```
    Print "Callback: ", list, selectedText, selectedItem
End
```

Example 6 - popupStr contains the name of a user-defined menu, asynchronous popup result

```
Function YourFunction()
    PopupContextualMenu/ASYN/N "ForContext"
    (YourFunction continues...)
End

// Selection result is handled in "ForContext" menu's execution texts, as in Example 4
```

See Also

Creating a Contextual Menu on page IV-162, **User-Defined Menus** on page IV-125.

Special Characters in Menu Item Strings on page IV-133.

Chapter III-14, **Controls and Control Panels**, for details about control panels and controls.

The **SetWindow** and **PopupMenu** operations.

PopupMenu

PopupMenu [**/Z**] ***ctrlName*** [***keyword = value*** [**,** ***keyword = value*** …]]

The PopupMenu operation creates or modifies a pop-up menu control in the target or named window.

For information about the state or status of the control, use the **ControlInfo** operation.

Parameters

ctrlName is the name of the PopupMenu control to be created or changed.

The following keyword=value parameters are supported:

align=*alignment*

Sets the alignment mode of the control. The alignment mode controls the interpretation of the *leftOrRight* parameter to the pos keyword. The align keyword was added in Igor Pro 8.00.

If *alignment*=0 (default), *leftOrRight* specifies the position of the left end of the control and the left end position remains fixed if the control size is changed.

If *alignment*=1, *leftOrRight* specifies the position of the right end of the control and the right end position remains fixed if the control size is changed.

appearance={*kind* [, *platform*]}

Sets the appearance of the control. *platform* is optional. Both parameters are names, not strings.

kind can be one of `default`, `native`, or `os9`.

platform can be one of `Mac`, `Win`, or `All`.

See **Button** and **DefaultGUIControls** for more appearance details.

bodyWidth=*width*

Specifies an explicit size for the body (nontitle) portion of a PopupMenu control. By default (bodyWidth=0), the body portion autosizes depending on the current text. If you supply a bodyWidth>0, then the body is fixed at the size you specify regardless of the body text. This makes it easier to keep a set of controls right aligned when experiments are transferred between Macintosh and Windows, or when the default font is changed.

disable=*d*

Sets user editability of the control.

d=0: Normal.

d=1: Hide.

d=2: Draw in gray state; disable control action.

fColor=(*r*,*g*,*b*[,*a*]) Sets the initial color of the title. *r*, *g*, *b*, and *a* specify the color and optional opacity as **RGBA Values**. The default is opaque black.

To further change the color of the title text, use escape sequences as described for title=*titleStr*.

focusRing=*fr* Enables or disables the drawing of a rectangle indicating keyboard focus:

fr=0: Focus rectangle will not be drawn.

fr=1: Focus rectangle will be drawn (default).

On Macintosh, regardless of this setting, the focus ring appears if you have enabled full keyboard access via the Shortcuts tab of the Keyboard system preferences.

font="*fontName*" Sets the font used for the pop-up title, e.g., `font="Helvetica"`.

fsize=*s* Sets the font size for the pop-up title.

fstyle=*fs* *fs* is a bitwise parameter with each bit controlling one aspect of the font style as follows:

Bit 0: Bold

Bit 1: Italic

Bit 2: Underline

Bit 4: Strikethrough

See **Setting Bit Parameters** on page IV-12 for details about bit settings.

help={*helpStr*} Sets the help for the control.

helpStr is limited to 1970 bytes (255 in Igor Pro 8 and before).

You can insert a line break by putting "\r" in a quoted string.

mode=*m* Specifies the pop-up title location.

m=0: Title is in pop-up menu.

m=1: Title is to the left of pop-up menu, the chosen menu item appears in the pop-up menu, and menu item number *m* is initially selected.

noproc Specifies that no procedure is to execute when choosing in the pop-up menu.

popColor=(*r*,*g*,*b*[,*a*]) Specifies the color initially chosen in the color pop-up palette. *r*, *g*, *b*, and *a* specify the color and optional opacity as **RGBA Values**. See the **Colors, Color Tables, Line Styles, Markers, and Patterns** section.

popmatch=*matchStr*

Sets mode to the enabled menu item that matches *matchStr*. *matchStr* may be a "wildcard" expression. See **StringMatch**. If no item is matched, mode is unchanged.

popvalue=*valueStr* Sets the string displayed by the menu when first created, if mode is not zero. See **Popvalue Keyword** section.

pos={*leftOrRight*,*top*} Sets the position in **Control Panel Units** of the top/left corner of the control if its alignment mode is 0 or the top/right corner of the control if its alignment mode is 1. See the align keyword above for details.

pos+={*dx*,*dy*} Offsets the position of the pop-up in **Control Panel Units**.

proc=*procName* Specifies the procedure to execute when the pop-up menu is clicked. See ***Pop-up Menu Action Procedure*** below.

rename=*newName* Gives pop-up menu a new name.

size={*width*,*height*} Sets pop-up menu size in **Control Panel Units**.

title=*titleStr* Sets title of pop-up menu to the specified string expression. Defaults to "" (no title).

Using escape codes you can change the font, size, style, and color of the title. See **Annotation Escape Codes** on page III-53 or details.

userdata(*UDName*)=*UDStr*

Sets the unnamed user data to *UDStr*. Use the optional (*UDName*) to specify a named user data to create. You can retrieve the data using **GetUserData**.

userdata(*UDName*)+=*UDStr*

Appends *UDStr* to the current unnamed user data. Use the optional (*UDName*) to append to the named *UDStr*.

value=*itemListSpec* Specifies the pop-up menu's items. *itemListSpec* can take several forms as described below under ***Setting The Popup Menu Items***.

win=*winName* Specifies which window or subwindow contains the named control. If not given, then the top-most graph or panel window or subwindow is assumed.

When identifying a subwindow with *winName*, see **Subwindow Syntax** on page III-92 for details on forming the window hierarchy.

Flags

/Z No error reporting.

Details

The target window, or the window named with the win=*winName* keyword, must be a graph or panel.

Pop-up Menu Action Procedure

The action procedure for a pop-up menu control takes a predefined WMPopupAction structure as a parameter to the function:

```
Function PopupMenuAction(PU_Struct) : PopupMenuControl
   STRUCT WMPopupAction &PU_Struct
   ...
   return 0
End
```

The ": PopupMenuControl" designation tells Igor to include this procedure in the list of available popup menu action procedures in the PopupMenu Control dialog used to create a popup menu.

See **WMPopupAction** for details on the WMPopupAction structure.

Although the return value is not currently used, action procedures should always return zero.

You may see an old format pop-up menu action procedure in old code:

```
Function PopupMenuAction (ctrlName,popNum,popStr) : PopupMenuControl
   String ctrlName
   Variable popNum     // which item is currently selected (1-based)
   String popStr       // contents of current popup item as string
   ...
   return 0
End
```

This old format should not be used in new code.

Setting The Popup Menu Items

This section discusses popup menus containing lists of text items. The next section discusses popup menus for choosing colors, line styles, markers and patterns.

The items in the popup menu are determined by the *itemListSpec* parameter used with the value keyword. *itemListSpec* can take several different forms from simple to complex.

No matter what the form, Igor winds up storing an expression that returns a string in the popup menu's internal structure. This expression may be a literal string ("Red;Green;Blue;"), a call to a built-in or user-defined function that returns a string, or the path to a global string variable. Igor evaluates this expression

when the popup menu is first created and again each time the user clicks on the menu. You can see the string expression for a given popup menu using the PopupMenu Control dialog.

The right form for *itemListSpec* depends on your application. Here is a guide to choosing the right form with the simpler forms first.

A literal string expression

Use this if you know the items you want in your popup menu when you write the PopupMenu call. For example:

```
Function PopupDemo1()  // Literal string
   NewPanel
   PopupMenu popup0, value="Red;Green;Blue;"
End
```

This method is limited to 2500 bytes of menu item text.

A function call

Use this if you need to compute the popup menu item list when the user clicks the popup menu. The function must return a string containing a semicolon-separated list of menu items. This example creates a popup menu which displays the name of each wave in the current data folder at the time the menu is clicked:

```
Function PopupDemo2()  // Built-in function
   NewPanel
   PopupMenu popup0, value=WaveList("*", ";", "")
End
```

You can also use a user-defined function. This example shows how to list waves from other than the current data folder:

```
Function/S MyPopupWaveList()
   DFREF saveDF

   // Create some waves for demo purposes
   saveDF = GetDataFolderDFR()
   NewDataFolder/O/S root:Packages
   NewDataFolder/O/S PopupMenuDemo
   Make/O demo0, demo1, demo2
   SetDataFolder saveDF

   saveDF = GetDataFolderDFR()
   SetDataFolder root:Packages:PopupMenuDemo
   String list = WaveList("*", ";", "")
   SetDataFolder saveDF

   return list
End

Function PopupDemo3()  // User-defined function
   NewPanel
   PopupMenu popup0, value=MyPopupWaveList()
End
```

 # followed by a local string variable specifying items

Use this when the popup menu item list is not known when you write the code but you can compute it at runtime. For example:

```
Function PopupDemo4()  // Local string variable specifying items
   NewPanel
   String quote = "\""
   String list
   if (CmpStr(IgorInfo(2),"Windows") == 0)
      list = quote + "Windows XP; Windows VISTA;" + quote
   else
      list = quote + "Mac OS X 10.4;Mac OS X 10.5;" + quote
   endif
   PopupMenu popup0, value=#list
End
```

The strange-looking use of the quote string variable is necessary because the parameter passed to the value=# keyword is evaluated once when the PopupMenu command executes and the result of that

evaluation is evaluated again when the PopupMenu is created or clicked. The result of the first evaluation must be a legal string expression.

This method is limited to 2500 bytes of menu item text.

 # followed by a local string variable specifying a function

Use this when you need to compute the popup menu item list at click time and you need to select the function which computes the list when the popup menu is created. For example:

```
Function/S WindowsItemList()
    String list
    list = "Windows XP; Windows VISTA;"
    return list
End

Function/S MacItemList()
    String list
    list = "Mac OS X 10.4;Mac OS X 10.5;"
    return list
End

Function PopupDemo5()  // Local string variable specifying function
    String listFunc
    if (CmpStr(IgorInfo(2),"Windows") == 0)
        listFunc = "WindowsItemList()"
    else
        listFunc = "MacItemList()"
    endif
    NewPanel
    PopupMenu popup0, value=#listFunc
End
```

This form is useful when you create a control panel in an independent module. Since the control panel runs in the global name space, you must specify the independent module name in the invocation of the function that provides the popup menu items. For example:

```
// Calling a non-static function in an independent module from #included code
#pragma IndependentModuleName=IM
. . .
String listFunc= GetIndependentModuleName()+"#PublicFunctionInIndepMod()"
PopupMenu popup0, value=#listFunc

// Calling a static function in an independent module from #included code
#pragma IndependentModuleName=IM
#pragma ModuleName=ModName
. . .
String listFunc= GetIndependentModuleName()+"#ModName#StaticFunctionInIndepMod()"
PopupMenu popup0, value=#listFunc
```

We use GetIndependentModuleName rather than hard-coding the name of the independent module so that the code will continue to work if the name of the independent module is changed. Also, because this code does not depend on the specific name of the independent module, it can be added to an independent module via a #included procedure file.

Also see **GetIndependentModuleName** and **Independent Modules and Popup Menus** on page IV-241.

 # followed by a quoted literal path to a global string variable

Use this if you want to compute the popup menu item list before it is clicked, not each time it is clicked. This would be advantageous if it takes a long time to compute the item list, and the list changes only at well-defined times when you can set the global string variable.

The global string variable must exist when the PopupMenu command executes and when the menu is clicked. In this example, the gPopupMenuItems global string variable is created and initialized when the popup menu is created but can be changed to a different value later before the menu is clicked:

```
Function PopupDemo6()  // Global string variable containing list
    NewDataFolder/O root:Packages
    NewDataFolder/O root:Packages:PopupMenuDemo
    String/G root:Packages:PopupMenuDemo:gPopupMenuItems = "Red;Green;Blue;"

    NewPanel
    PopupMenu popup0 ,value=#"root:Packages:PopupMenuDemo:gPopupMenuItems"
End
```

 # followed by a local string variable containing a path to a global string variable

Use this when the popup menu item list contents will be stored in a global string variable whose location is not known until the popup menu is created. For example:

```
Function PopupDemo7()  // Local string containing path to global string
    String graphName = WinName(0, 1, 1)// Name of top graph
    if (strlen(graphName) == 0)
        Print "There are no graphs."
        return -1
    endif

    NewDataFolder/O root:Packages
    NewDataFolder/O root:Packages:PopupMenuDemo

    // Create data folder for graph
    NewDataFolder/O root:Packages:PopupMenuDemo:$(graphName)

    String list = "Red;Green;Blue;"
    String/G root:Packages:PopupMenuDemo:$(graphName):gPopupMenuItems = list

    NewPanel

    String path         // Local string containing path to global string
    path = "root:Packages:PopupMenuDemo:" + graphName + ":gPopupMenuItems"
    PopupMenu popup0, value=#path

    return 0
End
```

Colors, Color Tables, Line Styles, Markers, and Patterns

You can create PopupMenu controls for color, color tables, line style (dash modes), markers, and patterns. To do so, simply specify the *itemListSpec* parameter to the value keyword as one of "*COLORPOP*", "*COLORTABLEPOP*", "*COLORTABLEPOPNONAMES*", "*LINESTYLEPOP*", "*MARKERPOP*", or "*PATTERNPOP*". In these modes the body of the control will contain a color box, a color table (gradient), a line style sample, a marker, or a pattern sample.

For these special pop-up menus, mode=0 ("Title in Box" checked) is not used.

For a line style pop-up menu, the mode value is the line style number plus one. Thus line style 0 (a solid line) is mode=1.

For a marker pop-up, the mode value is the marker number plus one, and marker 0 (the + marker) is mode=1.

For a pattern pop-up, the mode value is the **SetDrawEnv** fillPat number minus 4, so mode=1 corresponds to fillpat=5, the SW-NE lines fill pattern shown above.

For a color table pop-up, the mode value is the CTabList() index plus 1, so mode=1 corresponds to the first item in the list returned by **CTabList**, which is "Grays":

```
ControlInfo $ctrlName                           // Sets V_Value
Print StringFromList(V_Value-1,CTabList())      // Prints "Grays"
```

ControlInfo also returns the color table name in S_Value.

To set the pop-up to a given color table name, you can use code like this:

```
Variable m = 1 + WhichListItem(ctabName, CTabList())
PopupMenu $ctrlName mode=m
```

For color pop-up menus, you set the current value using the popColor=(*r*,*g*,*b*[,*a*]) keyword. On output (via the popStr parameter of your action procedure or via the S_value output from **ControlInfo**) the color is encoded as "(*r*,*g*,*b*)" or "(r,g,b,a)". To get these numerical values, you can extract them from the string using the MyRGBStrToRGB function below or use ControlInfo which sets V_Red, V_Green, V_Blue and V_Alpha.

The following example demonstrates the line style and color pop-up menus. To run the example, copy the following code to the procedure window of a new experiment and then run the panel macro.

```
Window Panel0() : Panel
    PauseUpdate; Silent 1            // building window …
    NewPanel /W=(150,50,400,182)
    PopupMenu popup0,pos={74,31},size={96,20},proc=ColorPopMenuProc,title="colors"
    PopupMenu popup0,mode=1,popColor= (0,65535,65535),value= "*COLORPOP*"
    PopupMenu popup1,pos={9,68},size={221,20},proc=LStylePopMenuProc
```

```
    PopupMenu popup1,title="line styles",mode=1,value= "*LINESTYLEPOP*"
EndMacro

Function ColorPopMenuProc(ctrlName,popNum,popStr) : PopupMenuControl
    String ctrlName
    Variable popNum
    String popStr

    Variable r,g,b,a
    MyRGBStrToRGB(popStr,r,g,b,a)     // One way to get r, g, b
    print popStr," gives: ",r,g,b,a

    ControlInfo $ctrlName             // Another way: Sets V_Red,V_Green,V_Blue,V_Alpha
    Printf "ControlInfo returned (%d,%d,%d,%d)\r", V_Red, V_Green, V_Blue, V_Alpha

    return 0
End

// Take (r,g,b, and possibly a) as a string and extract numeric r,g,b,a values
Function MyRGBstrToRGB(rgbStr,r,g,b,a)
    String rgbStr
    Variable &r, &g, &b, &a

    if (CmpStr(rgbStr[0], "(") == 0)
        rgbStr = rgbStr[1,inf]
    endif
    r = str2num(StringFromList(0, rgbStr, ","))
    g = str2num(StringFromList(1, rgbStr, ","))
    b = str2num(StringFromList(2, rgbStr, ","))
    a = 65535
    if (itemsinList(rgbStr) > 3)
        a = str2num(StringFromList(3, rgbStr, ","))
    endif
End

Function LStylePopMenuProc(ctrlName,popNum,popStr) : PopupMenuControl
    String ctrlName
    Variable popNum
    String popStr

    Print "style:",popNum-1

    return 0
End
```

Popvalue Keyword

There are times when the displayed value cannot be determined and saved such that it can be displayed when the pop-up menu is recreated. For instance, because window recreation macros are evaluated in the root folder, a pop-up menu of waves may not contain the correct list when a panel is recreated. That is, the intention may be to have the menu show a particular wave from a data folder other than root. When the panel recreation macro runs, the function that lists waves will list waves in the root data folder. The desired selection may be wrong or nonexistent.

Similarly, a pop-up menu of fonts may need to display a particular font upon recreation on a different computer having a different list of fonts. The mode=*m* keyword probably won't pick the correct font from the new list.

The solution to these problems is to save the correct selection with the popvalue=*valueStr* keyword. The list function will not be executed when the menu is first created. If the menu is popped, the list function will be evaluated, and the correct list will be displayed then.

It is a good idea to set the mode=*m* keyword to the correct number, if it is known. That way, when the menu is popped the correct item is chosen.

Normally you can let Igor redraw the pop-up menu when it redraws the graph or control panel containing it. However, there are situations in which you may want to force the pop-up menu to be redrawn. This can be done using the **ControlUpdate** operation.

See Also

Chapter III-14, **Controls and Control Panels**, for details about control panels and controls.

Control Panel Units on page III-444 for a discussion of the units used for controls.

The **GetUserData** function for retrieving named user data.

The **ControlInfo** operation for information about the control.