Chapter

IV-5

User-Defined Menus

Overview

You can add your own menu items to many Igor menus by writing a menu definition in a procedure window. A simple menu definition looks like this:

```
Menu "Macros"
   "Load Data File/1"
   "Do Analysis/2"
   "Print Report"
End
```

This adds three items to the Macros menu. If you choose Load Data File or press Command-1 (*Macintosh*) or Ctrl+1 (*Windows*), Igor executes the procedure LoadDataFile which, presumably, you have written in a procedure window. The command executed when you select a particular item is derived from the text of the item. This is an *implicit* specification of the item's execution text.

You can also *explicitly* specify the execution text:

```
Menu "Macros"
   "Load Data File/1", Beep; LoadWave/G
   "Do Analysis/2"
   "Print Report"
End
```

Now if you choose Load Data File, Igor will execute "`Beep; LoadWave/G`".

When you choose a user menu item, Igor checks to see if there is execution text for that item. If there is, Igor executes it. If not, Igor makes a procedure name from the menu item string. It does this by removing any characters that are not legal characters in a procedure name. Then it executes the procedure. For example, choosing an item that says

```
"Set Sampling Rate..."
```

executes the SetSamplingRate procedure.

If a procedure window is the top window and if Option (*Macintosh*) or Alt (*Windows*) is pressed when you choose a menu item, Igor tries to find the procedure in the window, rather than executing it.

A menu definition can add submenus as well as regular menu items.

```
Menu "Macros"
   Submenu "Load Data File"
      "Text File"
      "Binary File"
   End

   Submenu "Do Analysis"
      "Method A"
      "Method B"
   End

   "Print Report"
End
```

This adds three items to the Macros menu, two submenus and one regular item. You can nest submenus to any depth.

Menu Definition Syntax

The syntax for a menu definition is:

```
Menu <Menu title string> [,<menu options>]
   [<Menu help strings>]
   <Menu item string> [,<menu item flags>] [,<execution text>]
   [<Item help strings>]
   …
   Submenu <Submenu title string>
      [<Submenu help strings>]
      <Submenu item string> [,<execution text>]
      [<Item help strings>]
      …
   End
End
```

`<Menu title string>` is the title of the menu to which you want to add items. Often this will be Macros but you can also add items to Analysis, Misc and many other built-in Igor menus, including some submenus and the graph marquee and layout marquee menus. If `<Menu title string>` is not the title of a built-in menu then Igor creates a new main menu on the menu bar.

<Menu options> are optional comma-separated keywords that change the behavior of the menu. The allowed keywords are `dynamic`, `hideable`, and `contextualmenu`. For usage, see **Dynamic Menu Items** (see page IV-120), **HideIgorMenus** (see page V-294), and **PopupContextualMenu** (see page V-639) respectively.

`<Menu help strings>` specifies the help for the menu title. This is optional. As of Igor7, menu help is not supported and the menu help specification, if present, is ignored.

`<Menu item string>` is the text to appear for a single menu item, a semicolon-separated string list to define **Multiple Menu Items** (see page IV-122), or **Specialized Menu Item Definitions** (see page IV-123) such as a color, line style, or font menu.

<Menu item flags> are optional flags that modify the behavior of the menu item. The only flag currently supported is /Q, which prevents Igor from storing the executed command in the history area. This is useful for menu commands that are executed over and over through a keyboard shortcut. This feature was introduced in Igor Pro 5. Using it will cause errors in earlier versions of Igor. Menus defined with the `contextualmenu` keyword implicitly set all the menu item flags in the entire menu to /Q; it doesn't matter whether /Q is explicitly set or not, the executed command is not stored in the history area.

`<Execution text>` is an Igor command to execute for the menu item. If omitted, Igor makes a procedure name from the menu item string and executes that procedure. Use "" to prevent command execution (useful only with PopupContextualMenu/N).

`<Item help strings>` specifies the help for the menu item. This is optional. As of Igor7, menu help is not supported and the menu item help specification, if present, is ignored.

The Submenu keyword introduces a submenu with `<Submenu title string>` as its title. The submenu continues until the next End keyword.

`<Submenu item string>` acts just like `<Menu item string>`.

Built-in Menus That Can Be Extended

Here are the titles of built-in Igor menus to which you can add items:

Add Controls	AllTracesPopup	Analysis	Append to Graph
Control	Data	Edit	File
Gizmo	Graph	GraphMarquee	GraphPopup
Help	Layout	LayoutMarquee	Load Waves
Macros	Misc	Statistics	New
Notebook	Open File	Panel	Procedure
Save Waves	Table	TracePopup	

These menu titles must appear in double quotes when used in a menu definition.

Use these menu titles to identify the menu to which you want to append items even if you are working with a version of Igor translated into a language other than English.

The GraphMarquee, LayoutMarquee, TracePopup, AllTracesPopup, and GraphPopup menus are contextual menus. See **Marquee Menus** on page IV-129 and **Trace Menus** on page IV-129. The contextual GraphPopup appears when you control-click (*Macintosh*) or right-click (*Windows*) in a graph, away from any trace, while in operate mode.

All other Igor menus, including menus added by XOPs, can not accept user-defined items.

The **HideIgorMenus** operation (see page V-294) and the **ShowIgorMenus** operation (see page V-741) hide or show most of the built-in main menus (not the Marquee and Popup menus). User-defined menus that add items to built-in menus are normally not hidden or shown by these operations. When a built-in menu is hidden, the user-defined menu items create a user-defined menu with only user-defined items. For example, this user-defined menu:

```
Menu "Table"
    "Append Columns to Table...", DoIgorMenu "Table", "Append Columns to Table"
End
```

will create a Table menu with only one item in it after the HideIgorMenus "Table" command is executed.

To have your user-defined menu items hidden along with the built-in menu items, add the hideable keyword after the Menu definition:

```
Menu "Table", hideable
    "Append Columns to Table...", DoIgorMenu "Table", "Append Columns to Table"
End
```

Adding a New Main Menu

You can add an entirely new menu to the main menu bar by using a menu title that is not used by Igor. For example:

```
Menu "Test"
    "Load Data File"
    "Do Analysis"
    "Print Report"
End
```

Dynamic Menu Items

In the examples shown so far all of the user-defined menu items are static. Once defined, they never change. This is sufficient for the vast majority of cases and is by far the easiest way to define menu items.

Igor also provides support for dynamic user-defined menu items. A dynamic menu item changes depending on circumstances. The item might be enabled under some circumstances and disabled under others. It might be checked or deselected. Its text may toggle between two states (e.g. “Show Tools” and “Hide Tools”).

Because dynamic menus are much more difficult to program than static menus and also slow down Igor’s response to a menu-click, we recommend that you keep your use of dynamic menus to a minimum. The effort you expend to make your menu items dynamic may not be worth the time you spend to do it.

For a menu item to be dynamic, you must define it using a string expression instead of the literal strings used so far. Here is an example.

```
Function DoAnalysis()
   Print "Analysis Done"
End

Function ToggleTurboMode()
   Variable prevMode = NumVarOrDefault("root:gTurboMode", 0)
   Variable/G root:gTurboMode = !prevMode
End

Function/S MacrosMenuItem(itemNumber)
   Variable itemNumber

   Variable turbo = NumVarOrDefault("root:gTurboMode", 0)

   if (itemNumber == 1)
      if (strlen(WaveList("*", ";", ""))==0) // any waves exist?
         return "(Do Analysis"   // disabled state
      else
         return "Do Analysis"    // enabled state
      endif
   endif

   if (itemNumber == 2)
      if (turbo)
         return "!"+num2char(18)+"Turbo"  // Turbo with a check
      else
         return "Turbo"
      endif
   endif
End

Menu "Macros", dynamic
   MacrosMenuItem(1)
   MacrosMenuItem(2), /Q, ToggleTurboMode()
End
```

In this example, the text for the menu item is computed by the MacrosMenuItem function. It computes text for item 1 and for item 2 of the menu. Item 1 can be enabled or disabled. Item 2 can be checked or unchecked.

The dynamic keyword specifies that the menu definition contains a string expression that needs to be reevaluated each time the menu item is drawn. This rebuilds the user-defined menu each time the user clicks in the menu bar. Under the current implementation, it rebuilds *all* user menus each time the user clicks in the menu bar if *any* user-defined menu is declared dynamic. If you use a large number of user-defined items, the time to rebuild the menu items may be noticeable.

There is another technique for making menu items change. You define a menu item using a string expression rather than a literal string but you do not declare the menu dynamic. Instead, you call the BuildMenu operation whenever you need the menu item to be rebuilt. Here is an example:

```
Function ToggleItem1()
   String item1Str = StrVarOrDefault("root:MacrosItem1Str","On")
   if (CmpStr(item1Str,"On") == 0)      // Item is now "On"?
```

```
      String/G root:MacrosItem1Str = "Off"
   else
      String/G root:MacrosItem1Str = "On"
   endif
   BuildMenu "Macros"
End

Menu "Macros"
   StrVarOrDefault("root:MacrosItem1Str","On"), /Q, ToggleItem1()
End
```

Here, the menu item is controlled by the global string variable MacrosItem1Str. When the user chooses the menu item, the ToggleItem1 function runs. This function changes the MacrosItem1Str string and then calls BuildMenu, which rebuilds the user-defined menu the next time the user clicks in the menu bar. Under the current implementation, it rebuilds *all* user-defined menus if BuildMenu is called for *any* user-defined menu.

Optional Menu Items

A dynamic user-defined menu item *disappears* from the menu if the menu item string expression evaluates to ""; the remainder of the menu definition line is then ignored. This makes possible a variable number of items in a user-defined menu list. This example adds a menu listing the names of up to 8 waves in the current data folder. If the current data folder contains less than 8 waves, then only those that exist are shown in the menu:

```
Menu "Waves", dynamic
   WaveName("",0,4), DoSomething($WaveName("",0,4))
   WaveName("",1,4), DoSomething($WaveName("",1,4))
   WaveName("",2,4), DoSomething($WaveName("",2,4))
   WaveName("",3,4), DoSomething($WaveName("",3,4))
   WaveName("",4,4), DoSomething($WaveName("",4,4))
   WaveName("",5,4), DoSomething($WaveName("",5,4))
   WaveName("",6,4), DoSomething($WaveName("",6,4))
   WaveName("",7,4), DoSomething($WaveName("",7,4))
End

Function DoSomething(w)
   Wave/Z w

   if( WaveExists(w) )
      Print "DoSomething: wave's name is "+NameOfWave(w)
   endif
End
```

This works because WaveName returns "" if the indexed wave doesn’t exist.

Note that each potential item must have a menu definition line that either appears or disappears.

Multiple Menu Items

A menu item string that contains a semicolon-separated “string list” (see **StringFromList** on page V-865) generates a menu item for each item in the list. For example:

```
Menu "Multi-Menu"
   "first item;second item;", DoItem()
End
```

Multi-Menu is a two-item menu. When either item is selected the DoItem procedure is called.

This begs the question: How does the DoItem procedure know which item was selected? The answer is that DoItem must call the **GetLastUserMenuInfo** operation (see page V-264) and examine the appropriate returned variables, usually V_value (the selected item’s one-based item number) or S_Value (the selected item’s text).

The string list can be dynamic, too. The above "Waves" example can be rewritten to handle an arbitrary number of waves using this definition:

```
Menu "Waves", dynamic
   WaveList("*",";",""), DoItem()
End

Function DoItem()
   GetLastUserMenuInfo           // Sets S_value, V_value, etc.
   WAVE/Z w= $S_value
   if( WaveExists(w) )
      Print "The wave's name is "+NameOfWave(w)
   endif
End
```

Consolidating Menu Items Into a Submenu

It is common to have many utility procedure files open at the same time. Each procedure file could add menu items which would clutter Igor's menus. When you create a utility procedure file that adds multiple menu items, it is usually a good idea to consolidate all of the menu items into one submenu. Here is an example.

Let's say we have a procedure file full of utilities for doing frequency-domain data analysis and that it contains the following:

```
Function ParzenDialog()
   …
End

Function WelchDialog()
   …
End

Function KaiserDialog()
   …
End
```

We can consolidate all of the menu items into a single submenu in the Analysis menu:

```
Menu "Analysis"
   Submenu "Windows"
      "Parzen…", ParzenDialog()
      "Welch…", WelchDialog()
      "Kaiser…", KaiserDialog()
   End
End
```

Specialized Menu Item Definitions

A menu item string that contains certain special values adds a specialized menu such as a color menu.

Only one specialized menu item string is allowed in each menu or submenu, it must be the first item, and it must be the only item.

Menu Item String	Result
"*CHARACTER*	"Character menu, no character is initially selected, font is Geneva, font size is 12.
"*CHARACTER*(Symbol)	"Character menu of Symbol font.
"*CHARACTER*(Symbol,36)	"Character menu of Symbol font in 36 point size.
"*CHARACTER*(,36)	"Character menu of Geneva font in 36 point size.

Menu Item String	Result
"*CHARACTER*(Symbol,36,p)	"Character menu of Symbol font in 36 point size, initial character is p (π).
"*COLORTABLEPOP*	"Color table menu, initial table is Grays.
"*COLORTABLEPOP*(YellowHot)	"Color table menu, initial table is YellowHot. See **CTabList** on page V-100 for a list of color tables.
"*COLORTABLEPOP*(YellowHot,1)	"Color table menu with the colors drawn reversed.
"*COLORPOP*	"Color menu, initial color is black.
"*COLORPOP*(0,65535,0)	"Color menu, initial color is green.
"*FONT*	"Font menu, no font is initially selected, does not include "default" as a font choice.
"*FONT*(Arial)	"Font menu, Arial is initially selected.
"*FONT*(Arial,default)	"Font menu with Arial initially selected and including "default" as a font choice.
"*LINESTYLEPOP*	"Line style menu, no line style is initially selected.
"*LINESTYLEPOP*(3)	"Line style menu, initial line style is style=3 (coarse dashed line).
"*MARKERPOP*	"Marker menu, no marker is initially selected.
"*MARKERPOP*(8)	"Marker menu, initial marker is 8 (empty circle).
"*PATTERNPOP*	"Pattern menu, no pattern is initially selected.
"*PATTERNPOP*(1)	"Pattern menu, initial pattern is 1 (SW-NE light diagonal).

To retrieve the selected color, line style, etc., the execution text must be a procedure that calls the **GetLastUserMenuInfo** operation (see page V-264). Here's an example of a color submenu implementation:

```
Menu "Main", dynamic
   "First Item", /Q, Print "First Item"
   Submenu "Color"
      CurrentColor(), /Q, SetSelectedColor() // Must be first submenu item
      // No items allowed here
   End
End

Function InitializeColors()
   NVAR/Z red= root:red
   if(!NVAR_Exists(red))
      Variable/G root:red=65535, root:green=0, root:blue=1, root:alpha=65535
   endif
End

Function/S CurrentColor()
   InitializeColors()
   NVAR red = root:red
   NVAR green = root:green
   NVAR blue = root:blue
   NVAR alpha = root:alpha
   String menuText
   sprintf menuText, "*COLORPOP*(%d,%d,%d,%d)", red, green, blue, alpha
   return menuText
End

Function SetSelectedColor()
   GetLastUserMenuInfo// Sets V_Red, V_Green, V_Blue, V_Alpha, S_value, V_value
   NVAR red = root:red
   NVAR green = root:green
   NVAR blue = root:blue
   NVAR alpha = root:alpha
   red = V_Red
   green = V_Green
   blue = V_Blue
   alpha = V_Alpha

   Make/O/N=(2,2,4) root:colorSpot
   Wave colorSpot = root:colorSpot
   colorSpot[][][0] = V_Red
   colorSpot[][][1] = V_Green
   colorSpot[][][2] = V_Blue
   colorSpot[][][3] = V_Alpha

   CheckDisplayed/A colorSpot
   if (V_Flag == 0)
      NewImage colorSpot
   endif
End
```

Special Characters in Menu Item Strings

You can control some aspects of a menu item using special characters. These special characters are based on the behavior of the Macintosh menu manager and are only partially supported on Windows (see **Special Menu Characters on Windows** on page IV-126). They affect user-defined menus in the main menu bar. On Macintosh, but not on Windows, they also affect user-defined pop-up menus in control panels, graphs and simple input dialogs.

By default, special character interpretation is enabled in user-defined menu bar menus and is disabled in user-defined control panel, graph and simple input dialog pop-up menus. This is almost always what you would want. In some cases, you might want to override the default behavior. This is discussed under **Enabling and Disabling Special Character Interpretation** on page IV-127.

This table shows the special characters and their effect if special character interpretation is enabled. See **Special Menu Characters on Windows** on page IV-126 for Windows-specific considerations.

Character	**Behavior**
`/`	Creates a keyboard shortcut for the menu item. The character after the slash defines the item's keyboard shortcut. For example, `"Low Pass/1"` makes the item "Low Pass" with a keyboard shortcut for Command-1 (*Macintosh*) or Ctrl-1 (*Windows*). You can also use function keys. To avoid conflicts with Igor, use the numeric keys and the function keys only. See **Keyboard Shortcuts** on page IV-127 and **Function Keys** on page IV-128 for further details. Keyboard shortcuts are not supported in the graph marquee and layout marquee menus.
`-`	Creates a divider between menu items. If a hyphen (minus sign) is the first character in the item then the item will be a disabled divider. This can be a problem when trying to put negative numbers in a menu. Use a leading space character to prevent this. The string "(-" also disables the corresponding divider.
`(`	Disables the menu item. If the first character of the item text is a left parenthesis then the item will be disabled.

Character	Behavior

`!`	Adds a mark to the menu item.
	If an exclamation point appears in the item, any character after the exclamation point adds a checkmark to the left of the menu item. For example:
	`"Low Pass!*"`
	makes an item "Low Pass" with an checkmark to the left.
	For compatibility with Igor6, use:
	`"Low Pass!" + num2char(18)`

`;`	Separates one menu item from the next. Example: `"Item 1;Item 2"`

Whereas it is standard practice to use a semicolon to separate items in a pop-up menu in a control panel, graph or simple input dialog, you should avoid using the semicolon in user-defined main-menu-bar menus. It is clearer if you use one item per line. It is also necessary in some cases (see **Menu Definition Syntax** on page IV-118).

If special character interpretation is disabled, these characters will appear in the menu item instead of having their special effect. The semicolon character is treated as a separator of menu items even if special character interpretation is disabled.

Special Menu Characters on Windows

On Windows, these characters are treated as special in menu bar menus but not in pop-up menus in graphs, control panels, and simple input dialogs. The following table shows which special characters are supported.

Character	Meaning
`/`	Defines accelerator
`-`	Divider
`(`	Disables item
`!`	Adds mark to item
`;`	Separates items

In general, Windows does not allow using Ctrl+<punctuation> as an accelerator. Therefore, in the following example, the accelerator will not do anything:

```
Menu "Macros"
   "Test/["       // "/[" will not work on Windows.
End
```

On Windows, you can designate a character in a menu item as a mnemonic keystroke by preceding the character with an ampersand:

```
Menu "Macros"
   "&Test", Print "This is a test"
End
```

This designates “T” as the mnemonic keystroke for Test. To invoke this menu item, press Alt and release it, press the “M” key to highlight the Macros menu, and press the “T” key to invoke the Test item. If you hold the Alt key pressed while pressing the “M” and “T” keys and if the active window is a procedure window, Igor will not execute the item’s command but rather will bring up the procedure window and display the menu definition. This is a programmer’s shortcut.

Note: The mnemonic keystroke is not supported on Macintosh. For this reason, if you care about cross-platform compatibility, you should not use ampersands in your menu items.

On Macintosh, if you include a single ampersand in a menu item, it does not appear in the menu item. If you use a double ampersand, it appears as a single ampersand.

Enabling and Disabling Special Character Interpretation

The interpretation of special characters in menu items can sometimes get in the way. For example, you may want a menu item to say "m/s" or "A<B". With special character interpretation enabled, the first of these would become "m" with "s" as the keyboard shortcut and the second would become "A" in a bold typeface.

Igor provides WaveMetrics-defined escape sequences that allow you to override the default state of special character interpretation. These escape sequences are case sensitive and must appear at the very start of the menu item:

Escape Code	Effect	Example
`"\\M0"`	Turns special character interpretation off.	`"\\M0m/s"`
`"\\M1"`	Turns special character interpretation on.	`"\\M1m/s"`

The most common use for this will be in a user-defined menu bar menu in which the default state is on and you want to display a special character in the menu item text itself. That is what you can do with the "`\\M0`" escape sequence.

Another possible use on Macintosh is to create a disabled menu item in a control panel, graph or simple input dialog pop-up menu. The default state of special character interpretation in pop-up menus is off. To disable the item, you either need to turn it on, using "\\M1" or to use the technique described in the next paragraph.

What if we want to include a special character in the menu item itself *and* have a keyboard shortcut for that item? The first desire requires that we turn special character interpretation off and the second requires that we turn it on. The WaveMetrics-defined escape sequence can be extended to handle this. For example:

```
"\\M0:/1:m/s"
```

The initial "\\M0" turns normal special character interpretation off. The first colon specifies that one or more special characters are coming. The /1 makes Command-1 (*Macintosh*) or Ctrl+1 (*Windows*) the keyboard shortcut for this item. The second colon marks the end of the menu commands and starts the regular menu text which is displayed in the menu without special character interpretation. The final result is as shown above.

Any of the special characters can appear between the first colon and the second. For example:

```
Menu "Macros"
   "\\M0:/1:(Cmd-1 on Macintosh, Ctrl+1 on Windows)"
   "\\M0:(:(Disabled item)"
   "\\M0:!*:(Checked)"
   "\\M0:/2!*:(Cmd-2 on Macintosh, Ctrl+2 on Windows and checked)"
End
```

The \\M escape code affects just the menu item currently being defined. In the following example, special character interpretation is enabled for the first item but not for the second:

```
"\\M1(First item;(Second item"
```

To enable special character interpretation for both items, we need to write:

```
"\\M1(First item;\\M1(Second item"
```

Keyboard Shortcuts

A keyboard shortcut is a set of one or more keys which invoke a menu item. In a menu item string, a keyboard shortcut is introduced by the / special character. For example:

```
Menu "Macros"
   "Test/1"       // The keyboard shortcut is Cmd-1 (Macintosh)
End               // or Ctrl+1 (Windows).
```

All of the plain alphabetic keyboard shortcuts (/A through /Z) are used by Igor.

Numeric keyboard shortcuts (/0 through /9) are available for use in user menu definitions as are Function Keys, described below.

You can define a numeric keyboard shortcut that includes one or more modifier keys. The modifier keys are:

Macintosh:	Shift (S), Option (O), Control (L)
Windows:	Shift (S), Alt (O), Meta (L) (Meta is often called the “Windows key”.

For example:

```
Menu "Macros"
   "Test/1"        // Cmd-1, Ctrl+1
   "Test/S1"       // Shift-Cmd-1, Ctrl+Shift+1.
   "Test/O1"       // Option-Cmd-1, Ctrl+Alt+1
   "Test/OS1"      // Option-Shift-Cmd-1, Ctrl+Shift+Alt+1
End
```

Function Keys

Most keyboards have function keys labeled F1 through F12. In Igor, you can treat a function key as a keyboard shortcut that invokes a menu item.

Note: Mac OS X reserves nearly all function keys for itself. In order to use function keys for an application, you must check a checkbox in the Keyboard control panel. Even then the OS will intercept some function keys.

Note: On Windows, Igor uses F1 for help-related operations. F1 will not work as a keyboard shortcut on Windows. Also, the Windows OS reserves Ctrl-F4 and Ctrl-F6 for closing and reordering windows. On some systems, F12 is reserved for debugging.

Here is a simple function key example:

```
Menu "Macros"
   "Test/F5"       // The keyboard shortcut is F5.
End
```

As with other keyboard shortcuts, you can specify that one or more modifier keys must be pressed along with the function key. By including the “C” modifier character, you can specify that Command (*Macintosh*) or Ctrl (*Windows*) must also be pressed:

```
Menu "Macros"
   // Function keys without modifiers
   "Test/F5"       // F5

   // Function keys with Shift and/or Option/Alt modifiers
   "Test/SF5"      // Shift-F5 (Macintosh), Shift+F5 (Windows)
   "Test/OF5"      // Option-F5, Alt+F5
   "Test/SOF5"     // Shift-Option-F5, Shift+Alt+F5

   // Function keys with Command (Macintosh) or Ctrl (Windows) modifiers
   "Test/CF5"      // Cmd-F5, Ctrl+F5
   "Test/SCF5"     // Shift-Cmd-F5, Shift-Ctrl-F5
   "Test/OCF5"     // Option-Cmd-F5, Alt-Ctrl-F5
   "Test/OSCF5"    // Option-Shift-Cmd-F5, Alt-Shift-Ctrl-F5

   // Function keys with Ctrl (Macintosh) or Meta (Windows) modifiers
   "Test/LOSCF5"  // Control-Option-Shift-Cmd-F5, Meta+Alt+Shift+Ctrl+F5
End
```

Although some keyboards have function keys labeled F13 and higher, they do not behave consistently and are not supported.

Marquee Menus

Igor has two menus called “marquee menus”. In graphs and page layouts you create a marquee when you drag diagonally. Igor displays a dashed-line box indicating the area of the graph or layout that you have selected. If you click inside the marquee, you get a marquee menu.

You can add your own menu items to a marquee menu by creating a GraphMarquee or LayoutMarquee menu definition. For example:

```
Menu "GraphMarquee"
   "Print Marquee Coordinates", GetMarquee bottom; Print V_left, V_right
End
```

The use of keyboard shortcuts is not supported in marquee menus.

See **Marquee Menu as Input Device** on page IV-151 for details.

Trace Menus

Igor has two “trace” menus named “TracePopup” and “AllTracesPopup”. When you control-click or right-click in a graph on a trace you get the TracesPopup menu. If you press Shift while clicking, you get the AllTracesPopup (standard menu items in that menu operated on all the traces in the graph). You can append menu items to these menus with Menu “TracePopup” and Menu “AllTracesPopup” definitions.

For example, this code adds an Identify Trace item to the TracePopup contextual menu but not to the AllTracesPopup menu:

```
Menu "TracePopup"
   "Identify Trace", /Q, IdentifyTrace()
End

Function IdentifyTrace()
   GetLastUserMenuInfo
   Print S_graphName, S_traceName
End
```

GraphPopup Menu

Igor has a contextual menu named "GraphPopup". When you control-click or right-click in a graph away from any trace, you get the GraphPopup menu. You can append menu items to this menu with a GraphPopup menu definition.

For example, the following code adds an "Identify Graph" item to the GraphPopup contextual menu:

```
Menu "GraphPopup"
   "Identify Graph", /Q, IdentifyGraph()
End

Function IdentifyGraph()
   Print WinName(0,1)
End
```

Popup Contextual Menus

You can create a custom pop-up contextual menu to respond to a control-click or right-click. For an example, see **Creating a Contextual Menu** on page IV-149.