Stepping Through Your Code

Single-stepping through code is useful when you are not sure what path it is taking or how variables wound up containing their values.

Begin by enabling the debugger and setting a breakpoint on the line of code you are interested in, or begin when the debugger automatically opens because of an error. Use the buttons at the top of the debugger window to step through your code:

The Stop Button

The Stop button ceases execution of the running function or macro before it completes. This is equivalent to clicking Igor's Abort button while the procedure is running. If you have enabled Debug on Abort, the Stop button still causes execution to cease.

Keyboard shortcuts: Command-period (*Macintosh*), Ctrl+Break (*Windows*)

Pressing Command-period on a Macintosh while the debugger window is showing is equivalent to clicking the *Go* button, not the Stop button.

The Step Button

The Step button executes the next line. If the line contains a call to one or more subroutines, execution continues until the subroutines return or until an error or breakpoint is encountered. Upon return, execution halts until you click a different button.

Keyboard shortcuts: Enter, keypad Enter, or Return

The Step Into Button

The Step Into button executes the next line. If that line contains a call to one or more subroutines, execution halts when the first subroutine is entered. The Stack list of currently executing routines shows the most recently entered routine as the last item in the list.

Keyboard shortcuts: +, =, or keyPad +

The Step Out Button

The Step Out button executes until the current subroutine is exited, or an error or breakpoint is encountered.

Keyboard shortcuts: -, _ (underscore) or keypad -

The Go Button

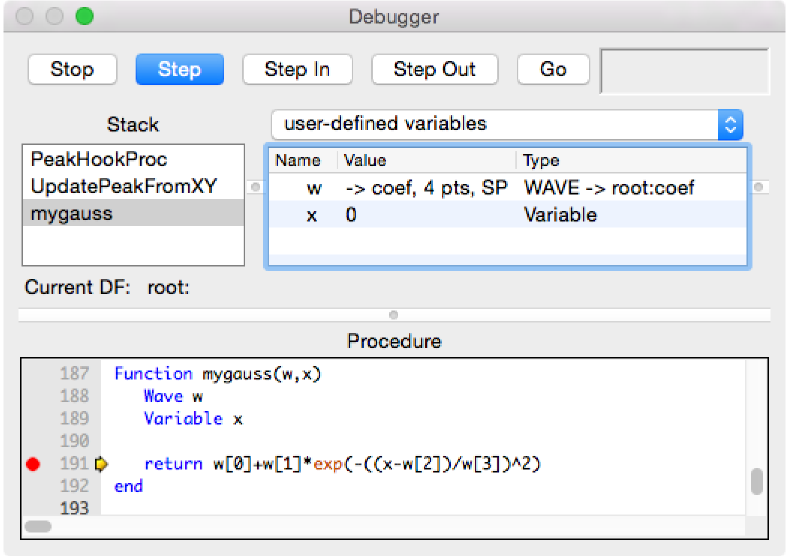
The Go button resumes program execution. The debugger window remains open until execution completes or an error or breakpoint is encountered.

If you press the Option (*Macintosh*) or Alt (*Windows*) key while clicking the Go button, the debugger window is closed until execution completes or an error or breakpoint is encountered.

Keyboard shortcuts: Esc

The Stack and Variables Lists

The Stack List shows the routine that is currently executing and the chain of routines that called it. The top item in the list is the routine that began execution and the bottom item is the routine which is currently executing.



PeakHookProc
UpdatePeakFromXY
mygauss

Name	Value	Type
w	-> coef, 4 pts, SP	WAVE -> root:coef
x	0	Variable

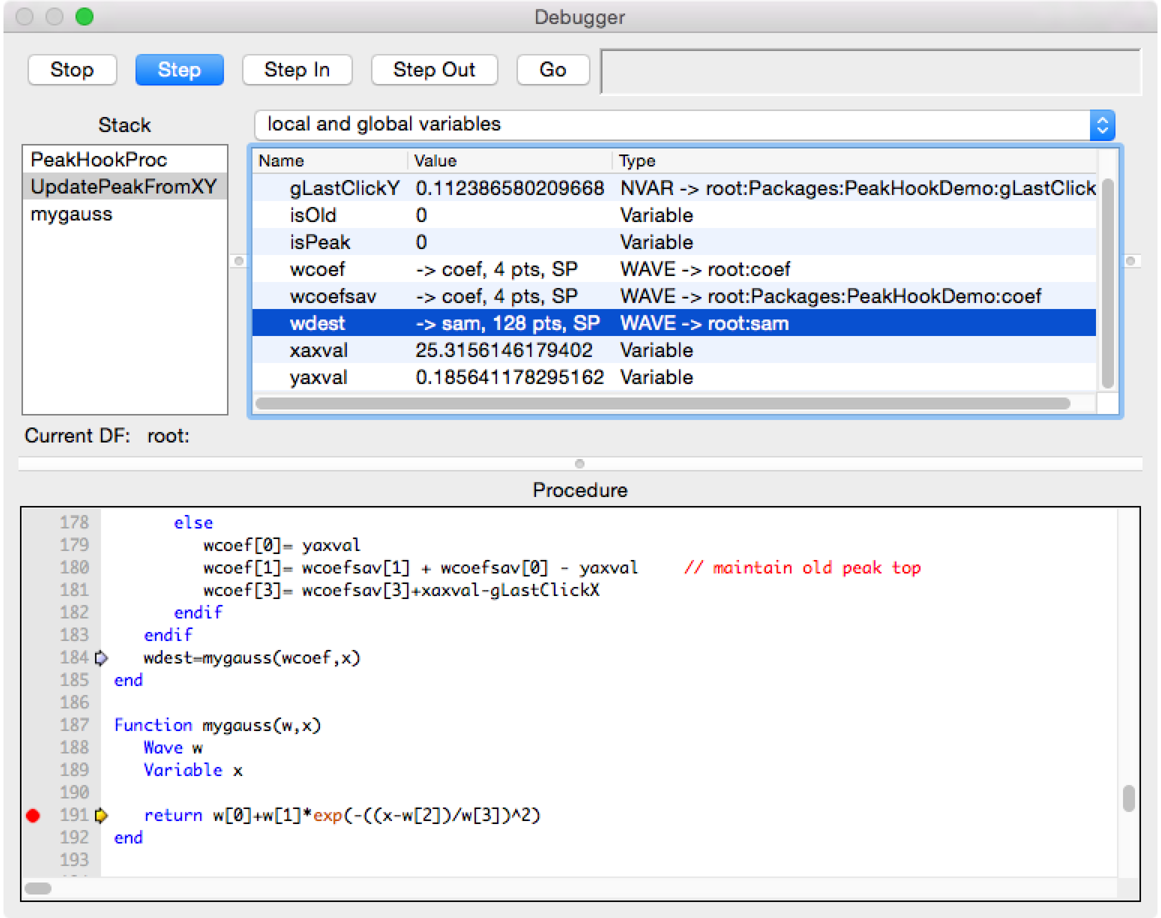
Current DF: root:

```
Function mygauss(w,x)
    Wave w
    Variable x

    return w[0]+w[1]*exp(-((x-w[2])/w[3])^2)
end

```

In this example, the routine that started execution is PeakHookProc, which most recently called UpdatePeakFromXY, which then called the currently executing mygauss user function.

The Variables List, to the right of the Stack List, shows that the function parameters w and x have the values coef (a wave) and 0 (a number). The pop-up menu controls which variables are displayed in the list; the example shows only user-defined local variables.

You can examine the variables associated with any routine in the Stack List by simply selecting the routine:



PeakHookProc
UpdatePeakFromXY
mygauss

Name	Value	Type
gLastClickY	0.112386580209668	NVAR -> root:Packages:PeakHookDemo:gLastClick
isOld	0	Variable
isPeak	0	Variable
wcoef	-> coef, 4 pts, SP	WAVE -> root:coef
wcoefsav	-> coef, 4 pts, SP	WAVE -> root:Packages:PeakHookDemo:coef
wdest	-> sam, 128 pts, SP	WAVE -> root:sam
xaxval	25.3156146179402	Variable
yaxval	0.185641178295162	Variable

Current DF: root:

```
        else
            wcoef[0]= yaxval
            wcoef[1]= wcoefsav[1] + wcoefsav[0] - yaxval      // maintain old peak top
            wcoef[3]= wcoefsav[3]+xaxval-gLastClickX
        endif
    endif
    wdest=mygauss(wcoef,x)
end

Function mygauss(w,x)
    Wave w
    Variable x

    return w[0]+w[1]*exp(-((x-w[2])/w[3])^2)
end

```