```
SetWaveTextEncoding /DF={root:,1,root:Packages} /ONLY=0 /TYPE=1 /CONV={1,4} 1, 16

// Mark a text wave as really containing binary data
SetWaveTextEncoding 255, 16, textWaveContainingBinaryData

// Convert Chinese, Japanese and Korean text wave data to UTF-8.
// This example illustrates how to do a conversion based on criteria
// that require inspecting each wave.
Function ConvertCJKToUTF8(dfr, recurse)
    DFREF dfr
    Variable recurse

    Variable index = 0
    do
        Wave/Z w = WaveRefIndexedDFR(dfr, index)
        if (!WaveExists(w))
            break
        endif
        if (WaveType(w) == 0)   // Text wave?
            Variable currentEncoding = WaveTextEncoding(w,5)// Wave content current
    encoding
            switch(currentEncoding)
                case 4:          // Japanese (Shift JIS)
                case 5:          // Traditional Chinese (Big5)
                case 6:          // Simplified Chinese (ISO-2022-CN)
                case 7:          // Macintosh Korean (EUC-KR)
                case 8:          // Windows Korean (Windows-949)
                    SetWaveTextEncoding /CONV=1 1, 16, w
                    break
            endswitch
        endif
        index += 1
    while(1)

    if (recurse)
        Variable numChildDataFolders = CountObjectsDFR(dfr, 4)
        Variable i
        for(i=0; i<numChildDataFolders; i+=1)
            String childDFName = GetIndexedObjNameDFR(dfr, 4, i)
            DFREF childDFR = dfr:$childDFName
            ConvertCJKToUTF8(childDFR, 1)
        endfor
    endif

    SetDataFolder saveDFR
End
```

See Also

Text Encodings on page III-459, **Wave Text Encodings** on page III-472, **Text Encoding Names and Codes** on page III-490, **Text Waves Containing Binary Data** on page III-475

WaveTextEncoding, **ConvertTextEncoding**, **ConvertGlobalStringTextEncoding**

SetWindow

SetWindow ***winName*** [**,** ***keyword = value***]…

The SetWindow operation sets the window note and user data for the named window or subwindow. SetWindow can also set hook functions for a base window or exterior subwindow (interior subwindows not supported).

Parameters

winName can be the name of any target window (graph, table, page layout, notebook, control panel, Gizmo, camera, or XOP target window). It can also be the keyword `kwTopWin` to specify the topmost target window.

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

activeChildFrame=*f* Determines if the frame indicating the active subwindow in the named window or subwindow is drawn.

- *f*=-1: Default. Recursively check ancestors of this window. If one is found with a value of either 0 or 1, use that value. If the topmost window has a value of -1, then the frame is drawn.
- *f*=1: If this subwindow or one of its descendants becomes active, then the frame is drawn.
- *f*=0: If this subwindow or one of its descendants becomes active, then the frame is not drawn.

doScroll=*mode* Controls scrolling mode for top-level graph and panel windows only. The doScroll keyword was added in Igor Pro 9.00.

mode=1: Turns scrolling mode on for the graph or panel.

When scrolling mode is on, the size of the content area of the graph or panel is fixed and the window no longer grows or shrinks when you resize the window. If you shrink the window smaller than the size of the content area, scroll bars appear which allow you to view different parts of the content area.

When scrolling mode is on, you can change the size of the content area of the graph or panel window using doScroll=(width, height).

mode=0: Turns scrolling mode off for the graph or panel.

When scrolling mode is off, scroll bars are not shown and the size of the content area of the graph or panel grows or shrinks when you resize the window.

doScroll=(*width*,*height*) Controls scrolling mode for top-level graph and panel windows only. The doScroll keyword was added in Igor Pro 9.00.

Turns scrolling mode on if it is off. Sets the size of the content area of the graph or panel. *width* and *height* are in units of points.

graphicsTech=*t* Sets the graphics technology used to draw the window. This flag may be useful in rare cases to work around graphics limitations. See **Graphics Technology** (see page III-506) for background information.

- *t*=0: Default: The graphics technology specified in Miscellaneous Settings dialog, Miscellaneous category. This is Qt Graphics by default.
- *t*=1: Native: Core Graphics on Macintosh, GDI+ on Windows.
- *t*=2: Old: Core Graphics on Macintosh, GDI on Windows.

hide=*h* Hides or unhides widows or subwindows.

- *h*=0: Unhides a subwindow or base window.
- *h*=1: Hides a subwindow or base window.
- *h*=2: Unhides without restoring minimized windows (*Windows* only).

When unhiding subwindows, you should combine with needUpdate=1 if conditions require the subwindow to be redrawn since the window was hidden.

hook=*procName* Sets the window hook function that Igor will call when certain events happen. Use `SetWindow hook=$""` to specify no hook function.

See **Unnamed Window Hook Functions** on page IV-305 for further details.

hook(*hName*)=*procName*

Defines a named window hook *hName* and sets the function that Igor will call when certain events happen. *hName* can be any legal name. Named hooks are called before any unnamed hooks.

Use $"" for *procName* to specify no hook.

See **Named Window Hook Functions** on page IV-295 for further details.

To hook a subwindow, see **Window Hooks and Subwindows** on page IV-294.

hookcursor=*number* Sets the mouse cursor. This keyword is antiquated. See **Setting the Mouse Cursor** on page IV-302 for the preferred technique.

hookevents=*flags* Bitfield of flags to enable certain events for the unnamed hook function:

Bit 0:	Mouse button clicks.
Bit 1:	Mouse moved events.
Bit 2:	Cursor moved events.

To set bit 0 and bit 1 (mouse clicks and mouse moved), use $2^0+2^1 = 1+2 = 3$ for *flags*. Use 7 to also enable cursor moved events. See **Setting Bit Parameters** on page IV-12 for details about bit settings.

This keyword applies to the unnamed hook function only. It does not affect named hook functions which always receive all events.

markerHook= {*hookFuncName*, *start*, *end*}

Specifies a user function and marker number range for custom markers. The marker range can be any positive integers less than 1000 and can overlap built-in marker numbers. See **Custom Marker Hook Functions** on page IV-308 for details.

Use $"" for hookFuncName to specify no hook.

needUpdate= *n* Marks a window as needing an update (*n*=1) or takes no action (*n*=0).

note=*noteStr* Sets the window note to *noteStr*, replacing any existing note.

note+=*noteStr* Appends *noteStr* to current contents of the window note.

sizeLimit= {*minWidth*, *minHeight*, *maxWidth*, *maxHeight*}

Imposes limits on a window's size when resized with the mouse or by calling **MoveWindow**. The units of the limits are the same as those returned by **GetWindow** wsize.

The sizeLimit keyword was added in Igor Pro 7.00.

To allow the window width to grow essentially without bound, pass INF for maxWidth.

To allow the window height to grow essentially without bound, pass INF for maxHeight.

If *minWidth* > *maxWidth* or *minHeight* > *maxHeight*, the maximum dimensions are set to the minimum, effectively fixing the size of the window in that dimension. For control panels, it is better to use `ModifyPanel fixedSize=1`.

Combining these limits with ModifyGraph width and height modes leads to unexpected results and is discouraged.

If you first use sizeLimit and then execute `ModifyPanel fixedSize=1` on the same window, the fixedSize command takes precedence. If you execute `SetWindow sizeLimit` on a control panel that has `fixedSize=1`, Igor generates an error.

Igor includes a `SetWindow sizeLimit` command in a window recreation macro if necessary. Igor6 does not support `SetWindow sizeLimit` so this causes an error when recreating the window in Igor6. If you never set the `sizeLimit` property for a window, or if the minimum dimensions are very small and maximum dimensions are INF, then no SetWindow command is generated.

tooltipHook(*hName*)=*procName*

A tooltip hook function is like a window hook function (see **Named Window Hook Functions** on page IV-295), except that it allows you to alter the text of a tooltip in a graph or table. At present, a tooltip hook function is called for a graph when the mouse hovers over a trace or image, and for a table when the mouse hovers over the data for a wave.

Unlike a window hook function, a tooltip hook function can be set for either a top-level window or a subwindow.

For details, see **Tooltip Hook Functions** on page IV-310.

userdata=*UDStr*
userdata(*UDName*)=*UDStr*

Sets the window or subwindow user data to *UDStr*. Use the optional (*UDName*) to specify a named user data to create.

userdata+=*UDStr*
userdata(*UDName*)+=*UDStr*

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

Details

For details on named window hooks, see **Window Hook Functions** on page IV-293.

Unnamed window hook functions are supported for backward compatibility only. New code should use named window hook functions. For details on unnamed window hooks, see **Unnamed Window Hook Functions** on page IV-305.

For details on marker hooks, see **Custom Marker Hook Functions** on page IV-308.

You can also attach user data to traces in graphs using the userData keyword of the ModifyGraph operation and to controls using the userData keyword of the various control operations.