

dependence on preferences can be seen as a feature or as a problem, depending on what you are trying to achieve. We normally prefer to keep procedures independent of preferences.

Using a style macro is a more robust technique. To do this, you would first create a prototype graph and create a style macro for the graph (see **Graph Style Macros** on page II-350). Then, you would put a call to the style macro at the end of the LoadAndGraph macro. The style macro would apply its styles to the new graph.

To make your code self-contained, you can set the graph formatting directly in the code. You should do this in a subroutine to avoid cluttering the LoadAndGraph function.

The last approach is to overwrite data in an existing graph rather than creating a new one. The simplest way to do this is to always use the same names for your waves. For example, imagine that you load a file with three waves and you name them wave0, wave1, wave2. Now you make a graph of the waves and set everything in the graph to your taste. You now load another file, use the same names and use LoadWave's overwrite option. The data from the new file will replace the data in your existing waves and Igor will automatically update the existing graph. Using this approach, the function simplifies to this:

```
Function LoadAndGraph(fileName, pathName)
    String fileName      // Name of file to load or "" to get dialog
    String pathName     // Name of path or "" to get dialog

    // load the waves, overwriting existing waves
    LoadWave/J/D/O/N/P=$pathName fileName
    if (V_flag==0)      // No waves loaded. Perhaps user canceled.
        return -1
    endif

    Textbox/C/N=TBFfileName/A=LT "Waves loaded from " + S_fileName
    return 0            // Signifies success.
End
```

There is one subtle change here. We have used the /N option with the LoadWave operation, which auto-names the incoming waves using the names wave0, wave1, and wave2.

You can see that this approach is about as simple as it can get. The downside is that you wind up with uninformative names like wave0. You can use the LoadWave /B flag to provide better names.

If you are loading data from Igor binary wave files or from packed Igor experiments, you can use the LoadData operation instead of LoadWave. This is a powerful operation, especially if you have multiple sets of identically structured data, as would be produced by multiple runs of an experiment. See **The LoadData Operation** on page II-156 above.

Loading and Graphing XY Data

In the preceding example, we treated all of the columns in the file the same: as waveforms. If you have XY data then things change a bit. We need to make some more assumptions about the columns in the file. For example, we might have a collection of files with four columns which represent two XY pairs. The first two columns are the first XY pair and the second two columns are the second XY pair.

Here is a modified version of our function to handle this case.

```
Function LoadAndGraphXY(fileName, pathName)
    String fileName      // Name of file to load or "" to get dialog
    String pathName     // Name of path or "" to get dialog

    // load the waves and set the globals
    LoadWave/J/D/O/P=$pathName fileName
    if (V_flag==0)      // No waves loaded. Perhaps user canceled.
        return -1
    endif
```

Chapter II-9 — Importing and Exporting Data

```
// Put the names of the waves into string variables.  
String sx0, sy0, sx1, sy1  
sx0 = StringFromList(0, S_waveNames)  
sy0 = StringFromList(1, S_waveNames)  
sx1 = StringFromList(2, S_waveNames)  
sy1 = StringFromList(3, S_waveNames)  
  
Wave x0 = $sx0                                // Create wave references.  
Wave y0 = $sy0  
Wave x1 = $sx1  
Wave y1 = $sy1  
  
SetScale d 0, 0, "s", x0, x1      // Set wave data units  
SetScale d 0, 0, "V", y0, y1  
  
Display y0 vs x0                      // Create a new graph  
AppendToGraph y1 vs x1  
  
Textbox/A=LT "Waves loaded from " + S_fileName // Annotate graph  
  
return 0          // Signifies success.  
End
```

The main difference between this and the waveform-based LoadAndGraph function is that here we append waves to the graph as XY pairs. Also, we don't set the X scaling of the waves because we are treating them as XY pairs, not as waveforms.

It is possible to write a more general function that can handle any number of XY pairs. Once again, adding generality adds complexity. Here is the more general version of the function.

```
Function LoadAndGraphXY(fileName, pathName)  
    String fileName      // Name of file to load or "" to get dialog  
    String pathName      // Name of path or "" to get dialog  
  
    // Load the waves and set the globals  
    LoadWave/J/D/O/P=$pathName fileName  
    if (V_flag==0)        // No waves loaded. Perhaps user cancelled.  
        return -1  
    endif  
  
    Display              // Create a new graph  
  
    String sww, syw  
    Variable index=0  
    do                  // Now append waves to graph  
        sww=StringFromList(index, S_waveNames) // Next name  
        if (strlen(sww) == 0)                 // No more?  
            break                // break out of loop  
        endif  
        syw=StringFromList(index+1, S_waveNames) // Next name  
  
        Wave xw = $sww                    // Create wave references.  
        Wave yw = $syw  
  
        SetScale d 0, 0, "s", xw      // Set X wave's units  
        SetScale d 0, 0, "V", yw      // Set Y wave's units  
        AppendToGraph yw vs xw  
  
        index += 2  
    while (1)           // Unconditionally loop back up to "do"  
  
    // Annotate graph  
    Textbox/A=LT "Waves loaded from " + S_fileName
```