaderWave","BaseName0003")
```
Then if Index is < 0 you know that it has not been done yet.

But an example function to get the Start Distance from a force plot wave:

```
   Function GetStartDistFromNote(DataWave)
```

```
      Wave DataWave

      String NoteStr = Note(DataWave)
      Variable StartDist = NumberByKey("StartDist",NoteStr,":","\r")
      return(StartDist)
End //GetStartDistFromNote
```

Then for the HeaderWave method

```
Function GetStartDistFromHeaderWave(DataWave)
      Wave DataWave

      String DataFolder = ARGetForceFolder("Parameters")
      Wave/T ForceHeaderWave = $DataFolder+"ForceHeaderWave"
      String FPName, BaseName, Suffix, DataType, SectionStr
      FPName = NameOfWave(DataWave)
      ExtractForceWaveName(FPName,BaseName,Suffix,DataType,SectionStr)
      Variable Index = FindDimlabel(ForceHeaderWave,1,BaseName+Suffix)
      if (Index < 0)
         ExtractForceHeader(ForceHeaderWave,DataWave)
         Index = FindDimlabel(ForceHeaderWave,1,BaseName+Suffix)
      endif
      Variable StartDist = str2num(ForceHeaderWave[%StartDist][Index])
      return(StartDist)
End //GetStartDistFromHeaderWave
```

As you can see, there is a fair amount more work to do with the header wave. There are a number of advantages though. The ExtractForceHeader function sets up some default values for things, where as the if StringByKey or NumberByKey do not find a parameter they return NaN.

I should also mention now about ExtractForceWaveName. If you have not already, you should right click on the function's name and select go to ExtractForceWaveName. This function uses Pass-By-Reference, so that the Strings BaseName, Suffix, DataType, and SectionStr, come out changed. This way we use ExtractForceWaveName to break a wave's name into it's components.

Increment the Display Controls to the next force plot

This example basically just drives the Force Plot List on the Force Display Panel.
The first step is to get the force plot list, and you should read about the waves in the Parameters section above. They are ForceListWave and ForceListBuddy.

```
SetDataFolder(ARGetForceFolder("Parameters"))
```
puts you where the ListWave and the ListBuddy are.
So you want to know where the first selected value is.
There is a function written to handle that for you.
```
Variable FirstSelected = Find1Index(ForceListWave,"==",1,0)
```
(see also Find1Index)
Now you want to check to make sure there is another value to select.
```
if (FirstSelected < DimSize(ForceListWave,0)-1)
      ForceListBuddy[FirstSelected] = 0   //Unselect
      ForceListBuddy[FirstSelected+1] = 1 //select
endif
```

Also in the If switch you want to call the Listbox's function, which is ARForceListFunc

```
ARForceListFunc("",0,0,0)
```

the parameters passed to the function are not used, except that the last parameter can not be 1 or 4

So all together now:

```
Function GoToNextForcePlot()

      String DataFolder = ARGetForceFolder("Parameters")
      Wave ForceListBuddy = $DataFolder+"ForceListWave"
      Variable FirstSelected = Find1Index(ForceListBuddy,"==",1,0)
      if (FirstSelected < DimSize(ForceListBuddy,0)-1)
         ForceListBuddy[FirstSelected] = 0        //UnSelect
         ForceListBuddy[FirstSelected+1] = 1      //select
         ARForceListFunc("",0,0,0)                //call the listbox function.
      endif

End //GoToNextForcePlot
```

But this is a limited case solution. If there are multiple Force plots selected, those will not be altered. But shifting all of the selected force plot by 1 is what the ">" button does. To call that function would be ShiftForceList(1)

Find a Displayed Wave pair

How to find a XY wave pair on the graph.
The easiest way is if you know already which left axes it will be on, and have some type of file name mask, such as you want to find the force.
So for this example we will find force on the bottom axes.

The graph's name can be obtained by calling ARGFGN()

```
String Graphstr = ARGFHN()
```

The bottom axes is named L0 (as in Lambda Zero)
To get a list of all the traces on the left axes you can use a function called ARTraceNameList

```
String TraceList = ARTraceNameList(GraphStr,"*Force*","L0","Bottom",";",1)
```

(see also help for ARTraceNameList)
This has already applied a mask to the traceNames so that only those with force in the name will be returned.
so now you want to go through the tracelist and get the X waves that go with those Y waves. That can be obtained from another function ARXTraceNameList.

```
String XWaveList = ARXTraceNameList(GraphStr,TraceList)
```

Then it is a simple matter of going through your lists to get access to your waves.

```
Function GetDisplayedForceWaves()
      String GraphStr = ARGFGN()
      DoWindow $GraphStr
      if (!V_Flag)  //if the window is not there
         return(0)  //then there is no point in going on.
      endif
      String TraceList = ARTraceNameList(GraphStr,"*Force*","L0","Bottom",";",1)

      String TraceName
      Variable A, nop = ItemsInList(TraceList,";")
      for (A = 0;A < nop;A += 1)
         TraceName = StringFromList(A,TraceList,";")
         Wave YData = TraceNameToWaveRef(GraphStr,TraceName)
         Wave XData = XWaveRefFromTrace(GraphStr,TraceName)
         //now you can do whatever you want with your X and Y data....
      endfor

End //GetDisplayedForceWaves
```

<u>Get a value from a wave pair</u>

So lets say you have your wave pair, you want to pull out the Adhesion force. So from the previous example, modify it so that you only get the Retract data (that is the TraceName mask is "*Force_Ret").

Then you will want to pass that data onto another function, something such as:

```
Function GetAdhesionFromRetract(YData,AdhesionWave)
     Wave YData, AdhesionWave

     //make room for the data we want to place in AdhesionWave
     Variable NumOfAdhesion = DimSize(AdhesionWave,0)
     InsertPoints/M=0 NumOfAdhesion,1,AdhesionWave
     //Maybe label it for future reference...
     SetDimLabel 0,NumOfAdhesion,$NameOfWave(YData),AdhesionWave


     //Lets offset the Ydata to the average of the last 5 points...
     //since this is retrace data, we know that away
     //from the surface is to the right of the wave.
     Variable NopOffset = 5
     Variable nop = DimSize(Ydata,0)


     Variable Offset = Sum(YData,pnt2x(YData,nop-NopOffset),pnt2x(YData,nop-1))/NopOffset


     //Now we assume that the lowest value of force is the adhesion....
     WaveStats/Q YData
     Variable Adhesion = (v_min-Offset)*-1      //reverse the sign...
     AdhesionWave[NumOfAdhesion] = Adhesion

End //GetAdhesionFromRetract
```

OK, So all we need to do to plug this into GetDisplayedForceWaves, is to first initalize the AdhesionWave.
and that final version would read something like this:

```
Function GetAdhesionFromForceWaves()
     String GraphStr = ARGFGN()
     DoWindow $GraphStr
     if (!V_Flag)  //if the window is not there
        return(0)  //then there is no point in going on.
     endif
     String TraceList = ARTraceNameList(GraphStr,"*Force_Ret","L0","Bottom",";",1)

     Wave/Z AdhesionData = root:AdhesionData
     if (WaveExists(AdhesionData) == 0)
        Make/N=0 Root:AdhesionData
        Wave AdhesionData = root:AdhesionData
     endif

     String TraceName
     Variable A, nop = ItemsInList(TraceList,";")
     for (A = 0;A < nop;A += 1)
        TraceName = StringFromList(A,TraceList,";")
        Wave YData = TraceNameToWaveRef(GraphStr,TraceName)
        GetAdhesionFromRetract(YData,AdhesionData)
     endfor

End //GetAdhesionFromForceWaves
```

Set up a loop to go through all the waves in the folder

So this example will outline how to go through all the waves in the root:ForceCurves folder, drive the Display controls to extract the wave fragments for you, and then Extract the adhesion data from the force retrace data.

First off we get the Force list wave
String DataFolder = ARGetForceFolder("Parameters")
Wave ForceListBuddy = $DataFolder+"ForceListBuddy"
then we need to clear the selections
ForceListBuddy = 0
and select the fist wave listed.
ForceListBuddy[0] = 1
then call the listbox function.

```
  ARForceListFunc("",0,0,0)
```

Then after we have initialized the adhesionwave we are ready to set up the loop.
Inside the loop we are just going to call functions from previous examples, such as
GoToNextForcePlot()
and
GetAdhesionFromForceWaves()

So the Final function would read:

```
  Function GetAllForceAdhesion()

       //Init our adhesion wave.
       Make/O/N=0 Root:AdhesionData
       //make sure this is the same name as in GetAhdesionFromForceWaves.
       SetScale d,0,0,"N",AdhesionData

       String DataFolder = ARGetForceFolder("Parameters")
       Wave ForceListBuddy = $DataFolder+"ForceListBuddy"

       ForceListBuddy = 0
       ForceListBuddy[0] = 1
       Variable A, nop = DimSize(ForceListBuddy,0)

       for (A = 0;A < nop;A += 1)
          GetAdhesionFromForceWaves()
          GoToNextForcePlot()
       endfor

       //OK, lets add in a little spice,
       //we would like to see our data...
       Wave AdhesionData = root:AdhesionData
       ARHistogram(AdhesionData,"AdhesionData",0)     //make a histogram.


  End //GetAllForceAdhesion
```

The only thing you have to make sure of before calling GetAllForceAdhesion, is that the force data type is selected in the first axes. This will probably crash if one of the force plots listed does not have deflection saved, well then again, it might be fine...

Calculate dependant data types

For this example we will calculate force from the deflection data. Only the deflection data is stored in saved wave, so if we want to deal with force, we need to calculate it. For more details of dependant data types see "Add a new calculated Data Type".

The easiest way to do this is to use the function

CalcForceDataType(FPName)
FPName is the BaseName+Suffix+DataType of the full wave you want.
So if you give it "BaseName0009AmpV" it will look for root:ForceCurves:BaseName0009Amp, and then create the wave root:ForceCurves:BaseName0009AmpV. It also returns this string so that you can get your wave ref in one line.
Wave YData = $CalcForceDataType("BaseName0009AmpV")
if (WaveExists(YData) == 0) //problem, could not get the data.
else
//all is good lets do something with it.
endif

So for an example, lets go through all the deflection force plots listed in the root:ForceCurves folder and make sure there is the force wave as well.

```
Function MakeAllForceWaves()
      String DataFolder = ARGetForceFolder("")
      String SavedDataFolder = GetDataFolder(1)
      SetDataFolder(DataFolder)
      //get a list of force waves in the root:ForceCurves folder
      String DataList = WaveList("*Defl*",";","")
      SetDataFolder(SavedDataFolder)

      String FPName, BaseName, Suffix, DataType, SectionStr

      variable A, nop = ItemsInList(DataList,";")
      for (A = 0;A < nop;A += 1)
         FPName = StringFromList(A,DataList,";")
         ExtractForceWaveName(FPName,BaseName,Suffix,DataType,SectionStr)
         Wave/Z Force = $CalcForceDataType(BaseName+Suffix+"Force")
      endfor

End //MakeAllForceWaves
```

Extract the wave subsets without Displaying them

This example is a bit more advanced that the previous cases. Before we were using the functions setup to display the force plots, but since they are running the graphics, they can be a bit slow. At some point you are probably going to want to get the wave fragments out yourself. The easiest way to do that is to call the function ExtractForceSection(SrcData,DestData)
Where SrcData is the full wave in root:ForceCurves, and the DestData wave is a wave you want the fragment to be stored in. It figures out what fragment you are looking for from the name of DestData.

So lets say you have a wave reference of Ydata = root:ForceCurves:BaseName0003Force, and you want the Retrace data placed in a wave in another folder.

So first you Initialize your folder
NewDataFolder/O/S root:MyForceData
Make/O/N=0 BaseName0003Force_Ret

the name structure of your wave is important, in that it needs to have the same basic structure as the wave fragments. The BaseName and Suffix tell ExtractForceSection where to get the header Parameters from the ForceHeaderWave. The _Ret part of the wave's name will be used to tell ExtractForceSection to get the retrace data.

So then you can call
ExtractForceSection(YData,BaseName0003Force_Ret)

Now BaseName0003Force_Ret contains the Retrace force data from YData.
It is also important to understand that if ExtractForceSection was not able to get you your data, it will Kill your destination wave.

So for our example function, lets set up a loop to go through all the Force plot in the root:ForceCurves: folder that have Force in their name and Get the Retrace data out of them. And adding onto the last example, we will take the deflection waves and create the force waves.

```
Function GetRetraceDataFromForceWaves()
     String DataFolder = ARGetForceFolder("")
     String SavedDataFolder = GetDataFolder(1)
     SetDataFolder(DataFolder)
     //get a list of force waves in the root:ForceCurves folder
     String DataList = WaveList("*Defl*",";","")

     //ok lets set up our data folder
     //here we will set it up off of the root:ForceCurves folder in
     //UserForceData
     String DestFolder = DataFolder+"UserForceData"

     NewDataFolder/O/S $DestFolder            //make the folder
     DestFolder += "://add in the Colon to make it easy to use later.
     Make/O/N=0 DestData0000Force_Ret        //set up our destination wave
     Wave DestWave = DestData0000Force_Ret       //get a local ref to it.

     Make/O/N=0 AdhesionWave                           //set up our adhesionwave
     Wave AdhesionWave = AdhesionWave
     SetScale d,0,0,"N",AdhesionWave

     SetDataFolder(SavedDataFolder)             //go back to where we started.
     String FPName, BaseName, Suffix, DataType, SectionStr

     Variable A, nop = ItemsInList(DataList,";")

     for (A = 0;A < nop;A += 1)
        //the next 3 lines are from the last example.
        FPName = StringFromList(A,DataList,";")
        ExtractForceWaveName(FPName,BaseName,Suffix,DataType,SectionStr)
        Wave/Z YData = $CalcForceDataType(BaseName+Suffix+"Force")

        if (WaveExists(YData) == 1)
           ReName DestWave,$BaseName+Suffix+"Force_Ret"
           ExtractForceSection(YData,DestWave)
           //NOW, we pull out the Retract into DestWave

           if (WaveExists(DestWave) == 0)
             //sorry, could not do it (probably Dwell section)
             Make/N=0 $DestFolder+"DestData0000Force_Ret"
             Wave DestWave = $DestFolder+"DestData0000Force_Ret"
           else
           //send our data somewhere else...
             GetAdhesionFromRetract(DestWave,AdhesionWave)
           endif
        endif
     endfor

     ARHistogram(AdhesionWave,"AdhesionData2",0)    //make a histogram.

End //GetRetraceDataFromForceWaves
```

Add a new parameter to the Wave note

This example is a bit different than the previous examples. Here we outline how to add a new parameter to the wave note that will be saved to the harddrive, as well as when the force plots are saved to Igor's memory. There is also another step as well with the offline force software and that is to setup the ForceHeaderWave to have a default value for this New parameter, which is important when dealing with backward compliance. The note for the force plots is built up in the function FinishForceFunc in Puller.ipf. So you will want that function overridden in your user ipf. an example of adding the parameter CurrentGain from the Global Variable Root:CurrentGain is provided in the UserForceReviewDemo.

Now to set up ForceHeaderWave.
Open up a fresh MFP3D template, without opening the force review panel, go to the ForceHeaderWave.

```
  SetDataFolder(ARGetForceFolder("Parameters"))
  Edit/K=1 ForceHeaderWave.LD
```

Insert a point somewhere for your new parameter.

```
  InsertPoint/M=0 1,1,ForceHeaderWave
```

Set your DimLabel for the new point, in the next example we need CurrentGain, so let's add that.

```
  SetDimLabel 0,1,CurrentGain,ForceHeaderWave
```

and finally give it a default value. "NaN" works good in that if the force plot does not have the parameter the calculated wave (from the next example at least) will be full of NaNs and not show up even though it will still be plotted.

```
  ForceHeaderWave[1][0] = "NaN"
```

Now save this wave to your user folder you created to set up load your code. You can load it in a similar fashion as in the ForceReviewDemo, in the init function.

Add a new calculated Data Type

This example is quite a bit different than the previous examples. This is here to outline how to add a new data type to the list of data types, and how to get the software to use that data type. For example you may be collecting a Current signal from UserIn0, and you want the software to be able to convert your UserIn0 voltage signal to the amp signal. You need to teach the software how to do that.

First you need to understand the 4 classes of data types we have broken the data into. There is the base data types, which are directly saved from the controller. a list of these data types are returned from the function

```
  GetAllForceTypes()
```

which is the standard channels such as deflection, amplitude, etc.
There are also Intrinsic data types, these data types are ones we can always calculate. Right now the only examples of intrinsic are drive and time. These data types are hard wired into the software and are a bit more difficult to add.
Then there are first order calculated data types. These are data types which are directly scaled from a single parent base data type. Good examples of this are force and AmpV. Even though technically speaking, AmpV is the parent data type, since that is what is brought off the controller, life gets a lot easier if we call the Amp (nm) channel the parent. A list of first order data channels is returned from the function:

```
  GetCalcAbleForceTypes()
```

the string returned is a string key list in the form
ParentDataType:ChildDataType0;ChildDataType1,ParentDataType:ChildDataType0,
Then there are the Second order data types, these require 2 base data types to calculate. Currently there is only Separation (which require deflection and Raw), and Dissipation (which requires Phase and Amplitude).
a list of these data types is returned from the function :

```
  GetDoubleCalcAbleForceTypes()
```

This is in the form:

ParentDaaType1_ParentDataType2:ChildDataType;ParentDaaType1_ParentDataType2:ChildDataTyp e;

So here we will outline how to add a new First order data type.

The first step to adding a new data type is to make your user function as described in the UserDemo. See also "How to Hack Asylum".

You are going to have to override the function GetCalcAbleForceTypes()

So if you wanted to add the Curr data type which is dependant on UserIn0

in your Userfunction you would have the function:

```
  Override Function/S GetCalcAbleForceTypes()

       String output = "Defl:Force;DeflV,Amp:AmpV,Raw:LVDT;RawV,UsI0:Curr,"
       return(output)
  End //GetCalcAbleForceTypes
```

(you need to look in GetAllForceTypes to see what the channel abbreviation is).

Now the tricky part, you need to teach CalcForceDataType what to do with this data type.

So first copy CalcForceDataType into your user function and put and override before the function keyword.
So inside calcForceDataType there is a call to ARGetDataCalcType. It will be passed the data type being requested. The case that we need to teach it is if Curr is begin asked for. Because we have GetCalcAbleForceTypes listing Curr, ARGetDataCalcType will return 2 (First order data type) and the string Source Type will be UsI0 (Pass-By-Reference).

Then CalcForceDataType will try to get the parent data type and call it data. If it can not get the parent it returns an empty string (which tells the calling function that it can not be done). If it can find the parent, then it duplicates the parent into the destination wave (called output). Below that there is a if switch matching the DataType string, (which in our case will be Curr).
So in your user version will need to add the case

```
  elseif (StringMatch(DataType,"Curr*") == 1)
       Variable Gain = str2num(HeaderWave[%CurrentGain][Index])
       output = (Gain) * output
       SetScale d,0,0,"A",output
```

and then you are done with this function. That is assuming that you have already set up the parameter CurrentGain in the HeaderWave as outlined in "Add a new parameter to the Wave note".

Now the only other thing you should be aware of is that the waves AxesListWave, AxesListBuddyX and AxesListColor (see Parameters -> waves for more details on these) are not updated when you load your user function. If the ForceDisplayPanel has been made they will have been built with the lists in the original functions. So it is a good idea to call the functions

```
  UpdateDataTypeList()
```

and

```
  CalcDataTypeColor()
```

in that order in your init function.

You can see the example functions in the user demo ForceReviewDemo

How to Edit a New parameter from the Parms Panel

This example outlines what you need to do to integrate the ability to edit a new parameter. You will need to override a number or AR functions, so if you have not gone through the topic "How to Hack Asylum", you should do so first. OK, so the parameters that can be changed are hardwired in 3 functions.
The first one enables the controls on the Force Parms Panel so that you can edit the parameters.
The second one does the work you need it to do
and the third one tells the undo which waves and parameters it needs to revert.

The first one is GetEditableForceParms(), which just returns a string list of ParmNames that can be edited. This is used by ARCanEditForceParm, which sets up which setvars on the force parm panel can be used and which are disabled. So just add in the parm you want in the same format as it is in the ForceHeaderWave.

The second function you need to change is ChangeForceParm. Don't worry about the input parameters, you just need to add your parameter in the strswitch. Below is the case statement for Invols:

```
strswitch (ParmStr)
   case "Invols":
          //OK, I know I have to redo
          //Defl, Force, and Sep
          Factor = varNum/str2num(OldValue)

          Wave/Z Defl = $SrcFolder+FPname+"Defl"
          if (WaveExists(Defl) == 1)
                 FastOp Defl = (Factor) * Defl
          endif
          Wave/Z Force = $SrcFolder+FPname+"Force"
          if (WaveExists(Force) == 1)
                 FastOp Force = (Factor) * Force
          endif
          Wave/Z Sep = $SrcFolder+FPname+"Sep"
          if (WaveExists(Sep) == 1)
                 KillWaves/Z Sep
                 CalcForceDataType(FPname+"Sep")
          endif
          ReExtractForceList(FPName,"Defl;Force;Sep;")
          break
```

srcFolder is root:ForceCurves, the source data. So it addresses the source data, and then modifies the waves according to new parameter (VarNum). str2num(OldValue) is the last used value (obtained from the ForceHeaderWave, NOT ForceUndoData)
Notice that the separation is a bit different than the other. Since this is a second order data type (see "Add a new calculated Data Type") it needs to be re-calculated based on its parents (Defl and LVDT). This is done with the function CalcForceDataType.
Once you have updated the source data in root:ForceCurves it is a matter of getting the data down into the Display folder for the Graph to show the new waves. This is done with the function ReExtractForceList, where you need to list all of the data types that you have modified.

The third function you need to modify is GetParmChangeDataTypeList. This function reflects the same strSwitch as ChangeForceParm.

```
strswitch (ParmStr)
   case "Invols":
```

```
Waveoutput += "Defl;Force;Sep;"
ParmOutput += ParmStr+";"
break
```

where this function needs to return the list of waves that are changed by a given parameter and the parameters changed (since changing one parameter may need to update additional parameters). This function governs which waves are changed when Undo / Redo does it's work. So when you change your parameter, that parameter name will be stored in ForceUndoData. Then if you need to click on the undo button, the undo needs to know which waves to swap. Just add them to the string Waveoutput. The full output from GetParmChangeDataTypeList, is a stringkey, of form:
Waves:DataType0;DataType1;\rParms:ParmType0;ParmType1;
So just use the += to add your waves and parms to the Waveoutput and Parmoutput so that GetParmChangeDataTypeList can put them together correctly at the end of the function.

Once you have these 3 functions fixed up the way you want them, then you should be all set to edit your new parameter. And I would just like to add that it is quite important to make clear comments around the places that you change our functions in your user function, so that when you update the AR code, you can go and get the new version of those 3 functions (which may or may not have changed), and easily insert your changes into the new version of your overridden functions.

• MFP3D Force Data Format

Maybe some time I will talk some about the Force data format. How the Waves are plug together in the IBW file. The various header parameters. Which ones are really important. That sort of thing....

Sub Toc

A topic

This will be a future topic....

- **MFP3D Generalized Waveform Input and Output**

This help file is intended to allow you to set up your own waveforms for output and input for the realtime force curves. This is one way of running the MFP-3D directly.

TOC

RealTime Force Glossary

Some of the basic concepts of the Asylum Controller interface.

LowLevel vs. HighLevel: Software can be described by two basic classes, high level (Igor), and low level (the “td_*” commands sent to the controller). Low level languages are very restrictive to program in. It requires a detailed understanding of the MFP-3D before a programmer will be able to make much progress. While low level programming has numerous restrictions on the sytax of the code, it often pays back in speed. On the other hand, high level code, usually provides a much friendlier interface, allowing much faster software development.. With the MFP3D, the commands sent to the controller are low level commands, where the code is stored in the MFP3D.XOP. Most of the controls the average user interacts with are based on the high level Igor code..

- **Overview of LowLevel Commands**

More Detail is given in MFP3D help.
[MFP3D.xop - Interface for the Asylum Research MFP-3D]

Sub Toc

Error Codes

Most of the td_* commands output an error code. For example, the function td_xSetInWave will return any zero if things went smoothly, and a non-zero value if there were any errors, however functions such as td_readValue will output the value being read. It is good practice to pay attention to these errors (even the non-fatal ones!) as you go along. An example of this is shown in the snippet of code below.

```
Function TestInNullWaveError()

Variable Error = 0
error += td_xSetInWave(0,"1%Event","Fast%Input",$"","",1)
if (error)
     print "Error code: "+num2str(Error)+"In TestInNullWaveError"
endif

End //TestInNullWaveError
```

This code will generate an error code because the wave passed to it is a Null (non-existing) wave.

Extra help for td_xSetOutWave

First read the help for td_xSetOutWave.
Now lets try to translate some of that info.
WhichBank:
you have 3 banks, 0 to 2
Bank 0:
Can take 2 waves, is 16 bit data, can have a decimation of 1
Bank 1:
Can take 2 waves, is 16 bit data, must have at least decimation of 2
Bank 2:
Can only take 1 Wave, is 32 bit data, must have at least decimation of 2.

Event string
this is a string to tells you what "event" will trigger the waves to actually go.
there are 6 user events (we almost always use user 0).
for we use "0%Event" as our event string.
You may want to use
"1%Event" for all your evenStrings
This means that whenever
td_writeString("1%Event","Once") is called, then the user 1 event is triggered and all the waves set up on the banks to run during that event go.

ChannelString:
This one is pretty tricky to learn, but describes the channel string you want to have the waves running on.
for outputs you have all the Digital to analog converters, which are: the X,Y, Z Piezos, and the A, B, and C user DACs. So the strings for this will be "X%output", to "C%output".

Wave:
This is the output wave, the wave that the will be sent to the DACs to drive the various outputs of the MFP3D

Interpolation:
Same concept as decimation. If you give the function 10 points, and interpolation of 100, the DAC will be driven with 1000 values. There are a couple tricks to keep in mind. Positive values for the interpolation use a cubic spline, negative values use a linear interpolation. The other trick is if the event is continuous or not. Rule of thumb is that if the wave is continous, make it one point short of the full wave. If it is a one shot deal, give the wave the full waveform you want to feed it. When the low level code interpolates a continous wave, it will match up the missing point with the first point for you.

- **Examples**

Examples on how to drive the Z piezo yourself.

Sub Toc

Clearing the Controller

It is good practice to always clear the controller before doing any work on it. You don't know exactly what has been started before. There are 2 lowlevel commands to call:

```
  td_StopThermal()
```

and

```
  td_stop()
```

Stop thermal stops the thermal data collecting, which is a very different mode than the standard operations. You should always call td_StopThermal before calling td_Stop, because using td_Stop to clear thermal data collection can leave the controller in an error state.

Make your waves

You typically want to have a drive wave and a wave to collect data on. So lets make a wave called data, and another called Drive. *The most important thing to remember is that the number of points in the waves has to be a multiple of 32, and they need to be the same length.* In this example, we'll make our waves 1024 points long.

```
  Make/O/N=1024 DataWave, DriveWave
```

Don't worry about setting the scale of the waves, the td commands will do that for you.

The relationship between the length of the waves and the time it will take to complete the scan is determined with Decimation. Decimation is something we will cover in the next example.

Now for fun, lets make our Drive wave a sin wave.

```
  DriveWave = sin(pi*p/50)
  Display/K=1 DriveWave
```

We will display DataWave as well:

```
  Display/K=1 DataWave
```

Setting your Drive wave

Once you have made your waves you want to set them. These use the function td_xSetOutWave.
You should defiantly read the help for this function before going on.
And make sure you have read the topic "Extra help for td_xSetOutWave" from above.
OK, now that you understand the arguments for td_xSetOutWave, you should be able to write your own line by yourself.

But we will provide you with an example anyway...

```
  Variable ScanRate = 2                    //Hz
  print td_xSetOutWave(0,"1%Event","Z%output",DriveWave,round(100000/ScanRate/DimSize(DriveWave,0))
```

If you were watching your DriveWave on the graph you built, our would have noticed that this command set the X scaling of your DriveWave. Not it is clear that this wave will take 1/2 Second to run.
The Decimation argument is Sampling Rate over (number of points to collect Times ScanRate), and should always be a whole number.
So we want to drive the Z piezo with our drive wave, for 1/2 Second, one bank zero which will be triggered off of user event 1.

Setting your Data Wave

This uses the function td_xSetInWave, which is very similar to td_xSetOutWave. The only real difference from the point of view of using the function is that you have the extra parameter of the callback. This is a string that will be executed when the scan is done. You can have it call a user function, to say scale your Deflection signal which is in Volts to nm based on the Invols. Now you need to be aware that there is a "bug" with Igor, it will not update the waves collected after a scan is done. For a reason which can take a while to explain, Igor does not know that the wave has changed, so it does not update the graphs. So it is good practice to "change" the wave so that any graphs will update. The easiest way to do that is to add zero to an element of the wave:
DataWave[0] += 0

So our example td_xSetInWave will be:

```
  print td_xSetInWave(0,"1%Event","A%Input",DataWave,"DataWave[0]+=0",round(100000/ScanRate/DimSize
```

This will collect data from InA. But to see what that is connected to you need to look at the crosspoint panel. Your DataWave should now have a X scaling the same as the DriveWave and a Data scale in units of Volts.

Triggering your event

OK, now that you have your waves set up you want to trigger them.
This is done with the function td_WriteString.

```
  print td_writeString("1%Event","Once")
```

will trigger all the waves set up on the user 1 event to go once.

• How to hack Asylum

This topic will guide you on how to set up your own set of code.

Sub Toc

The dilemma

So you want to change our functions to do what you want them to do. That is fine, we are behind you all the way. The problem is, if you go in directly to our software and change things to work the way you want them to do, you won't be able to get updated software from us. Since we are going to overwrite all the changes you have done. So we have set up a mechanism for you to load up external code that can override our software. These are the User functions, loaded from the Programming -> Load User Func menu. When you contain all of your software in these external ipf's then when you run a new installer to get software updates from us, your code will remain the same. There may be some problems if we change any of the functions that you use, but that is a small problem next to trying to figure out all the places we have changed our software to update how you want to run things.

Making your user function

If you have not done so yet, you should open up the User demo from the Programming -> Load User Func Menu. Then select User Demo. Open up the UserDemo.ipf and DemoUserProc.ipf files. Basically when you load a user function, it first lists all the ipf's in the AslumResearch:Code3D:UserProc folder. You select the one you want and it loads that ipf. It is that ipf's job to do 2 things. Include additional ipf's (which by convention are stored in a folder off AslumResearch:Code3D:UserProc named the same as the ipf minus the name User, but that is not required. The second job required by the ipf is to have a function called "Init"+NameOfIPF. so for example if your ipf is names UserDemo, then it should have a function in it called InitUserDemo. The user function loader will call this function when it is loaded, so that you can initialize things.

So for a guided example we will make a UserFunc called MyNameForce.
Make a directory called
AslumResearch:Code3D:UserProc:MyNameForce
Open up UserDemo.ipf into Igor, and save that as
AslumResearch:Code3D:UserProc:MyNameForce.ipf
Open up DemoUserProc into Igor and save that as
AslumResearch:Code3D:UserProc:MyNameForce:NewForceFunctions.ipf

Change the include statement in MyNameForce.ipf to reflect the second ipf
#include ":AsylumResearch:Code3D:UserProcs:MyNameForce:NewForceFunctions"

Change the RemoveUserFunc call to RemoveUserFunc("MyNameForce")
Change the InitUserDemo function name to InitMyNameForce
then delete the Override function DoScanFunc
and delete the function from NewForceFunctions.ipf

Save both your Ipf's
Now you have a clean slate to start writing your new functions.

Initializing your user function

Well I guess most of this topic was really covered in the previous topic. If you want to do a number of things it is probably more convenient to just call a function located in your second user func from your InitMyNameForce function.

Overriding the AR functions

So you want have one of our functions do something different, maybe you want to call one of your functions at some point. The best way to do this is to but an override function in your MyNameForce.ipf. In the userDemo set there is an example of overriding the DoScanFunc, which is one of the main functions in the AR software to do any scanning. The way that is set up now whenever you click on one of the buttons that call DoScanFunc, it will call the version in MyNameForce.ipf, and in that example, it was then re-directed to the function DemoUserDoScanFunc, which lives in the second ipf, DemoUserProc. The easiest way to get going is to copy and paste functions from AR ipf's into your ipf's putting the override keyword before the function keyword. Then you can alter your version the way you want to. And example of this can be found in [MFP3D Force Review Programming]. The problem with this method is that it is tough to make sure things will remain working after an update. If we add new features to a function that you have overridden then when you load your user function that new feature will be broken (since your version did not know about it). There is no perfect solution, getting updates will always be a problem to your code. But by keeping the functions you override in the first ipf, you have a list of functions to watch whenever you get an update, make clear comments around those changes, and even if it is only a few lines, just call another function in your second ipf. That way you can easily grab a new version of the function after an update and alter it again to handle your software.