

## **Window Controls Contents**

Overview

Parent Windows

Location and Size

Child Controls

Child Classes

Richedit Control

Statusbar

Tab Control

Toolbar

Common Dialog Boxes

Fonts

Accelerator Keys

Menus

Tab and Cursor Keys

Ownerdraw

## Overview

Windows supports several types of graphic controls that can be included in the user interface. All these controls are accessed using the Window Driver.

This chapter describes each control, plus other Window Driver commands used in developing user interfaces. For an example illustrating the main types of control, run menu Studio|Demos|controls

All controls are displayed in a window called the *parent* window. The parent window is created first, and controls are then added to it. The controls in a parent window are referred to as *child* controls.

The parent window and child controls have names, referred to as their *ids*, that are set when created. These start with an alphabetic character and consist of alphanumeric characters. Ids are case-sensitive and can be up to 31 characters. For example:

```
wd 'pc abc'
```

creates a parent window with id abc.

Several parent windows may be created at a time. Only one parent window may be *selected* at a time: wd 'p sel abc' selects the parent with id abc. The wd commands affect the selected parent.

## Parent Windows

The parent window is the display area for the menus and child windows required by a user interface. The command `pc`, optionally followed by various styles, creates a new parent window:

- `pc` creates a standard parent window
- `pc owner` creates a parent window that is owned by the currently selected window. The owner is disabled until the new parent is closed.
- `pc dialog` creates a window that has a dialog box frame, that is, it cannot be resized.
- `pc closeok` creates a window that closes without an event when the user selects the window control menu item: Close.

Parent windows are initially not visible. They can be displayed in various ways using the `pshow` command, for example `pshow sw_showmaximized` shows the parent window maximized (to fit the available screen). Typically, a parent window and all its children are created first, then the `pshow` command is given.

## Location and Size

The location and size of controls in a parent is set with the *rectangle*. The `xywh` command sets all values for the rectangle.

4 integers define the rectangle:

x	units from the left
y	units from the top
w	units in width (to the right)
h	units in height (to the bottom)

These units are in terms of the average character size of a notional *system font*. The size of this system font can be obtained from the `qm` (query metrics) command. `wd 'qm'` returns the screen size in pixels, and the pixel size of the system font.

- x and w values are 1/4 of the system font width
- y and h values are 1/8 of the system font height

For example:

```
wd 'xywh 40 80 100 160'
```

sets the rectangle to be 40 units from the left edge of the parent; 80 units down from the top edge of the parent; 100 units wide; and 160 units high.

Since location and size are defined relative to the size of the notional system font, window definitions should display reasonably well on different screens at various resolutions.

## Child Controls

Child controls are created in the currently selected parent window. A parent window may have several child controls. `wd` commands affect the child controls of the selected parent window.

Child controls are created with a given class and styles. The class defines the type of control, the styles customize the control. Some styles are specific to a particular type of control, other styles can be used with most or all controls. Style names reflect their use, for example `es_uppercase` is a style applicable only to edit boxes. Styles whose names begin with the letters `ws_` apply to any control. The `group` style groups the control with the previous control. Tab and Cursor keys recognize groups.

The command `cc` creates child controls. Typical syntax is:

```
cc id class style1 style2 ...
```

## Child Classes

The class defines the appearance and behavior of the control. The classes are:

button	pushbutton
checkbox	check box
combobox	combination of edit and listbox
combodrop	dropdown combo box
combolist	combination of edit and listbox
edit	single line edit box
editm	multiline edit box
groupbox	group box
isigraph	graphics box
isipicture	picture from a BMP, WMF, or ICO file
listbox	list box
radiobutton	radio button
scrollbar	horizontal scrollbar
scrollbarv	vertical scrollbar
static	static text
staticbox	display box
progress	progress bar
richedit	single line rich edit control
richeditm	multiline rich edit control
spin	horizontal spin button
spinv	vertical spin button
tab	tab control
trackbar	horizontal trackbar
trackbarv	vertical trackbar

### ***button***

A *button* simply initiates a Windows event when pushed. Possible styles are:

- `bs_defpushbutton` the default pushbutton. Pressing Enter has the same effect as clicking this button.
- `bs_ownerdraw` a button with a picture from a graphics file

### ***checkbox***

A *checkbox* allows a two way selection - checked or unchecked. The only style is:

- `bs_lefttext` the caption is displayed on the left, instead of on the right.

Use the `set` command to set the value of a checkbox, for example, to check control `cb`:

```
wd 'set cb 1'
```

The result of `wd 'q'` contains the current value of each check box.

### ***combobox*** ***combodrop***

### ***combolist***

A *combobox* is a combination of a listbox and an edit box. You can enter a response into the edit box, or select a response from the listbox. You may select only one item in a combobox.

A *combodrop* is similar to a combobox, but initially shows only the edit box, and allows the list box to appear when selected.

A *combolist* is similar to a combobox, except that you may select only from the list box, and may not enter any response into the edit box.

Class styles are:

- `cbs_autohscroll`      allow horizontal scrolling of the edit box
- `cbs_sort`              sort entries alphabetically

Use the `set` command to set the possible selections in a checkbox. This can be a list of names, that may be delimited by LF, EAV or ". For example:

```
wd 'set cbox red green blue brown'  
wd 'set cbox "red" "green" "blue" "brown"  
wd 'set cbox ',; (::'red green blue brown') ,each LF
```

Use `setselect` to set the selection. For example, the following sets item 2:

```
wd 'setselect cbox 2'
```

The result of `wd 'q'` contains the current value and selection of each combobox. For example:

```
...  
+-----+-----+  
| cbox            | blue            |  
+-----+-----+  
| cbox_select    | 2               |  
+-----+-----+
```

### ***edit***

#### ***editm***

An *edit* control is a single-line box, and an *editm* control is a multiline box, in which text can be displayed and edited. Edit control styles are:

- `es_autohscroll`      use horizontal scroll bars if required
- `es_autovscroll`     use vertical scroll bars if required
- `es_center`            center text
- `es_lowercase`        lowercase text only
- `es_readonly`         display text as read-only
- `es_right`             right justify text in control
- `es_uppercase`        uppercase text only

Use the `set` command to write text to an edit control. For example:

```
wd 'set edit *Jeremy Fisher'
```

The result of `wd 'q'` contains the current text for each edit box, and indices of any selected text:

```
...
+-----+-----+
|edit      |Peter Rabbit |
+-----+-----+
|edit_select|6 12         |
+-----+-----+
```

### ***groupbox***

A groupbox control is a box used to group controls, most often radiobuttons. There are no class styles.

### ***isigraph***

An isigraph control is a window that can display graphics. First create an isigraph control, then use graphics commands to create and display the graphics. All graphics commands start with `g`. Graphics are displayed only when the `gshow` command is given.

Graphics commands apply to the currently selected isigraph control. The graphics window has coordinates running from (0,0) in the bottom left hand corner to (1000,1000) at the top right.

See the demos, Studio\Demos\isigraph.

### ***isipicture***

An isipicture control displays an image from a file. The file may be a DIB (device independent bitmap), WMF (Windows metafile) or ICO (icon file). DIB files usually have a file extension of `.bmp`.

The `set` command sets the filename. For example, to display the J logo icon:

```
wd 'set isipicture *system\examples\data\j.ico'
```

### ***listbox***

A listbox control displays a list of items that can be selected by the user. Listbox control styles are:

- `lbs_extendsel` extended select holding down the Shift key
- `lbs_multicolumn` display in multiple columns
- `lbs_multipleselect` allow multiple selections
- `lbs_ownerdrawfixed` create an ownerdrawn listbox
- `lbs_sort` sort entries alphabetically

Use the `set` command to put items in the listbox (as in combobox above).

In the result of `wd 'q'`, multiple selections are delimited by `LF`. For example:

```
+-----+-----+
|listbox    |green blue brown |
+-----+-----+
|listbox_select|1 2 3           |
+-----+-----+
```

### ***progress***

A progress control displays a bar used to indicate the status of a program. Use the `set` command to set values between 0 and 100. For example, to indicate the half way stage:

```
wd 'set pg 50'
```

### ***radiobutton***

radiobutton controls typically allow a single selection from a group of controls. Selecting one control switches the others off. The only style is:

- `bs_lefttext`                    the caption is displayed on the left, instead of on the right.

Use the `set` command to set the value of a radiobutton, for example, to check control `rb`:

```
wd 'set rb 1'
```

The result of `wd 'q'` contains the current value of each radiobutton.

### ***richedit***

#### ***richeditm***

A *richedit* control is a single line edit box, and a *richeditm* control is a multiline edit box, each containing rich edit text. Class styles are:

- `es_automscroll`                    use horizontal scroll bars if required
- `es_automscroll`                    use vertical scroll bars if required
- `es_center`                            center text
- `es_readonly`                        display text as read-only
- `es_right`                            right justify text in control
- `es_sunken`                            display sunken

For more information, see the Richedit Control section later in this chapter.

### ***scrollbar***

#### ***scrollbarv***

A *scrollbar* control displays a horizontal scrollbar, and a *scrollbarv* displays a vertical scrollbar. There are no class styles.

The `set` command sets four values for the scrollbar: leftmost (or top) position, current position, rightmost (or bottom) position, and the size of change resulting from a click in the area between the thumbbox and either end of the scrollbar.

```
wd 'set sb 0 500 1000 50'
```

A single parameter sets the current position:

```
wd 'set sb 600'
```

Clicking a scrollbar signals an event. The result of `wd 'q'` has the current position of the scrollbar. For example:

```

...
+-----+-----+
|sb      |550      |
+-----+-----+

```

***spin***  
***spinv***

A *spin* control displays two arrows horizontally, and a *spinv* control displays the arrows vertically. Clicking an arrow initiates a Windows event, and the result of `wd 'q'` is `_1` for the down or left arrow, `1` for the up or right arrow.

***static***

A static control is used to display text. It is never active. Static control styles are:

- `ss_center`                      center text
- `ss_leftnowordwrap`            left justify, no word wrap
- `ss_noprefix`                    allow & characters in control
- `ss_right`                        right justify text
- `ss_simple`                        static text control

***staticbox***

A static control is used to display boxes with various types of frames and backgrounds. It is never active. Staticbox control styles are:

- `ss_blackframe`                black frame
- `ss_blackrect`                 black rectangle
- `ss_etchedframe`                etched frame
- `ss_etchedhorz`                etched horizontally only
- `ss_etchedvert`                etched vertically only
- `ss_grayframe`                 gray frame
- `ss_grayrect`                 gray rectangle
- `ss_sunken`                     sunken
- `ss_whiteframe`                white frame
- `ss_whiterect`                 white rectangle

***tab***

A tab control is used to display several Windows controls on tab forms. Class styles are:

- `tcs_button`                    display tabs as buttons
- `tcs_multiline`                allow tabs to be displayed in several lines

For more information, see the Tab Control section later in this chapter.

***trackbar***  
***trackbarv***

A *trackbar* control displays a horizontal trackbar, and a *trackbarv* displays a vertical trackbar. Class styles are:

- `tbs_autoticks` display tick marks along control
- `tbs_both` display tick marks on both sides of control
- `tbs_enablese xrange` enables `setselect` to mark a range on the trackbar
- `tbs_left` display thumb pointing to left / tick marks on left
- `tbs_nothumb` do not display a thumb control
- `tbs_noticks` do not display any tick marks
- `tbs_top` display thumb pointing to top / tick marks on top

The `set` command sets up to five values for the trackbar: leftmost (or top) position, current position, rightmost (or bottom) position, and the size of change resulting from a click in the area between the thumbbox and either end of the trackbar, and line size.

```
wd 'set tb 0 3 20 1 1'
```

A single parameter sets the current position:

```
wd 'set tb 4'
```

Clicking a trackbar signals an event. The result of `wd 'q'` has the current position of the trackbar. For example:

```
...
+-----+-----+
|tb           |4           |
+-----+-----+
```

## Richedit Control

A richedit control is an edit control that displays RTF (rich text format) data.

RTF is a text description language that uses only standard ASCII characters, so that you can create and view RTF data as ordinary text. For a summary of the language, see file: `system\examples\data\rtf.txt`.

You can also create RTF data from most Windows editors, for example, WordPad, by saving your text in RTF format. Thus you could create and format some text in WordPad, save it in RTF format, then read in the RTF file and display it in J using a richedit control.

You can also copy and paste between a richedit control and any application, such as WordPad, that supports RTF.

Events and event data for richedit controls are the same as for edit controls. The event data is the simple text from the control, not the rtf data. To read the text in rtf format, use command `qrtf`.

Here is a summary of commands applicable to richedit controls:

`set`                    set rtfdata into a richedit control. For example:

```
wd 'set rid *',rtfdata
```

`setreplace`            replace selected rtfdata in a richedit control For example:

```
wd 'setreplace rid *',rtfdata
```

`setbkgnd`              set background color. For example:

```
wd 'setbkgnd rid 255 0 0'
```

A useful color is the gray window background, which is typically 192 192 192. A richedit control with this background color and without the sunken style appears as text on the form.

`setreadonly`          prevent the user from making changes. For example:

```
wd 'setreadonly rid'
```

`qrtf`                    reads the RTF data from the control. For example:

```
rtfdata=. wd 'qrtf rid'
```

Here is a brief overview of the RTF format:

The `\` character starts an RTF command and curly braces `{ }` group data.

Some RTF commands:

```
\b                    bold
```

<code>\cfn</code>	color from color table
<code>\fn</code>	font from font table
<code>\fsn</code>	font size
<code>\i</code>	italic
<code>\par</code>	paragraph (new line)
<code>\ul</code>	underline

Commands like `\b` can be followed by a 0 or 1 to turn the attribute off or on.

The font table definition is enclosed in `{ }` and each font definition is also enclosed in `{ }`. For example:

```
{\fonttbl{\f0\fcourier Courier New;}}
```

The `\fn` command indicates which font to use. For example:

```
\f0 test
```

The `\fsn` command indicates font size. For example:

```
\fs90 test
```

To replace the current selection with "test" in Courier New size 90:

```
wd 'setreplace rid *{\fonttbl{\f0\fcourier Courier New;}}\f0\ fs90 test}'
```

Colors are managed in a manner similar to fonts. Define a color table and then select colors from that table. The following table defines colors black and red:

```
{\colortbl\red0\green0\blue0;\red255\green0\blue0;}
```

To replace the current selection with 'test' in red:

```
wd 'setreplace rid *{\colortbl\red0\green0\blue0;\red255\green0\blue0;}\cf1 test}'
```

The `/` character is an escape character that treats the following character as text. For example to enter a `}` as part of text, rather than treat it as a grouping character:

```
wd 'setreplace red *{the character /{ appears}{'
```

The following example (in script examples\demo\rtf.ifs) creates some RTF data, then displays it in a form with a richeditm control:

```
rtfdata=: 0 : 0
{
{\fonttbl{\f0\fcourier Courier New;}}
{\colortbl\red0\green0\blue0;\red255\green0\blue0;}
\f0 black
\par
\b
\cf1 bold red
```

```
\par
\i
\fs60 big bold italic red
}
)
```

```
wd 0 : 0
pc abc closeok;
xywh 148 8 34 12;cc ok button;cn "OK";
xywh 148 23 34 12;cc cancel button;cn "Cancel";
xywh 9 7 128 69;cc rid richeditm es_autovscroll es_sunken;
pas 6 6;pcenter;
rem form end;
)
wd 'set rid *',rtfdata
wd 'pshow'
```

## Statusbar

A statusbar can be shown at the foot of a form.

There are 3 statusbar commands:

<code>sbar n;</code>	n is number of panes in the statusbar
<code>sbarshow b;</code>	b is 0 to hide, 1 to show (default)
<code>sbarset id width text;</code>	a width of -1 leaves at current value

The first pane is special. Its width is variable and takes up what is left. It does not have a border. It can be used to display help text for menu and toolbar items.

## Tab Control

A tab control is a panel used to display other controls.

Commands applicable to the tab control are:

<code>set tab text;</code>	add a tab label to the control
<code>setreplace tab n text;</code>	replace the text of a tab label
<code>setinsert tab n text;</code>	insert a tab label
<code>setdelete tab n;</code>	delete a tab label

There are two uses for tab controls:

- You can set up a single set of controls, and use the tab control to switch between different values for the controls. In this case, the controls in the tab area are created in the same form as the tab itself.
- You can set up several forms, and use the tab control to switch between forms. In this case, the forms should be created separately from the main form.

As an example of the first use, you could create controls that contain values for days of the week. Selecting the tab for the day changes the values of the controls accordingly.

For example, the following demo shows an edit control that depends on the day of week:

```
load 'system\examples\demo\days.ijs'
```

An example of the second type of tab control is the controls demo that displays the main Windows controls supported by J, see menu Studio|Demos|Controls.

In this example, the controls displayed on each tab were created in separate script files: `system\examples\demo\control1.ijs`, `control2.ijs` etc.

This second type of tab control uses the `creategroup` command to allow several forms to be displayed in a single form.

A form definition used with a `creategroup` command is an ordinary form that can be designed and tested with the form editor. It is loaded by the parent of the tab control by doing a `form_run''` that is bracketed by `creategroup` commands. The first `creategroup` command gives the id of the tab control where the new controls are being created. The final `creategroup` command has no parameter.

`creategroup` causes parent commands such as `pc` to be ignored so that when the form definition is run, the child creates occur in the original parent form. The initial argument to `creategroup` is an id, usually of a tab control, in the current form. Controls created under a `creategroup` command are created as hidden, and as part of a group with the id from their (ignored) `pc` command. The `setshow` command with a group id, shows or hides the controls in a group.

For example, here is the relevant code from the controls demo:

```
wd 'creategroup tabs'  
buttons_run''  
edits_run''  
selects_run''  
wd 'creategroup'
```

This code:

- uses `creategroup` to prepare to load forms for the `tabs` control
- loads each form to be shown on the `tabs` control
- uses `creategroup` with no parameter to finish up

Note that a tab control should be created before any controls that appear on top of it - this ensures controls will be painted properly.

The event data for a tab control is the label text and the `id_select` variable contains the index of the selected tab.

## Toolbar

A toolbar can be shown at the top of a form, beneath any menu.

There are 3 toolbar commands:

<code>tbar filename;</code>	filename of toolbar bitmap
<code>tbarshow b;</code>	b is 0 to hide, 1 to show (default)
<code>tbarset id index image;</code>	toolbar index and image number in bitmap

The `tbarset` command with an empty id sets the toolbar index as a separator with a width as specified in the image value.

Toolbar buttons are usually for a command that is also on the menu. If the menu item and the toolbar button are given the same id, then `set` and `setenable` commands affect both, and they will cause the same event.

The `set` command for an id that is a menu or toolbar command will check or uncheck the menu or toolbar.

The `setenable` command for an id that is a menu or toolbar command enables or disable the commands.

Any toolbar bitmap file can be used. An example is provided in file: `system\examples\data\isitbar.bmp`, which is referenced by the controls demo.

You can create and edit toolbars using Paint. To do so, open the `.bmp` file using Paint, maximize the zoom setting and select grid on.

## Common Dialog Boxes

Windows provides several built-in dialog boxes called *Common Dialog Boxes* that perform useful functions.

The *color* dialog box allows selection of colors. The result is a list of the RGB values for 17 colors, of which the first is chosen in the standard color dialog box, and the rest in the custom color dialog box. Try:

```
_3[\ ". wd 'mbcolor'
```

This returns a matrix of the 17 values, one row per color.

The *font* dialog box allows selection of a logical font. The result describes the font chosen (in this case the font used in the J logo):

```
wd 'mbfont'  
"Bookman Old Style" 24
```

The *message box* dialog box displays a message and waits for a user response. For example:

```
wd 'mb title text'  
wd 'mb title text mb_iconstop mb_retrycancel'
```

To list all message box styles:

```
list wd 'qs mb'  
mb_abortretryignore mb_defbutton2      mb_defbutton3  
mb_iconasterisk      mb_iconexclamation  mb_iconhand  
mb_iconinformation  mb_iconquestion   mb_iconstop  
mb_ok                mb_okcancel       mb_retrycancel  
mb_yesno             mb_yesnocancel
```

The *open filename* dialog box allows the user to select a fully qualified file name:

```
wd 'mbopen'  
wd 'mbopen title "" "" Write(*.wri)|*.wri| Word(*.doc) |*.doc"  
ofn_filemustexist'
```

To list all open filename styles:

```
list wd 'qs mbopen'  
ofn_createprompt    ofn_filemustexist...  
ofn_overwriteprompt ofn_pathmustexist
```

The *save filename* dialog box is similar to the open filename dialog box:

```
wd 'mbsave'
```

## Fonts

A font can be chosen with the `mbfont` command. The result is a description of a font, that can be used as the argument to other commands.

A maximum of 20 fonts can be selected for use in controls. Deleting a control does not release the font resource. A `wd 'reset'` command frees all font resources.

Use the `setfont` command to set the font. For example, the following sets the font for control `bn` to: Lucida Console, 24 point, italic, bold:

```
wd 'setfont bn "Lucida Console" 24 italic bold'
```

## **Accelerator Keys**

& in the name of a button or menu item sets a keyboard accelerator. The & is not displayed and the next character is underlined. Pressing ALT + the character is the same as clicking on the button or menu item.

## Menus

Several Window Driver commands support menus:

menu id text	add menu item
menupop text	add popup menu item
menupopz	ends popup menu and drops down a level
menusep	add separator line in a popup menu

To add a popup menu item pop1, displayed in the menu bar:

```
wd 'menupop pop1'
```

To add individual items to the popup menu, displayed when the menu is selected:

```
wd 'menu item1 "item name"'
```

This displays the text “item name”, and if selected, the Windows result contains the name item1.

To add a separator line:

```
wd 'menusep'
```

To end the popup menu:

```
wd 'menupopz'
```

To check a menu item:

```
wd 'set item 1'          NB. 1=check, 0=uncheck
```

To enable a menu item:

```
wd 'setenable item 1'   NB. 1=enable, 1=disable
```

## **Tab and Cursor Keys**

TAB and SHIFT+TAB keys cycle the focus through the children in the order they were created.

Cursor keys cycle through the controls in a *group*. By default, controls, except for radiobuttons, are created as part of a group consisting only of themselves.

## Ownerdraw

A button created with the *bs\_ownerdraw* style is an ownerdraw button. A listbox created with *lbs\_ownerdrawfixed* style is an ownerdraw listbox.

A control with ownerdraw style displays a picture from a file. The files supported are the same types supported for the pictures in isipicture class windows.

The file names to display are set with the `set` command.

For example, the following will display an ownerdraw button with the J icon:

```
wd 'pc abc; xywh 10 10 30 30'  
wd 'cc b1 button bs_ownerdraw'  
wd 'cn "system\examples\data\jb.ico"'  
wd 'pas 10 10; pshow'
```