STYLE(STYLE EX)

Applies to

All controls.

Description

Sets or retrieves the Windows SDK style (long integer) for a control.

Usage

style = **Get_Property** (*controlID*, **"STYLE"**)

oldstyle = **Set_Property** (*controlID*, **"STYLE"**, *newstyle*)

Remarks

The styles for all standard Windows controls are defined in the WINDOWS.H file that comes with the Windows SDK or any Windows C/C++ compiler. The following styles apply to the edit table custom control:

Value	Description
4	Editable (allow row insertion and deletion)
8	Allow Resizing
16	Numbers - Column heading are 1, 2, 3, ...
32	Letters - Column headings are A, B, C, ...
48	OwnerDef - Column headings are set by the user
64	Display Horizontal Grid
128	Display Vertical Grid
256	Permit Row Selection
512	Multi-Row Selection
1024	Permit Column Selection
4096	Allow >64k of data (but disallow column addition or deletion using Send_Message)
8192	Display Row Buttons
16384	Display Row Numbers

See also

PSStyle property, COLSTYLE message, Bitwise operators, IConv(expression, "MX"), OConv(expression, "MX"), RTI_Style_Equates

Microsoft Windows SDK

Examples

Window and control styles are designed as bitmasks. Each style typically uses one bit of the STYLE property, and is either set (1) or not set (0). To set a style, you bitor the current style with the style that you want to add. For example, the following subroutine could be used to set a specific style for a specific control:

```
subroutine SetStyle(CtrlEntID, AddStyle)

declare function   Get_Property
declare subroutine Set_Property

* get the current style

Style = Get_Property(CtrlEntID, "STYLE")

* the style property can be in hex format but bitor only works with decimal integers
if Style [1,2] _eqc "0x" then
convert @lower.case to @upper.case in Style
Style = iconv(Style [3,99], "MX")

end

* add the new style
Style = bitor(Style, AddStyle)
Set_Property(CtrlEntID, "STYLE", Style)

return
```

To clear a style, you bitand the current style with all styles but the style that you are removing. The following subroutine could be used to clear a specific style for a specific control:

```
subroutine ClearStyle(CtrlEntID, RemoveStyle)

declare function   Get_Property
declare subroutine Set_Property

* get the current style
Style = Get_Property(CtrlEntID, "STYLE")


* the style property can be in hex format but bitor only works with decimal integers
if Style [1,2] _eqc "0x" then
convert @lower.case to @upper.case in Style
Style = iconv(Style [3,99], "MX")

end

* remove the specified style
Style = bitand(Style, bitnot(RemoveStyle))
Set_Property(CtrlEntID, "STYLE", Style)

return
```

To test if a style is set, use the bitand function on the current style with the style that you are looking for. The following function could be used to test if a style is set for a specific control:

```
function IsStyleSet(CtrlEntID, TestStyle)

declare function   Get_Property
declare subroutine Set_Property

* Get the current style
Style = Get_Property(CtrlEntID, "STYLE")

* the style property can be in hex format but bitor only works with decimal integers.
if Style [1,2] _eqc "0x" then

  convert @lower.case to @upper.case in Style

  Style = iconv(Style [3,99], "MX")

end

* check for the specified style
bExists = (bitand(Style, TestStyle) > 0)

return bExists
```

Some styles cannot be set or cleared using the STYLE property. This is due to how the controls are implemented internally. Because of this, save your work before testing style changes. For styles that can not be set or cleared, you must destroy the control and re-create it with the desired style. Although this sounds difficult, the following example can be modified and used for almost all cases. Also the TABBED_TEMPLATE form in the Examples application uses a similar method to change bitmap check-boxes into bitmap radio buttons in the CREATE event.

```
/* this code snippet changes a bitmap control into a control that displays system icons, as displayed in
Windows messages;

this code is from the message designer in the UI Workspace(PSPOS_BITMAP$ can be INFO, QUESTION, EXCLAMATION,
and HAND) */

$insert PS_Equates
Struct = Get_Property(CtrlEntID, "ORIG_STRUCT")
Struct<1, PSPOS_SDKSTYLE$> = "0x50000003"
Struct<1, PSPOS_BITMAP$  > = "INFO"
Struct<1, PSPOS_VISIBLE$ > = TRUE$
Struct<1, PSPOS_PSSTYLE$ > = ""
Utility("DESTROY", CtrlEntID)
Utility("CREATE", Struct)

* Here is another example which changes the justification of an edit control:

* This code snippet assumes that the variable Just is set to L, R, or C (left, right or center)

$insert PS_Equates
equ ES_LEFT     to 0
equ ES_CENTER   to 1
equ ES_RIGHT    to 2

/* since the control is going to be destroyed then recreated,
get the current text value so it isn't lost */

Text   = Get_Property(CtrlEntID, "TEXT")
Style  = Get_Property(CtrlEntID, "STYLE")
Struct = Get_Property(CtrlEntID, "ORIG_STRUCT")
if Style [1,2] _eqc "0x" then

convert @lower.case to @upper.case in Style
Style = iconv(Style [3,99], "MX")

end

* first turn off left, right, and center justification
All   = bitor(ES_LEFT, bitor(ES_CENTER, ES_RIGHT))
Style = bitand(Style, bitnot(All))


* next, turn on specific justification style

begin case
case Just = "L"
AddStyle = ES_LEFT
case Just = "C"
AddStyle = ES_CENTER
case Just = "R"
AddStyle = ES_RIGHT
end case

Style = bitor(Style, AddStyle)
* next, destroy and recreate the control with the new just
Struct<1, PSPOS_SDKSTYLE$> = Style
Struct<1, PSPOS_TEXT$    > = Text
Utility("DESTROY", CtrlEntID)
Utility("CREATE", Struct)
```