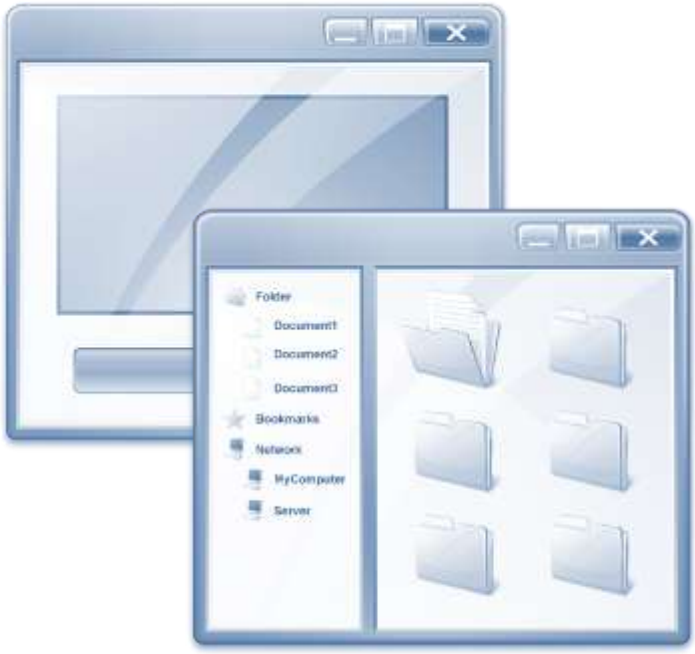
Common GUI Object API

These core properties, methods and events apply to *most* GUI objects (forms and controls) except where noted in individual descriptions later.



Common GUI Properties

These core properties apply to most GUI forms and controls except where noted in individual descriptions later.

Name	Description
ACCEPTDROPFILES	Specifies if an object can accept files dragged from the Windows Explorer.
ALLOWCONTAINER	Returns TRUE$ if an object is allowed to contain other objects when using the Form Designer.
ALLPAGES	Specifies if an object appears on all pages of a multi-page Container parent.
AUTOSIZEHEIGHT	Specifies if a control changes its height relative to the height of the parent object.
AUTOSIZEWIDTH	Specifies if a control changes its width relative to the width of the parent object.
BACKCOLOR	Specifies the color(s) to use when painting an object's background.
BOTTOM	Specifies the bottom coordinate of an object relative to its parent object.
BOTTOMANCHOR	Specifies if a control is anchored to the bottom of its parent.
CANGETFOCUS	Indicates if an object can receive the input focus.
CLASSNAME	Returns the registered window class name for an object.
CHILDOBJECT	Returns TRUE$ if an object is a "child window".
CLIENTHEIGHT	Specifies height of an object's client area.
CLIENTSIZE	Specifies the width and height of an object's client area.
CLIENTWIDTH	Specifies width of an object's client area.
CLIPCHILDREN	Specifies if the object excludes child objects from its update region before painting itself.
CLIPSIBLINGS	Specifies if the object excludes sibling objects from its update region before painting itself.
COLUMN	Specifies the database column that a control is bound to.
COMPOSITED	Specifies if system double-buffering is used by an object.
CONTEXTMENU	Specifies the CONTEXTMENU entity used by an object.
CONV	Specifies the Output Conversion pattern for a control.
CURSOR	Specifies the cursor to use when over the control.
CUSTOMPROPERTIES	Specifies a list of user-defined properties to set for the object at runtime.
DEFPROP	Specifies the value of the "default property" for a control.
DEFPROPID	Returns the name of a control's "default property".
DEFPROPPOS	Specifies the value of a control's "default position property".
DEFPOSPROPID	Returns the name of a control's "default position property".
DEFPROPRAW	Specifies the value of the "default property" for a control but without updating the RECORD/SAVEWARN properties.
DEFVALUE	Specifies the default content to insert into a control when it is empty and receives the input focus.
DESIGNMODE	Returns TRUE$ if a control is in "design mode".

DESIGNSELECTED	Returns TRUE$ if a control is selected in the Form Designer.
DOUBLEBUFFER	Specifies if an object is using double-buffering for drawing.
DPI	Returns the current DPI settings for an object.
ECHO	Allows keyboard input to be turned off for a control.
EDGESTYLE	Specifies the appearance of a control's non-client area border.
ENABLED	Enables or disables mouse and keyboard input for a control.
FOCUS	Specifies if the control has the system input focus.
FONT	Specifies the font used when rendering text in a control.
FORECOLOR	Specifies the foreground text color for a control.
GOTFOCUSVALUE	Returns the value in a control from when it last received the input focus.
HANDLE	Returns the window handle (HWND) of an object.
HEIGHT	Specifies the height of an object.
IMAGE	Specifies the properties of an object's IMAGE sub-object at *design-time*.
IMAGELIST	Specifies the properties of an object's IMAGELIST sub-object at *design-time*.
INVALUE	Specifies the value of the "default property" for a control in internal format.
LEFT	Specifies the left-coordinate of an object relative to its parent object.
MONITOR	Returns the details of the monitor that an object is displayed on.
MOUSECAPTURED	Specifies if mouse messages are redirected to a specific object.
NEXT	Returns or updates the next control in tab order from the specified control.
ORIGARRAY	Returns the original "List" attribute from the structure used to create the object at runtime, but in "Array" format.
ORIGBACKCOLOR	Returns the original "BackColor" attribute from the structure used to create the object at runtime.
ORIGENABLED	Returns the original "Enabled" attribute from the structure used to create the object at runtime.
ORIGFONT	Returns the original "Font" attribute from the structure used to create the object at runtime.
ORIGFORECOLOR	Returns the original "ForeColor" value from the structure used to create the object at runtime.
ORIGHEIGHT	Returns the original "Height" attribute from the structure used to create the object at runtime.
ORIGHIGH	Synonym for the ORIGHEIGHT property.
ORIGLABEL	Returns the original label value from the structure used to create the object at runtime.
ORIGLEFT	Returns the original Left coordinate used when the object was created at runtime.
ORIGLIST	Returns the original "List" attribute from the structure used to create the object at runtime.

ORIGROWVALUE	Returns the original data as read into a form or control during a form's READ event.
ORIGSIZE	Returns the original size attributes from the structure used to create the object at runtime.
ORIGSTRUCT	Returns the original structure used to create an object.
ORIGTEXT	Returns the original text used when the object was created at runtime.
ORIGTOP	Returns the original TOP coordinate used when the object was created at runtime.
ORIGVALUE	Returns the original value used when the object was created at runtime.
ORIGVISIBLE	Returns the original "Visible" attribute from the structure used to create the object at runtime.
ORIGWIDE	Synonym for the ORIGWIDTH property.
ORIGWIDTH	Returns the original "Width" attribute from the structure used to create the object at runtime.
ORIGX	Synonym for the ORIGLEFT property.
ORIGY	Synonym for the ORIGTOP property.
PAGENUMBER	Specifies the page number that an object should appear on in a multi-page parent object.
PARENT	Returns the parent for a specified object.
PARENTFORM	Returns the parent form object for a specified object.
PART	Returns the key part of a control that is bound to a database key column.
POS	Returns the position of the column, relative to the data table structure, of a data-bound control.
PREVIOUS	Returns or updates the previous control in tab order from the specified control.
QUALIFIEDWINMSGS	Returns a list of Window Messages that trigger the WINMSG event for an object.
RECT	Determines the position of a control relative to its parent control in client area coordinates.
REDRAW	Specifies if an object is allowed to update itself on screen.
REQUIRED	Specifies if a control must contain data.
RIGHT	Specifies the right coordinate of an object relative to its parent object.
RIGHTANCHOR	Specifies if a control is anchored to the right side of its parent.
SCALEFACTOR	Returns the scale-factor value for an object.
SCALEMETRICS	Returns an array of scaling information for the specified object.
SCALEUNITS	Returns the scale-units value for an object.
SCREENRECT	Returns the position of an object in screen coordinates.
SCREENSIZE	Returns the position and size of an object in screen coordinates.
SCROLLBARS	Specifies which scrollbars are used with an object,
SIZE	Determines the position and size of an object relative to its parent object in client area coordinates.
STYLE	Specifies the Windows Style flags for a control.

STYLEN	Specifies the Windows Style flags for a control in numeric format.
STYLEEX	Specifies the Extended Windows style flags for a control.
STYLEEXN	Specifies the Extended Windows style flags for a control in numeric format.
TABLE	Returns the database table that an object is bound to.
TEXT	Specifies the text associated with the object.
TIMER	Specifies the interval for firing an object's TIMER event.
TOOLTIP	Specifies the tooltip to display for an object.
TOP	Specifies the top coordinate of an object relative to its parent object.
TRANSLUCENCY	Specifies the degree of transparency applied to an object's background.
VALID	Specifies the Validation and Input conversion pattern for a control.
VALIDMSG	Specifies alternative text for a validation error message.
VISIBLE	Determines if a control is visible.
WIDTH	Specifies the width of an object.

ACCEPTDROPFILES property

Description

Specifies if the object can accept files dragged from the Windows Explorer.

Property Value

This property is a boolean value. If the object can accept files dragged from the Windows Explorer this property returns TRUE$, otherwise it returns FALSE$.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

When set to TRUE$, and files are dragged and dropped onto the object, a DROPFILES event is triggered.

For more information on this property please refer to the Windows documentation regarding the WS_EX_ACCEPTFILES extended window style and the WM_DROPFILES message on the Microsoft website.

Example

```
// Allow the LST_FILELIST ListBox control to accept files dropped from
// the Windows Explorer

Call Set_Property_Only( @Window : ".LST_FILELIST", "ACCEPTDROPFILES", TRUE$ )
```

See also

Common GUI STYLEEX property, Common GUI STYLEEXN property, Common GUI DROPFILES event.

ALLOWCONTAINER property

Description

Specifies if an object is allowed to contain another object in a parent-child relationship.

Property Value

This property is a boolean value. If TRUE$ then the Form Designer will allow this object to contain other objects when creating or modifying a form.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

Only forms and panel-based controls like PANEL, SIMPLEPANEL and TABBEDPANEL will return TRUE$ for this property (It is intended more for use with the Form Designer rather than as a "normal" developer property).

Example

```
// Check to see if the current control is a container

IsContainer = Get_Property( CtrlEntID, "ALLOWCONTAINER" )
```

See also

Common GUI CHILDWINDOW property, Common GUI PARENT property.

ALLPAGES property

Description

Specifies if a control appears on all pages of a multi-page Container parent, such as a Form or Panel.

Property Value

This property is a boolean value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	N/a	No	No	No

Remarks

This property is design-time only. Use the PAGENUMBER property at runtime to change the page that a control appears on.

Example

N/a.

See also

Common GUI PAGENUMBER property, Container CURRENTPAGE property, Container PAGECOUNT property, Container PAGECHANGED event, WINDOW PAGE event.

AUTOSIZEHEIGHT property

Description

Specifies if a control changes its HEIGHT property relative to the height of its parent object (Form or Panel).

For example, if the parent Form's height is increased by 40 pixels, a control with AUTOSIZEHEIGHT set to TRUE$ will also increase its HEIGHT property by 40 pixels.

Property Value

This property is a boolean value. When Set to TRUE$ changing the control's parent height also applies the same change to the control. When set to FALSE$ the control's height is not changed when the parent height is changed.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

If this property is set to TRUE$ the BOTTOMANCHOR property is automatically set to FALSE$.

The AUTOSIZEHEIGHT property is preserved and updated as necessary if the object is moved or resized.

Example

```
// Set the AUTOSIZEHEIGHT of the EDB_NOTES EditBox control to autosize it's
// Height

Call Set_Property_Only( @Window : ".EDB_NOTES", "AUTOSIZEHEIGHT", TRUE$ )
```

See also

Common GUI AUTOSIZEWIDTH property, Common GUI BOTTOMANCHOR property, Common GUI CLIENTHEIGHT property, Common GUI CLIENTSIZE property, Common GUI HEIGHT property, Common GUI RECT property, Common GUI RIGHTANCHOR property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, Common GUI MOVE method, Common GUI OFFSET method, WINDOW SIZE event.

AUTOSIZEWIDTH property

Description

Specifies if a control changes its WIDTH property relative to the height of its parent object (Form or Panel).

For example, if the parent Form's width is decreased by 50 pixels, a control with AUTOSIZEWIDTH set to TRUE$ will also decrease its WIDTH property by 50 pixels.

Property Value

This property is a boolean value. When Set to TRUE$ changing the control's parent width also applies the same change to the control. When set to FALSE$ the control's width is not changed when the parent width is changed.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

If this property is set to TRUE$ the RIGHTANCHOR property is automatically set to FALSE$.

The AUTOSIZEWIDTH property is preserved and updated as necessary if the object is moved or resized.

Example

```
// Set the AUTOSIZEWIDTH of the EDB_NOTES EditBox control to autosize it's
// Width

Call Set_Property_Only( @Window : ".EDB_NOTES", "AUTOSIZEWIDTH", TRUE$ )
```

See also

Common GUI AUTOSIZEHEIGHT property, Common GUI BOTTOMANCHOR property, Common GUI CLIENTSIZE property, Common GUI CLIENTWIDTH property, Common GUI RECT property, Common GUI RIGHTANCHOR property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, Common GUI WIDTH property, Common GUI MOVE method, Common GUI OFFSET method, WINDOW SIZE event.

BACKCOLOR property

Description

Specifies the RGB color value used for an object's background. This can be a single "solid" color, or a pair of colors that are used to paint a gradient effect instead.

An RBG color is an integer value that is constructed from a combination of Red, Green and Blue color intensities using the following formula:

RGB = (Red) + (Green * 256) + (Blue * 65536)

Each of the color intensities should be a value between 0 and 255.

In addition to this there are also a set of "special" values that are not valid RBG colors, but represent default system colors that can be used:

```
         -1   : Transparent
         -2   : Use default color for the object
              :
2147483648    : Use a Windows system colors
    to         : (See the COLORS Insert record
2483027968    : for more details)
```

Property Value

This property can be a single RGB value for a solid background color, or an @Fm-delimited array to specify a gradient background effect like so:

```
<1> Color From (RBG value)
<2> Color To (RBG value)
<3> Gradient direction (Horizontal or Vertical)
```

If only "Color From" is specified the BACKCOLOR property is assumed to be a solid color background.

The Gradient direction value can be one of the following:

```
0 (or null) – No gradient
1 – Vertical Gradient
2 – Horizontal Gradient
```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

Most, but not all, objects support gradient backgrounds. Individual object documentation will point out any that differ from this.

Be aware that the use of gradient background does involve a little extra overhead when drawing controls because of the calculations needed to blend the colors – while the PS attempts to mitigate this using cached bitmaps and double-buffering there will always be *some* impact.

Equated constants for use with the BACKCOLOR property can be found in the PS_EQUATES insert record. Common RGB color values can be found in the COLORS insert record.

Example

```
$Insert PS_Equates
$Insert Colors

// Set the BACKCOLOR of the form to Red

prevColor = Set_Property( @Window, "BACKCOLOR", RED$ )

// Set the BACKCOLOR of the form to a vertical gradient

bkColor = ""
bkColor<PS_BKCOLOR_FROM$>      = ORANGE$
bkColor<PS_BKCOLOR_TO$>        = RED$
bkColor<PS_BKCOLOR_GRADSTYLE$> = PS_GRADSTYLE_VERT$

Call Set_Property_Only( @Window, "BACKCOLOR", bkColor )

// Set the BACKCOLOR property of the LBL_NAME control to Transparent

Call Set_Property_Only( @Window : ".LBL_NAME", "BACKCOLOR", CLR_TRANSPARENT$)

// Set the BACKCOLOR property of the EDL_SURNAME control to the default
// color

Call Set_Property_Only( @Window : ".LBL_NAME", "BACKCOLOR", CLR_USEDEFAULT$ )
```

See also

Common GUI FORECOLOR property, SYSTEM CHOOSECOLOR method, RBG stored procedure.

BOTTOM property

Description

Specifies the bottom coordinate of an object relative to its parent object.

Property Value

This property is an integer value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	Yes	No

Remarks

Setting this property adjusts the height of the object, it does not affect the TOP property.

Coordinates are always relative to the top left corner of the screen, or, for a child window, the upper left corner of the parent window's client area.

Example

```
// Set the BOTTOM of the EDB_NOTES EditBox control to position 240

Call Set_Property_Only( @Window : ".EDB_NOTES", "BOTTOM", 240 )
```

See also

Common GUI AUTOSIZEHEIGHT property, Common GUI BOTTOMANCHOR property, Common GUI CLIENTHEIGHT property, Common GUI CLIENTSIZE property, Common GUI HEIGHT property, Common GUI RECT property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, WINDOW SCALEUNITS property, WINDOW SIZE event, Appendix K – High-DPI Programming.

BOTTOMANCHOR property

Description

Specifies if the bottom coordinate of a control maintains the same distance from the bottom of its parent object (Form or Panel) when the latter is resized.

For example, if the parent Form's height is increased by 40 pixels, a control with BOTTOMANCHOR set to TRUE$ will be moved down by 40 pixels as well.

Property Value

This property is a boolean value. When Set to TRUE$ changing the control's parent height will move the control up or down by the same amount. When set to FALSE$ the control's position is not changed when the parent height is changed.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

If this property is set to TRUE$ the AUTOSIZEHEIGHT property is automatically set to FALSE$.

The BOTTOMANCHOR property is preserved and updated as necessary if the object is moved or resized.

In previous versions of OpenInsight this property was named ANCHORBOTTOM – this name is still supported for backwards compatibility.

Example

```
// Set the BOTTOMANCHOR of the EDB_NOTES EditBox control

Call Set_Property_Only( @Window : ".EDB_NOTES", "BOTTOMANCHOR", TRUE$ )
```

See also

Common GUI AUTOSIZEHEIGHT property, Common GUI BOTTOM property, Common GUI CLIENTHEIGHT property, Common GUI CLIENTSIZE property, Common GUI HEIGHT property, Common GUI RECT property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, Common GUI MOVE method, Common GUI OFFSET method, WINDOW SIZE event.

CANGETFOCUS property

Description

Indicates if an object can receive the input focus.

Property Value

This property is a boolean value. If TRUE$ then the object can receive the input focus.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

To be able to receive the input focus an object must pass each of the following tests:

- The object is a valid GUI object
- The ALLOWFOCUS property is TRUE$
- The object is enabled
- The object is visible

Example

```
// Check to see if the current control is allowed to get the focus

CanIHazFocus = Get_Property( CtrlEntID, "CANGETFOCUS" )
```

See also

Common GUI ALLOWFOCUS property, Common GUI ENABLED property, Common GUI FOCUS property, Common GUI HANDLE property, Common GUI VISIBLE property, SYSTEM FOCUS property, Common GUI GOTFOCUS event, Common GUI LOSTFOCUS event.

CLASSNAME property

Description

Returns the "window class name" of the object as registered with Windows. A window class is a set of attributes that the Windows uses as a template to create a GUI object such as a form or control. Every GUI object is a member of a window class.

Property Value

This property is a string value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

The CLASSNAME property is basically a wrapper around the GetClassName Windows API function, so more information on window classes can be found on the Microsoft website.

Example

```
// Get window class name of the current form

ClassName = Get_Property( @Window, "CLASSNAME" )
```

See also

N/a.

CHILDOBJECT property

Description

Returns TRUE$ if the control is a "child" object, i.e. it has the WS_CHILD style bit set.

Property Value

This property is a boolean value. If object is a child window this property returns TRUE$, otherwise it returns FALSE$.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

Only top-level forms (i.e. WINDOW objects that have the desktop as a "parent") return FALSE$ for the property. All other objects will return TRUE$.

For more information on the WS_CHILD style please refer to the Window Styles documentation on the Microsoft website.

Example

```
// Check if the current control is a child.

IsChild = Get_Property( CtrlEntID, "CHILDWINDOW" )
```

See also

Common GUI PARENT property, Common GUI PARENTFORM property, Common STYLE property, Common GUI SETPARENT method.

CLIENTHEIGHT property

Description

Specifies the height of an object's client area.

Property Value

This property is an integer value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
See remarks	Get/Set	No	Yes	No

Remarks

The client area is the part of an object which displays the content, such as text, data, and images etc.

The title bar, menu bar, window menu, minimize and maximize buttons, sizing border, edge and scroll bars are referred to collectively as the window's nonclient area. The system manages most aspects of the nonclient area, while the application manages the appearance and behavior of its client area. The CLIENTHEIGHT property excludes the height components of the nonclient area.

Note that this property is only supported at design time for WINDOW objects.

Example

```
// Set the CLIENTHEIGHT of the EDB_NOTES EditBox control to 200 DIPs high

Call Set_Property_Only( @Window : ".EDB_NOTES", "CLIENTHEIGHT", 200 )
```

See also

Common GUI AUTOSIZEHEIGHT property, Common GUI BOTTOMANCHOR property, Common GUI CLIENTSIZE property, Common GUI HEIGHT property, Common GUI RECT property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, WINDOW SCALEUNITS property, WINDOW SIZE event, Appendix K – High-DPI Programming.

CLIENTSIZE property

Description

Specifies the width and height of an object's client area.

Property Value

This property is an @FM-delimited array that contains two integer values representing the client area width and height respectively:

```
<1> Client Area Width
<2> Client Area Height
```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	Yes	No

Remarks

The client area is the part of an object which displays the content, such as text, data, and images etc.

The title bar, menu bar, window menu, minimize and maximize buttons, sizing border, and scroll bars are referred to collectively as the window's nonclient area. The system manages most aspects of the nonclient area, while the application manages the appearance and behavior of its client area. The CLIENTWIDTH property excludes the width components of the nonclient area.

Example

```
// Get the CLIENTSIZE of the EDB_NOTES EditBox control

ClientSize   = Get_Property( @Window : ".EDB_NOTES", "CLIENTSIZE" )

ClientWidth  = ClientSize<1>
ClientHeight = ClientSize<2>
```

See also

Common GUI AUTOSIZEWIDTH property, Common GUI CLIENTSIZE property, Common GUI RECT property, Common GUI RIGHTANCHOR property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, Common GUI WIDTH property, WINDOW SCALEUNITS property, WINDOW SIZE event, Appendix K – High-DPI Programming.

CLIENTWIDTH property

Description

Specifies the width of an object's client area.

Property Value

This property is an integer value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
See remarks	Get/Set	No	Yes	No

Remarks

The client area is the part of an object which displays the content, such as text, data, and images etc.

The title bar, menu bar, window menu, minimize and maximize buttons, sizing border, and scroll bars are referred to collectively as the window's nonclient area. The system manages most aspects of the nonclient area, while the application manages the appearance and behavior of its client area. The CLIENTWIDTH property excludes the width components of the nonclient area.

Note that this property is only supported at design time for WINDOW objects.

Example

```
// Set the CLIENTWIDTH of the EDB_NOTES EditBox control to 480 DIPs wide

Call Set_Property_Only( @Window : ".EDB_NOTES", "CLIENTWIDTH", 480 )
```

See also

Common GUI AUTOSIZEWIDTH property, Common GUI CLIENTSIZE property, Common GUI RECT property, Common GUI RIGHTANCHOR property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, Common GUI WIDTH property, Common GUI WINDOW SCALEUNITS property, WINDOW SIZE event, Appendix K – High-DPI Programming.

CLIPCHILDREN property

Description

Specifies if the object excludes child objects from its update region before painting itself.

Property Value

This property is a boolean value. If the object clips children from its update region this property returns TRUE$, otherwise it returns FALSE$.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

By default this property is set to TRUE$ because it prevents a parent object from painting over it's children which usually results in "flickering" during the redraw cycle. Note that this is different to previous versions of OpenInsight where this property was not available so objects did not have WS_CLIPCHILDREN style set.

For more information on this property please refer to the Windows documentation regarding the WS_CLIPCHILDREN window style on the Microsoft Website.

Example

```
// Ensure the current window has the WS_CLIPCHILDREN style set
// to prevent flickering.

Call Set_Property_Only( @Window, "CLIPCHILDREN", TRUE$ )
```

See also

Common GUI COMPOSITED property, Common GUI REDRAW property, Common GUI INVALIDATE method, Common GUI REPAINT method.

CLIPSIBLINGS property

Description

Specifies if the object excludes sibling objects from its update region before painting itself.

Property Value

This property is a boolean value. If the object clips siblings from its update region this property returns TRUE$, otherwise it returns FALSE$.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

By default, this property is set to TRUE$ because it prevents an object from painting over its siblings during the redraw cycle.

For more information on this property please refer to the Windows documentation regarding the WS_CLIPSIBLINGS window style on the Microsoft Website.

Example

```
// Ensure the current window has the WS_CLIPSIBLINGS style set.

Call Set_Property_Only( @Window, "CLIPSIBLINGS", TRUE$ )
```

See also

Common GUI REDRAW property, Common GUI INVALIDATE method, Common GUI REPAINT method.

COLUMN property

Description

If the object is data-bound then this property specifies the database column that it is bound to.

Property Value

This property is a string value and must be a valid database column name.

For controls that support associated multivalued data, like the EditTable control, this property will return an @Svm-delimited list of bound data columns at runtime.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get	No	No	Yes

Remarks

N/a.

Example

```
// Get the database column name of the EDL_SURNAME control

ColName = Get_Property( @Window : ".EDL_SURNAME", "COLUMN" )
```

See also

Common GUI PART property, Common GUI POS property, Common GUI MV property, Common GUI TABLE property, WINDOW ATRECORD property, WINDOW ID property, WINDOW RECORD property, Common GUI CALCULATE event.

COMPOSITED property

Description

Specifies if the object uses system double-buffering for painting.

Property Value

This property is a boolean value. If the object uses system double-buffering this property returns TRUE$, otherwise it returns FALSE$.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

This property is a thin wrapper over the WS_EX_COMPOSITED extended window style. When set it enables double-buffering on the object, which means that it renders into an off-screen buffer, and only when rendering is complete is the result copied to the screen. This avoids flicker because only a completely drawn object is put on the screen; it is never seen in a partially-drawn state.

By default this property is FALSE$, because many OpenInsight objects implement their own internal double buffering which can interfere with the COMPOSITED property, and it can also conflict with the Windows Desktop Window Manager (DWM) system.

For more information on this property please refer to the Windows documentation regarding the WS_EX_COMPOSITED extended window style on the Microsoft Website.

Example

```
// Ensure the current window has the WS_EX_COMPOSITED style removed

Call Set_Property_Only( @Window, "COMPOSITED", FALSE$ )
```

See also

Common GUI CLIPCHILDREN property, Common GUI DOUBLEBUFFER property, Common GUI REDRAW property, Common GUI STYLEEX property, Common GUI STYLEEXN property, Common GUI INVALIDATE method, Common GUI REPAINT method.

CONTEXTMENU property

Description

Specifies the CONTEXTMENU repository entity to use when a user right-clicks on an object.

Property Value

This property is a string value and must be a valid, fully-qualified CONTEXTMENU entity name.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

When setting this property at runtime the menu will not actually be instantiated until the first attempt to use it.

Example

```
// Set the name of the context menu for the IMG_PHOTO image control.

MenuName = @AppID<1> : "*CONTEXTMENU**PHOTO_MENU"

PrevName = Set_Property( @Window : ".IMG_PHOTO", "CONTEXTMENU", MenuName )
```

See also

Common GUI ATTACHMENU method, Common GUI SHOWMENU method, Common GUI TRACKPOPUPMENU method, Common GUI INITCONTEXTMENU event, Common GUI CONTEXTMENU event, Common GUI MENU event, ContextMenu stored procedure.

CONV property

Description

Specifies the Output Conversion pattern for a control. This pattern is used to convert internally formatted data before displaying it in the control.

For example, dates in OpenInsight are held internally as simple integer values – an output conversion needs to be applied to this data so that it becomes a meaningful date string for a user.

Property Value

This property is a string value and must be one of the following:

- A valid OpenInsight output conversion format such as "D4/E" etc.
- A null string (no output conversion applied).

At design time the following special strings may also be used:

- "<<None>>" – same as a null string, i.e. no conversion applied.
- "<<Default>>" – If the control is data-bound then the database column's output conversion is used, otherwise this is treated as a null string.

For controls that support associated multivalued data, like the EditTable control, this property will return an @Svm-delimited list of output conversion strings at runtime (one for each column in the control).

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	Yes

Remarks

N/a.

Example

```
// Set the output conversion for the EDL_DOB control to a four-year European format:
//
//    "D4/E" (dd/mm/yyyy)
//
// (you know, the date format that makes sense ;).

PrevConv = Set_Property( @Window : ".EDL_DOB", "CONV", "D4/E" )
```

See also

Common GUI INVALUE property, Common GUI VALID property.

CURSOR property

Description

Specifies the cursor to use when over a control.

Property Value

This property can be one of the following formats:

- A path and file name of a cursor (.CUR) file.
- A path and file name to a resource file (such as a DLL) containing the cursor image, along with its resource ID. The resource ID is separated from the file name by a "#" character.

  E.g.

  ```
  .\res\MyAppRes.Dll#192
  .\res\MyAppRes.Dll#MYCURSOR
  ```

  Note that if the cursor image is stored in a custom resource section (rather than the usual CURSOR section) the custom section name may be specified by inserting it before the resource name like so:

  ```
  .\res\MyAppRes.Dll#SOMESECTION#192
  .\res\MyAppRes.Dll#SOMESECTION#MYCURSOR
  ```

- A symbol that specifies one of the standard Windows cursors. These are:

Symbol	Description
A	Arrow
H	Wait
I	I-Beam (for text entry)
C	Cross
V	Vertical (Up) Arrow
&	Hand
S	App Starting
?	Help
N	No
+	Size All
****	Size NWSE
/	Size NESW
-	Size WE
|	Size NS
D	DragMove
DC	DragCopy
""	Set to null to use the control's default value.

Property Traits

Development	Runtime	Indexed	Scaled	Synthetic
N/a	Get/Set	No	No	No

Remarks

This property is used in response to the windows WM_SETCURSOR message to set the cursor shape when the cursor is over the control.

Note that Windows doesn't always send a WM_SETCURSOR message unless the mouse is moved, so it's possible that you may set the CURSOR property and not see a change. If you need to force a change straight after setting the property use the SYSTEM SETCURSOR method to do so.

Example

```
// Example - Set the cursor for the current window to an
// hourglass while it performs a task and then reset it
// afterwards

PrevCursor = Set_Property( @Window, "CURSOR", "H" )

// Because the system doesn't send a WM_SETCURSOR message
// unless the mouse is moved, it can look like the cursor
// is stuck, so we'll back up that property change with the
// SYSTEM CURSOR message too, because that change is
// immediate.

Call Exec_Method( "SYSTEM", "SETCURSOR", "H" )

GoSub ProcessAllTheThings

// Reset the cursor and force the change...
Call Set_Property( @Window, "CURSOR", PrevCursor )
Call Exec_Method( "SYSTEM", "SETCURSOR", PrevCursor )
```

See also

SYSTEM CURSOR property, SYSTEM SETCURSOR method.

CUSTOMPROPERTIES property

Description

Specifies a list of user-defined property names and values to apply to the object at runtime.

Property Value

This property is a list of custom property names and associated values that are turned into User Defined Properties (UDPs) at runtime.

Unlike creating a UDP with Set_Property there no need to prefix a property name with an "@" character.

A dynamic array value may be specified for a property by using the standard OI "[]" delimiter notation that is used by the System Monitor.

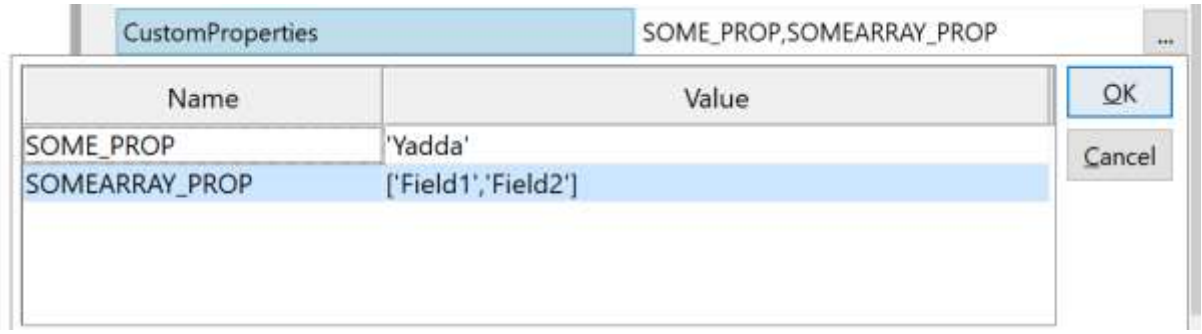
Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	N/a	No	No	No

Remarks

This is a design-time only property that allows the Form Designer to define one or more UDPs for the object that are created and applied when it is instantiated.

Example



Using the example above would create two user-defined properties:

- "@SOME_PROP" with a string value of "Yadda"
- "@SOMEARRAY_PROP" with a dynamic array value of:

```
<1> Field1
<2> Field2
```

See also

Common '@' (User-defined) property, Common UDPLIST property, Executing System Monitor Commands (for array notation),

DEFPROP property

Description

Specifies the *value* of the "default property" for a control. Controls that support DEFPROP have a property that they consider to be the "default", i.e. the one that represents their data contents the best – for example, an EditLine's DEFPROP is the TEXT property, while an EditTable's DEFPROP is the ARRAY property.

If the control is data-bound then setting the DEFPROP property will update the parent form's RECORD and SAVEWARN properties too.

Property Value

This property is usually a string or a dynamic array value depending on the designated default property for the control.

Control Type	*Default Property Name*
Animate	CLIPNAME
Bitmap	BITMAP
CheckBox	CHECKED
ColorDropDown	COLOR
ComboBox	TEXT
DateTime	VALUE
EditLine	TEXT
EditBox	TEXTVAL
EditTable	ARRAY
FileExplorer	PATH
FilePreview	FILENAME
GroupBox	TEXT
GroupBoxEx	TEXT
HScrollBar	HPOSITION
Hyperlink	LINK
ListBox	TEXT
Panel	TEXT
ProgressBar	VALUE
PropertyGrid	VALUE
PushButton	TEXT
RadioButton	VALUE
RichEditBox	RTFTEXT
Static	TEXT
TabControl	VALUE
TreeListBox	TEXT
UpDown	VALUE
VScrollBar	VPOSITION
Window	TEXT

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	Yes

Remarks

When using DEFPROP to update a data-bound control with a conversion format specified in it's CONV property, the new value is passed to the Iconv function first – this is to ensure that it is translated into appropriate internal format before being stored in the parent form's RECORD property. *If the Iconv conversion fails it does so silently, and a null value will be stored – it is assumed that the developer is responsible for ensuring any data used in programmatic updates is correct!*

This property is mainly implemented for the OpenInsight data-binding process, to allow generic code to interact with data-bound controls.

Example

```
// Set the DEFPROP (TEXT) of the EDL_SURNAME control:

PrevVal = Set_Property( @Window : ".EDL_SURNAME", "DEFPROP", PatientSurname )
```

See also

Common GUI CONV property, Common GUI DEFPROPID property, Common GUI DEFPROPPOS property, Common GUI DEFPROPPOSID property, Common GUI DEFPROPRAW property, WINDOW RECORD property, WINDOW SAVEWARN property.

DEFPROPID property

Description

Returns the name of a control's DEFPROP ("default property") property. Controls that support DEFPROP have a property that they consider to be the "default", i.e. one that represents their data contents – for example, an EditLine's DEFPROP is the TEXT property, while an EditTable's DEFPROP is the ARRAY property.

Property Value

This property contains the name of the property used for the default property value. See the DEFPROP property for a list of control types and their default property names.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

N/a.

Example

```
// Get the name of the default property for the EDL_SURNAME control:

DefPropID = Get_Property( @Window : ".EDL_SURNAME", "DEFPROPID" )
```

See also

Common GUI DEFPROP property, Common GUI DEFPROPRAW property.

DEFPROPPOS property

Description

Specifies the *value* of the "default position property" for a control. Controls that support DEFPROPPOS have a property that they consider to be the best one to represent the position for user interaction – for example, an EditTable's DEFPROPPOS is the CARETPOS property.

Property Value

This property is usually an integer or a dynamic array value depending on the designated default position property for the control.

Control Type	*Default Position Property Name*
EditTable	CARETPOS

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	Yes

Remarks

This property is mainly implemented for the OpenInsight data-binding process, to allow generic code to interact with data-bound controls.

Example

```
// Set the DEFPROPPOS (CARETPOS) of the EDT_INVOICES control:

PrevPos = Set_Property( @Window : ".EDT_INVOICES", "DEFPROPPOS", 1 : @Fm : rowNo )
```

See also

Common GUI DEFPROP property, Common GUI DEFPROPPOSID property.

DEFPOSPROPID property

Description

Returns the name of a control's DEFPROPPOS ("default position") property.

Property Value

This property contains the name of the property used to access the default position property value. See the DEFPROPPOS property for a list of control types and their default position property names.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

N/a.

Example

```
// Get the name of the default position property for the EDT_INVOICES
// EditTable control:

DefPropPosID = Get_Property( @Window : ".EDT_INVOICES", "DEFPROPPOSID" )
```

See also

Common GUI DEFPROPPOS property.

DEFPROPRAW property

Description

Emulates a control's DEFPROP property *without* updating the parent form's RECORD and SAVEWARN properties if the control is data-bound.

Property Value

This property is usually a string or a dynamic array value depending on the designated default property for the control. See the DEFPROP property for more details.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	Yes

Remarks

This property is mainly implemented for the OpenInsight data-binding process, to allow generic code to interact with data-bound controls.

Example

```
// Set the DEFPROPRAW (TEXT) of the EDL_SURNAME control:

PrevVal = Set_Property( @Window : ".EDL_SURNAME", "DEFPROPRAW", PatientSurname )
```

See also

Common GUI DEFPROP property, Common GUI DEFPROPID property, WINDOW RECORD property, WINDOW SAVEWARN property.

DEFVALUE property

Description

Specifies the default content to insert into a control when it is empty and receives the input focus.

Property Value

This property is a string value and may be one of the following special tokens:

Token	*Description*

COUNTER	Inserts the current value of a sequential counter and increments it by one ready for the next time it is used.
	
	This value may only be used for a data-bound control and is stored in the associated dictionary with a default key of:
	
	`"%S%*" <columnName>`
	
	However, a key may be specified with the COUNTER token as a suffix like so:
	
	`"COUNTER:" <keyNameToUse>`
	
	E.g.
	`COUNTER:WIDGET_COUNT`

DATE	Inserts the current date.
DATETIME	Inserts the current date and time.
PREVVAL	Inserts the previously saved value for a data-bound control.
SEQKEY	Inserts the next sequential key count for a data-bound form. The sequential key count itself is stored in the dictionary of the bound table with a key of "%SK%". A different key may be used by appending it to the "SEQKEY" token with a ":" delimiter. E.g. Using "SEQKEY:%MYCTR$" uses the "%MYCTR$" record to store the sequential key counter instead of the default "%SK%".
TIME	Inserts the current time.
TIMEDATE	Inserts the current date and time using the TimeDate() function.
USER	Inserts the current username.

"Literal"	A literal string value enclosed in quotes.
	
	E.g.
	`"Waiting"`
	`"TEL001"`

{CALCULATED}	The name of a symbolic (calculated) column enclosed in braces.
	
	E.g.
	`{CUSTOMER_FULL_NAME}`
	`{DELIVERY_FULL_ADDRESS}`

At design time the following special tokens may also be used:

- "<<None>>" – same as a null string, i.e. no default value is inserted.
- "<<Default>>" – If the control is data-bound then the database column's default value is used, otherwise this is treated as a null string as above.

If none of the above tokens are specified the DEFVALUE value is assumed to be the name of a stored procedure and its parameters:

```
<procName> "(" <param1>"," <param2> "," <param2> "," … etc ")"
```

E.g.

```
AMSYS_PATREC( "GETDEF", "@FOCUS", "KCODE" )
@COMMUTER( "@SELF", "GETDEFVAL" )
```

Note that the same rules for specifying QuickEvent parameters apply:

The following special "@" placeholder tokens may be used:

- @COMMUTER
- @EVENT
- @FOCUS
- @MDIACTIVE
- @MDIFRAME
- @NEXT
- @PARAM1 (PrevFocusID from the calling GOTFOCUS event)
- @PREV
- @SELF
- @WINDOW

The standard "[]" notation maybe used for specifying dynamic arrays as parameters.

For controls that support associated multivalued data, like the EditTable control, this property will return an @Svm-delimited list of default value strings at runtime (one for each column in the control).

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	Yes

Remarks

This property is used by the system GOTFOCUS event handler.

The form designer uses a list of options to define this property at design time. The list itself is stored in the SYSENV table in the CFG_RTI_IDE_DEFVALUE record.

Example

```
// Set the DEFVALUE property of the EDL_NAME control to call the
// form's commuter module property passing the name of the control
// and a method called GET_DEF_NAME_VAL

Call Set_Property_Only( @Window : ".EDL_NAME", "DEFVALUE",          |
                        "@COMMUTER( '@FOCUS', 'GET_DEF_NAME_VAL' )" )


// Set the DEFVALUE property of the EDL_ADDRESS EditLine to use the
// result of the FULL_ADDRESS_ONE_LINE symbolic column

Call  Set_Property_Only( @Window : ".EDL_ADDRESS", "DEFVALUE", |
                                    "{FULL_ADDRESS_ONE_LINE}" )


// Set the DEFVALUE property of the EDL_Notes EditLine to use the
// string "No Notes"

Call  Set_Property_Only( @Window : ".EDL_ADDRESS", "DEFVALUE", |
                                                  '"No Notes"' )
```

See also

Common GUI CONV property, Common GUI DEFPROP property, Common GUI INVALUE property, WINDOW ALLOWSEQKEYRESET property, Common GUI GOTFOCUS event,

DESIGNMODE property

Description

Indicates if the control is in "design mode", i.e. it is currently hosted in the Form Designer.

Property Value

This property is boolean value. It returns TRUE$ if the control is in "design mode", or FALSE$ otherwise.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

N/a.

Example

```
// Check if the current control is in DesignMode

IsDesignMode = Get_Property( ctrlEntID, "DESIGNMODE" )
```

See also

Common GUI DESIGNSELECTED property.

DESIGNSELECTED property

Description

Indicates if the control is currently selected in the Form Designer.

Property Value

This property is boolean value. It returns TRUE$ if the control is selected, or FALSE$ otherwise.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

N/a.

Example

```
// Check if the current control is selected in the Form Designer

IsDesignSel = Get_Property( ctrlEntID, "DESIGNSELECTED" )
```

See also

Common GUI DESIGNMODE property.

DOUBLEBUFFER property

Description

Indicates if an object is using double-buffering for drawing operations.

Property Value

This property is boolean value. It returns TRUE$ if the object is using double-buffering, or FALSE$ otherwise.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

Not all objects support this property – for those that do the default value will be TRUE$.

This property is not the same as the system double-buffering mentioned in the COMPOSITED property.

Using this property to turn off double-buffering is intended to be a trouble-shooting tool rather than a normal option.

Example

```
// Check if the current control is double-buffering for painting.

IsDoubleBuffered = Get_Property( ctrlEntID, "DOUBLEBUFFER" )
```

See also

Common GUI COMPOSITED property, Common GUI REDRAW property, Common GUI INVALIDATE method, Common GUI REPAINT method.

DPI property

Description

Returns the current DPI (dots-per-inch) settings for the object.

Property Value

This property is an @Fm-delimited array containing the DPI values:

```
<1> X (horizontal) DPI
<2> Y (vertical) DPI
```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

The DPI property from a control is always the same as its top-level parent form.

Beginning with Windows 8.1 individual monitors can have their own DPI settings. This property returns the DPI for the monitor that the top-level parent form is currently displayed on, or that the majority of the form is displayed on if using more than one monitor. Prior to this the form DPI is always the same as the SYSTEM DPI property.

The form DPI is combined with its SCALEFACTOR property when calculating scaling attributes.

Example

```
// Get the DPI settings for the current control

CtrlDPI = Get_Property( ctrlEntID, "DPI" )
```

See also

SYSTEM DPI property, WINDOW DPI property, WINDOW SCALEFACTOR property, WINDOW SCALED event, Appendix K – High-DPI Programming.

ECHO property

Description

Specifies if the control displays keystrokes entered by the user.

Property Value

This property is a boolean value. When set to FALSE$ a CHAR event is still raised but entered characters are not displayed. The default is TRUE$.

Property Traits

Development	*Runtime*		*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set		No	No	No

Remarks

N/a.

Example

```
// Turn off the "keyboard echo" for the EDL_NAME EditLine control.

PrevVal = Set_Property( @Window : ".EDL_NAME", "ECHO", FALSE$ )
```

See also

Common GUI VALIDCHARS property, Common GUI CHAR event.

EDGESTYLE property

Description

Specifies the appearance of a control's non-client area border. For most controls Windows provides a border that it maintains and paints to match the currently selected visual style.

Property Value

This property is an integer value that specifies if the style of the border. It may be one of the following values:

Value	*Description*
0	None – the control is drawn without an edge.
1	Single - the control is drawn with a simple single pixel border.
2	Sunken – the control is draw with a "sunken" edge style. Note that the term "sunken" is a throwback to the old "Classic Windows" theme – in modern themes this style is usually rendered as single pixel edge with another single pixel inner border.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

The Single and Sunken edge values map into the Windows WS_EX_STATICEDGE and WS_EX_CLIENTEDGE extended window styles respectively. For more information on these styles please refer to the Windows documentation on the Microsoft website.

Equated constants for use with the EDGESTYLE property can be found in the PS_EQUATES insert record.

Example

```
// Set the current control's EDGESTYLE to "sunken"
$Insert PS_Equates

Call Set_Property_Only( CtrlEntID, "EDGESTYLE", PS_EGS_SUNKEN$ )
```

See also

Common GUI STYLEEX property, Common GUI STYLEEXN property.

ENABLED property

Description

Enables or disables mouse and keyboard input to a control. When input is disabled, the window does not receive input such as mouse clicks or key presses.

Property Value

The ENABLED property is an integer value that specifies if the control is enabled. For a standard control it can be one of the following values:

Value	*Description*
0	The control is disabled.
1	The control is enabled.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

Most controls treat this property as a boolean value, but there are exceptions to this such as the EDITFIELD control. Any such exceptions are noted in the individual control descriptions.

Example

```
$Insert Logical

// Example - Disabling a control
Call Set_Property_Only( CtrlEntID, "ENABLED", FALSE$ )
```

See also

N/a.

FOCUS property

Description

Indicates if a control has the system input focus, or sets the input focus to a control.

Property Value

This property is a boolean value. TRUE$ if the control has the input focus, or FALSE$ otherwise.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

When setting the input focus via this property the PS will remove all events from the event *queue both before and after setting the actual focus*. This was designed to allow the system to reset the focus back to a problem control when a validation error was encountered (validation usually occurs on a LOSTFOCUS event), but without triggering any more conflicting validation checks. Because events are removed wholesale during this process it is not considered the best way to move the input focus in your applications – the best method is to use the SYSTEM FOCUS property along with the SYSTEM BLOCKEVENTS property if needed.

Setting the focus to a control on an MDI Child window will cause the MDI child to be activated first if necessary.

Example

```
// Check if the current control has the input focus

bFocused = Get_Property( CtrlEntID, "FOCUS" )

// Move the focus to the EDL_SURNAME control without triggering
// any focus-based events:
//
// Don't do this:
//
//    Call Set_Property_Only( @Window : ".EDL_SURNAME", "FOCUS", TRUE$ )
//
// Do this instead:

Call Set_Property_Only( "SYSTEM", "BLOCKEVENTS", TRUE$ )
Call Set_Property_Only( "SYSTEM", "FOCUS", @Window : ".EDL_SURNAME" )
Call Set_Property_Only( "SYSTEM", "BLOCKEVENTS", FALSE$ )
```

See also

SYSTEM FOCUS property, Common GUI GOTFOCUS event, Common GUI LOSTFOCUS event, WINDOW ACTIVATED event.

FONT property

Description

Specifies the font used to display text in a control.

Property Value

This property is an @Svm-delimited array font values. For getting and setting a font the first twelve sub-values are the same:

Position	Name	Description
<0,0,1>	Facename	Name of the font, e.g. "Arial" or "Courier New" etc.
<0,0,2>	Height	Height of the font's character cell in pixels (expressed as a negative value).
<0,0,3>	Weight	The weight of the font from 0 to 1000 – normal is 400, bold is 700 etc.
<0,0,4>	Italic	TRUE$ if the font is italic.
<0,0,5>	Underline	TRUE$ if the font is underlined.
<0,0,6>	Width	Average width of characters in the font.
<0,0,7>	CharSet	ID of the font character set.
<0,0,8>	PitchAndFamily	Pitch and Family values of the font.
<0,0,9>	StrikeOut	TRUE$ if the font is a strikeout font.
<0,0,10>	OutPrecision	Output precision of the font.
<0,0,11>	ClipPrecision	Clipping precision of the font.
<0,0,12>	Quality	Output quality of the font.

When getting the FONT property, the following extra sub-value members are returned:

Position	Name	Description
<0,0,13>	Ascent	The ascent (units above the base line) of characters.
<0,0,14>	Internal Leading	The amount of leading (space) inside the bounds set by the Height.
<0,0,15>	External Leading	The amount of extra leading (space) that the application adds between rows.
<0,0,16>	MaxCharWidth	The width of the widest character in the font.

When setting the FONT property, the following extra sub-value members can be applied (they are optional).

Position	Name	Description
<0,0,13>	Red	Red color intensity for the FORECOLOR.
<0,0,14>	Green	Green color intensity for the FORECOLOR.
<0,0,15>	Blue	Blue color intensity for the FORECOLOR.

- Sub-values 13-15 can be used to set the FORECOLOR property of the object using the individual Red, Green and Blue color intensities to produce an RGB color value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	Yes	No

Remarks

This property sets the default font for a control. Certain controls may support other styling options that can override the normal FONT property.

The sub-values in the FONT property map onto several members of the LOGFONT and TEXTMETRIC Windows C-structures, so for more information on these attributes please refer to the Windows documentation on the Microsoft website.

Equated constants for use with the FONT property can be found in the following insert records:

- PS_FONT_EQUATES
- MSWIN_LOGFONT_EQUATES

Example

```
$Insert MSWin_LogFont_Equates
$Insert PS_Font_Equates

// Get the FONT for the current control

CtrlFont = Get_Property( CtrlEntID, "FONT" )

// And make sure it is bold and underlined

CtrlFont<0,0,PS_FONT_POS_WEIGHT$>    = FW_BOLD$
CtrlFont<0,0,PS_FONT_POS_UNDERLINE$> = TRUE$

Call Set_Property_Only( CtrlEntID, "FONT", CtrlFont )
```

See also

Common GUI FORECOLOR property, SYSTEM CHOOSEFONT method, RBG stored procedure.

FORECOLOR property

Description

Specifies the RGB color value used draw text in a control.

An RBG color is an integer value that is constructed from a combination of Red, Green and Blue color intensities using the following formula:

RGB = (Red) + (Green * 256) + (Blue * 65536)

Each of the color intensities should be a value between 0 and 255.

In addition to this there are also a set of "special" values that are not valid RBG colors, but represent default system colors that can be used:

```
          -1   : Transparent
          -2   : Use default color for the object
               :
2147483648     : Use a Windows system colors
    to          : (See the COLORS Insert record
2483027968     : for more details)
```

Property Value

This property is a single RGB value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

Common RGB color values can be found in the COLORS insert record.

Example

```
$Insert PS_Equates
$Insert Colors

// Set the FORECOLOR of the LBL_NAME control to Blue

prevColor = Set_Property( @Window : ".LBL_NAME", "FORECOLOR", BLUE$ )

// Set the BACKCOLOR property of the LBL_NAME control to "Use Default"

Call Set_Property_Only( @Window : ".LBL_NAME", "FORECOLOR", CLR_USEDEFAULT$ )
```

See also

Common GUI BACKCOLOR property, Common GUI FONT property, SYSTEM CHOOSECOLOR method, RBG stored procedure.

GOTFOCUSVALUE property

Description

Returns the value for a control when it received the input focus.

Property Value

This property is usually a string or a dynamic array value depending on the type of default property (DEFPROP) for the control.

Property Traits

Development	Runtime	Indexed	Scaled	Synthetic
N/a	Get	No	No	Yes

Remarks

The DEFPROP property is used to obtain the value when the control receives focus where it is cached for later comparison.

This property is mainly implemented for the OpenInsight validation process and is set in the system GOTFOCUS event handler.

In previous versions of OpenInsight this property was named GOTFOCUS_VALUE – this name is still supported for backwards compatibility.

Example

```
// Get the GOTFOCUSVALUE (TEXT) of the EDL_SURNAME control:

GFVal = Get_Property( @Window : ".EDL_SURNAME", "GOTFOCUSVALUE" )
```

See also

Common GUI DEFPROP property, Common GUI DEFPROPID property, Common GUI FOCUS property, SYSTEM FOCUS property, Common GUI GOTFOCUS event, Common GUI LOSTFOCUS event.

HANDLE property

Description

Returns the internal Windows "handle" (unique ID), for an object. This is commonly referred to in Windows documentation as the HWND.

Property Value

This property is an unsigned integer. If the object referred to is not valid then a null value is returned.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

Because this property returns a null value if used with an invalid object ID it is often used to test if that object exists (see example below).

Example

```
// Get the Windows handle for current form

hwnd = Get_Property( CtrlEntID, "HANDLE" )

// Check if the OpenInsight IDE (RTI_IDE) is running

If bLen( Get_Property( "RTI_IDE", "HANDLE" ) ) Then
   // IDE is running...
End
```

See also

N/a.

HEIGHT property

Description

Specifies the height of an object.

Property Value

This property is an integer value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	Yes	No

Remarks

The HEIGHT property includes an object's non-client area as well the client area.

Example

```
// Set the HEIGHT of the EDB_NOTES EditBox control to position 200 DIPs

Call Set_Property_Only( @Window : ".EDB_NOTES", "HEIGHT", 200 )
```

See also

Common GUI AUTOSIZEHEIGHT property, Common GUI BOTTOMANCHOR property, Common GUI CLIENTHEIGHT property, Common GUI CLIENTSIZE property, Common GUI RECT property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, WINDOW SCALEUNITS property, WINDOW SIZE event, Appendix K – High-DPI Programming.

IMAGE property

Description

Specifies the properties of an object's IMAGE sub-object at *design-time*.

Property Value

This property is a collection of IMAGE object properties and values that are compiled into the form structure by the Form Designer and applied to an IMAGE sub-object at runtime.

The following properties for an image can be set in the Form Designer using the Image Property Editor dialog:

- Entity ID (this is a repository ID, not a file name/path)
- Alignment
- AutoScale flag
- Image index
- Image Offset
- Image Origin
- Image Style (Clip, Tile etc).
- Translucency

The following image properties cannot be edited here because they are intrinsic to the chosen image entity itself (ColorKey and Image Count *can* be edited in the IDE's Image Designer tool however).

- ColorKey
- Image Count
- FrameCount
- Size

Property Traits

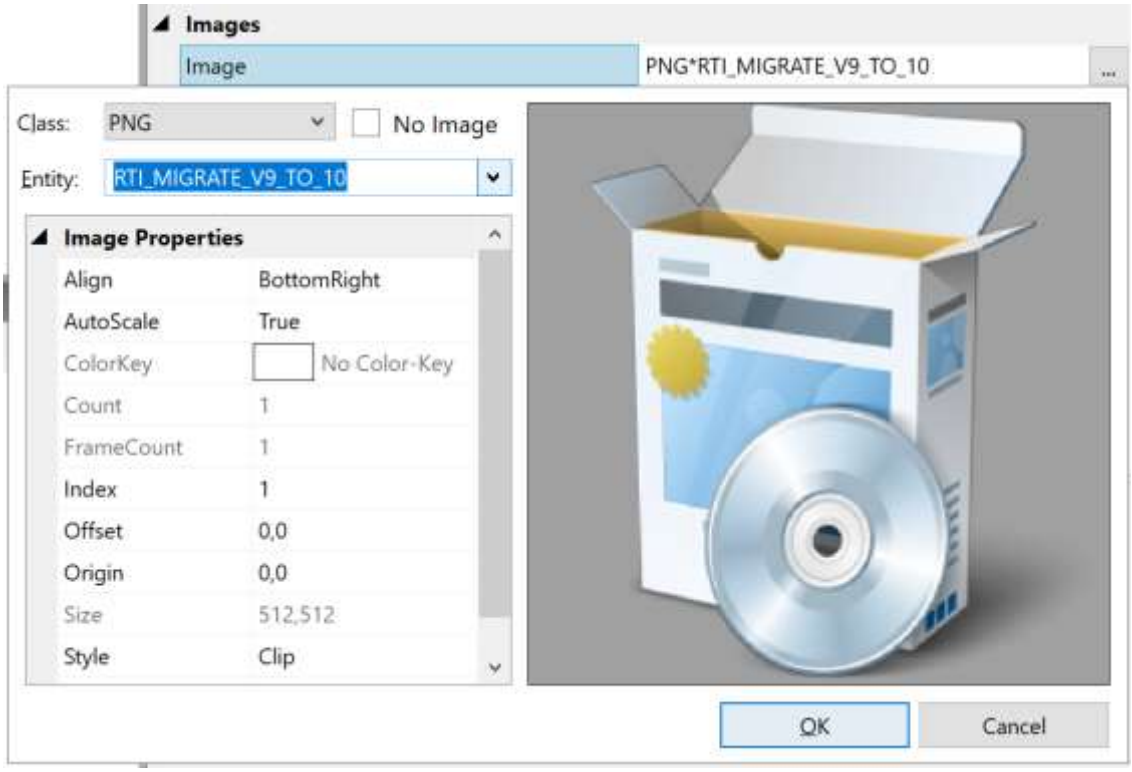
Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	N/a	No	No	No

Remarks

This is a *design-time* only property that allows the Form Designer to define properties for the IMAGE sub-object.

Most, but not all, objects support an IMAGE sub-object so please consult the individual object type documentation for more details.

Example



At runtime the equivalent code, using the Image Object API, would be:

```
// Example showing hwo to set properties of the current window's IMAGE
// sub-object at runtime

$Insert PS_Equates

ImageObjID = @Window : ".IMAGE"

ImageEntity = @AppID<1> : "*IMAGE*PNG*RTI_MIGRATE_V9_TO_10"

Call Exec_Method( ImageObjID, "SETREPOSIMAGE", ImageEntity )
Call Set_Property_Only( ImageObjID, "ALIGN", PS_IA_BOTTOMRIGHT$ )
Call Set_Property_Only( ImageObjID, "AUTOSCALE", TRUE$ )
Call Set_Property_Only( ImageObjID, "INDEX", 1 )
Call Set_Property_Only( ImageObjID, "OFFSET", 0 : @Fm : 0 )
Call Set_Property_Only( ImageObjID, "ORIGIN", 0 : @Fm : 0 )
Call Set_Property_Only( ImageObjID, "STYLE", PS_IS_CLIP$ )
Call Set_Property_Only( ImageObjID, "TRANSLUCENCY", 70 )
```

See also

Common GUI BITMAP property, IMAGE Object API, IMAGELIST Object API

IMAGELIST property

Description

Specifies the properties of an object's IMAGELIST sub-object at *design-time*.

Property Value

This property is a collection of IMAGELIST object properties and values that are compiled into the form structure by the Form Designer and applied to an IMAGELIST sub-object at runtime.

The following properties for an imagelist can be set in the Form Designer using the Image Property Editor dialog:

- Entity ID (this is a repository ID, not a file name/path)
- AutoScale flag

The following imagelist properties cannot be edited here because they are intrinsic to the chosen entity itself (ColorKey and Image Count *can* be edited in the IDE's Image Designer tool however).

- ColorKey
- Image Count
- Frame Count
- Size

Property Traits

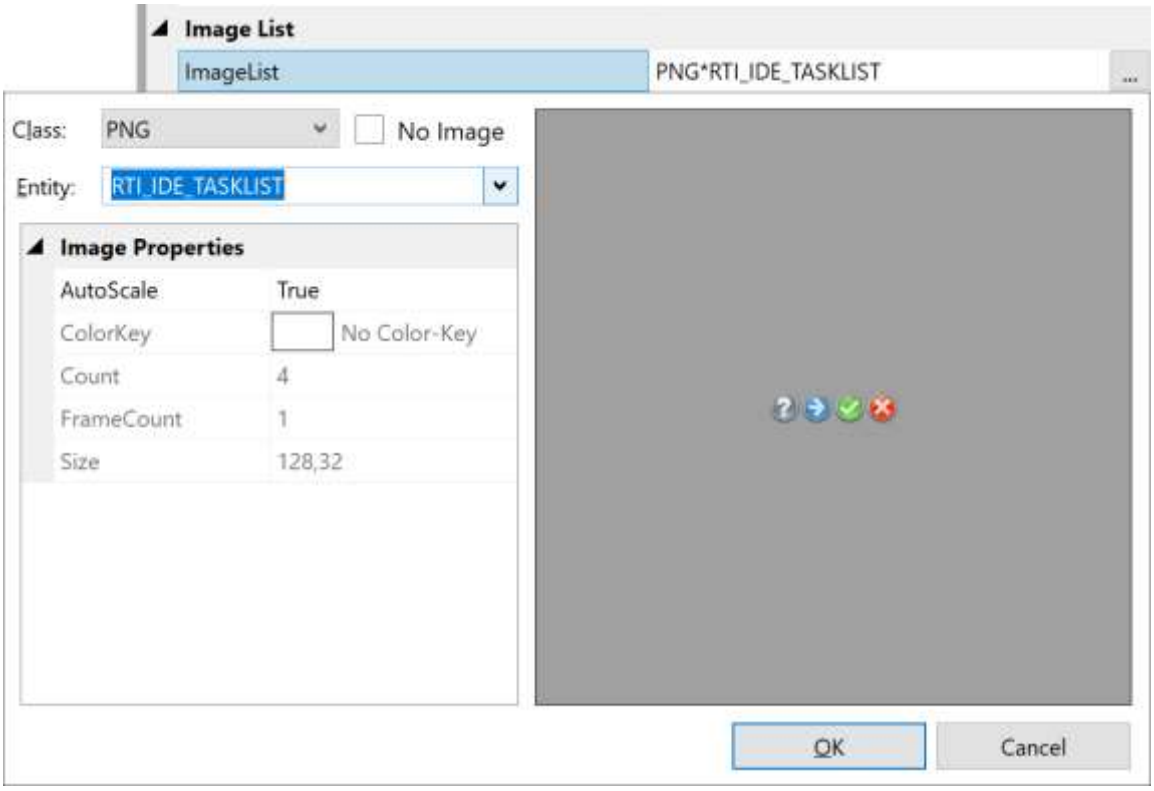
Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	N/a	No	No	No

Remarks

This is a *design-time* only property that allows the Form Designer to define properties for the IMAGELIST sub-object.

Some, but not all, objects support an IMAGELIST sub-object so please consult the individual object type documentation for more details.

Example



At runtime the equivalent code, using the Image Object API, would be:

```
// Example showing how to set properties of the LST_TASKS IMAGELIST
// sub-object at runtime

$Insert PS_Equates

ImageListObjID = @Window : ".LST_TASKS.IMAGE"

ImageEntity = @AppID<1> : "*IMAGELIST*PNG*RTI_IDE_TASKLIST"

Call Exec_Method( ImageListObjID , "SETREPOSIMAGE", ImageEntity )
Call Set_Property_Only( ImageObjID, "AUTOSCALE", TRUE$ )
```

See also

IMAGE object API, IMAGELIST Object API

INVALUE property

Description

Specifies the *value* of the "default property" for a control, but in "internal format if a conversion pattern (CONV property) is also defined.

This property is essentially the same as the DEFPROP property but allows the value to be specified in internal format rather than the usual external format. For example, a date string such as "31/01/2021" has an internal integer format of "19724", and INVALUE allows you to use the latter representation when getting and setting the value of a control (assuming a conversion pattern of "D4/E")

As with DEFPROP, if the control is data-bound then setting the INVALUE property will update the parent form's RECORD and SAVEWARN properties too.

Property Value

This property is usually a string or a dynamic array value depending on the designated default property for the control. See the DEFPROP property for more details.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	Yes

Remarks

When getting the INVALUE property the DEFPROP process is used to get the data in external format which is then passed to the Iconv() function to be converted into an internal format.

When setting the INVALUE property the data is first passed to the Iconv() function for conversion to an external format before being passed to the DEFPROP process to be set in the control.

If the control does not have a conversion pattern defined in the CONV property then no conversions take place and INVALUE behaves exactly like DEFPROP.

Example

```
// Get the date from the EDL_EXPIRY control (assume the CONV property has a
// pattern of "D4/E"), and increment it by 7 days

IDate = Get_Property( @Window : ".EDL_EXPIRY", "INVALUE" )

// IDate is in internal integer format so it's easy to increment
IDate += 7

// Put the updated value back in the control - the user will see the
// normal "dd/mm/yyyy" format
Call Set_Property_Only( @Window : ".EDL_EXPIRY", "INVALUE", IDate )
```

See also

Common GUI CONV property, Common GUI DEFPROP property, Common GUI DEFPROPID property, Common GUI DEFPROPRAW property, WINDOW RECORD property, WINDOW SAVEWARN property.

LEFT property

Description

Specifies the left coordinate of an object relative to its parent object.

Property Value

This property is an integer value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	Yes	No

Remarks

Setting this property adjusts the left coordinate of the object, it does not affect the width so the object is moved, not resized.

Coordinates are always relative to the top left corner of the screen, or, for a child window, the upper left corner of the parent window's client area.

Example

```
// Set the LEFT coordinate of the EDB_NOTES EditBox control to position 8

Call Set_Property_Only( @Window : ".EDB_NOTES", "LEFT", 8 )
```

See also

Common GUI AUTOSIZEWIDTH property, Common GUI CLIENTSIZE property, Common GUI CLIENTWIDTH property, Common GUI RECT property, Common GUI RIGHTANCHOR property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, Common GUI WIDTH property, Common GUI MOVE method, Common GUI OFFSET method, WINDOW SCALEUNITS property, WINDOW SIZE event, Appendix K – High-DPI Programming.

MONITOR property

Description

Returns the details for the monitor that is nearest to the specified object (i.e. has the largest area of intersection with the coordinates of the specified object).

Property Value

This property is @Fm-delimited dynamic array of monitor information with the following format:

```
<1> Monitor Handle
<2> Display Rectangle (@Vm-delimited)
      <2,1> Display Left
      <2,2> Display Top
      <2,3> Display Right
      <2,4> Display Bottom
<3> Work-area Rectangle (@Vm-delimited)
      <3,1> Work-area Left
      <3,2> Work-area Top
      <3,3> Work-area Right
      <3,4> Work-area Bottom
<4> Flags
<5> Device name
<6> DPI X value
<7> DPI Y value
```

The Display Rectangle contains the coordinates of the entire monitor surface. The Work-area rectangle contains the coordinates of the entire monitor surface minus the TaskBar and any docked "AppBars".

Note that the returned coordinates will be scaled with respect to the monitor DPI.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	Yes	No

Remarks

If the object's coordinates intersect one or more display monitor rectangles, the return value is the display monitor that has the largest area of intersection with the object.

This property uses the MonitorFromWindowWindows API function internally, so please refer to the documentation on the Microsoft website for further information.

Constants for use with this property can be found in the MSWin_Monitor_Equates and PS_Monitor_Equates insert records.

Example

```
// Get the details for the monitor that the current form
// is displayed on

MonitorInfo = Get_Property( @Window, "MONITOR" )
```

See also

Common GUI DPI property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, SYSTEM MONITORLIST property, SYSTEM SIZE property, Appendix K – High-DPI Programming.

MOUSECAPTURED property

Description

Specifies if mouse messages are redirected to ("captured by") a specific object.

Property Value

This property is a boolean value. If TRUE$ then Windows will direct all mouse messages to the specified object.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

Once the mouse is captured by an object all mouse messages are re-directed at that object until the MOUSECAPTURED property is set to FALSE$, or something like another application gaining the foreground status which will force a release. When this happens a LOSTCAPTURE event is sent to the object to notify it of the change.

This property uses the SetCapture and ReleaseCapture Windows API functions internally, so please refer to the documentation on the Microsoft website for further information on mouse capturing.

Example

```
// Example BUTTONDOWN event code - check if the user wants to "drag"
// the current object, and if so capture the mouse messages so that
// all subsequent MOUSEMOVE events will be directed to it.

If Exec_Method( CtrlEntID, "DRAGDETECT", MouseButton, xDown, yDown ) Then
   //  User wants to drag, so capture the mouse...
   Call Set_Property_Only( CtrlEntID, "MOUSECAPTURED", TRUE$ )
End
```

See also

Common GUI CURSOR property, Common GUI DRAGDETECT method, Common GUI BUTTONDOWN event, Common GUI BUTTONUP event, Common GUI LOSTCAPTURE event, Common GUI MOUSEMOVE event.

NEXT property

Description

Specifies the next control in the tab order from the specified control.

Property Value

This property is a string value that should contain a valid, fully qualified, control name.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

This property can be used to dynamically change the tab order of control on a form at runtime.

Example

```
// Skip over the next control in the tab order to the
// one after it

NextCtrl         = Get_Property( CtrlEntid, "NEXT" )
TheCtrlAfterNext = Get_Property( NextCtrl, "NEXT" )

Call Set_Property_Only( CtrlEntId, "NEXT", TheCtrlAfterNext )

// Now update that control so "tabbing back" from it goes to
// this control

Call Set_Property_Only( TheCtrlAfterNext, "PREVIOUS", CtrlEntID )
```

See also

Common GUI PREVIOUS property.

ORIGARRAY property

Description

Returns the original "List" attribute from the structure used to create the object at runtime, but in EditTable-style "Array" format.

Property Value

This property is an @Svm/@Tm-delimited dynamic array containing the data written to the object's List attribute when it was saved in the Form Designer, but returned in "Array" format – i.e. @Svm-delimited Columns, with @Tm-delimited Rows.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_LIST$ value from the ORIGSTRUCT property, but with the delimiters transposed by using the SYSTEM LIST2ARRAY method.

Note that not all object types use or support an Array attribute. Please consult the documentation for the specific control types for more details.

Example

```
// Get the Array used when the current control was created

OrigArray = Get_Property( CtrlEntID, "ORIGARRAY" )
```

See also

Common GUI ARRAY property, Common GUI ORIGSTRUCT property, SYSTEM CREATE method, SYSTEM LIST2ARRAY method.

ORIGBACKCOLOR property

Description

Returns the original "BackColor" attribute from the structure used to create the object at runtime.

Property Value

This property is an @Svm-delimited dynamic array containing the BackColor values that were set in the Form Designer when the object was saved.

```
<0,0,1> Color From (RGB value)
<0,0,2> Color To (RGB value)
<0,0,3> Gradient style
```

If a simple solid color was chosen only the first sub-value will be present.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_BKCOLOR$ value from the ORIGSTRUCT property.

In previous versions of OpenInsight this property was named ORIG_BACKCOLOR – this name is still supported for backwards compatibility.

Example

```
// Get the BackColor attributes when the current control was created

OrigBkColor = Get_Property( CtrlEntID, "ORIGBACKCOLOR" )
```

See also

Common GUI BACKCOLOR property, Common GUI ORIGSTRUCT property, SYSTEM CREATE method.

ORIGENABLED property

Description

Returns the original "Enabled" attribute from the structure used to create the object at runtime.

Property Value

This property is an integer containing the Enabled value that was set in the Form Designer when the object was saved.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_ENABLED$ value from the ORIGSTRUCT property.

In previous versions of OpenInsight this property was named ORIG_ENABLED – this name is still supported for backwards compatibility.

Example

```
// Get the Enabled value when the current control was created

OrigEnabled = Get_Property( CtrlEntID, "ORIGENABLED" )
```

See also

Common GUI ENABLED property, Common GUI ORIGSTRUCT property, SYSTEM CREATE method.

ORIGFONT property

Description

Returns the original "Font" value from the structure used to create the object at runtime.

Property Value

This property is an @Svm-delimited dynamic array representing the Font that was set in the Form Designer when the object was saved.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_LOGFONT$ value from the ORIGSTRUCT property.

In previous versions of OpenInsight this property was named ORIG_FONT – this name is still supported for backwards compatibility.

Example

```
// Get the Font value when the current control was created

OrigFont = Get_Property( CtrlEntID, "ORIGFONT" )
```

See also

Common GUI FONT property, Common GUI ORIGSTRUCT property, SYSTEM CREATE method.

ORIGFORECOLOR property

Description

Returns the original "ForeColor" value from the structure used to create the object at runtime.

Property Value

This property is an integer representing the RBG value for the ForeColor value that was set in the Form Designer when the object was saved.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_FORECOLOR$ value from the ORIGSTRUCT property.

In previous versions of OpenInsight this property was named ORIG_FORECOLOR – this name is still supported for backwards compatibility.

Example

```
// Get the ForeColor value when the current control was created

OrigFgColor = Get_Property( CtrlEntID, "ORIGFORECOLOR" )
```

See also

Common GUI FORECOLOR property, Common GUI ORIGSTRUCT property, SYSTEM CREATE method.

ORIGHEIGHT property

Description

Returns the original Height value used when the object was created at runtime.

Property Value

This property is an integer containing the Height coordinate that was set in the Form Designer when the object was saved.

If this is a negative value it refers to the desired client-area height only (e.g. "-300" means the object was created with a client area height of "300" DIPs). Negative values only apply to WINDOW (Form) objects.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_HIGH$ value from the ORIGSTRUCT property.

In previous versions of OpenInsight this property was named ORIG_HIGH – this name is still supported for backwards compatibility.

Example

```
// Get the Height value used when the current control was created

OrigHeight = Get_Property( CtrlEntID, "ORIGHEIGHT" )
```

See also

Common GUI HEIGHT property, Common GUI ORIGHIGH property, Common GUI ORIGSIZE property, Common GUI ORIGSTRUCT property, SYSTEM CREATE method.

ORIGHIGH property

Description

Returns the original Height value used when the object was created at runtime.

Property Value

This property is an integer containing the Height coordinate that was set in the Form Designer when the object was saved.

If this is a negative value it refers to the desired client-area height only (e.g. "-300" means the object was created with a client area height of "300" DIPs). Negative values only apply to WINDOW (Form) objects.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property is a synonym for the ORIGHEIGHT property.

In previous versions of OpenInsight this property was named ORIG_HIGH – this name is still supported for backwards compatibility.

Example

```
// Get the Height value used when the current control was created

OrigH = Get_Property( CtrlEntID, "ORIGHIGH" )
```

See also

Common GUI HEIGHT property, Common GUI ORIGHEIGHT property, Common GUI ORIGSIZE property, Common GUI ORIGSTRUCT property, SYSTEM CREATE method.

ORIGLABEL property

Description

Returns the original "Label" value from the structure used to create the object at runtime.

Property Value

This property is an @Svm-delimited dynamic array representing the Label values that were set in the Form Designer when the object was saved.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_LABEL$ value from the ORIGSTRUCT property.

Note that not all object types use or support a Label attribute. Please consult the documentation for the specific control types for more details.

In previous versions of OpenInsight this property was named ORIG_LABEL – this name is still supported for backwards compatibility.

Example

```
// Get the Label value when the current control was created

OrigLabel = Get_Property( CtrlEntID, "ORIGLABEL" )
```

See also

Common GUI ORIGSTRUCT property, SYSTEM CREATE method.

ORIGLEFT property

Description

Returns the original Left coordinate used when the object was created at runtime.

Property Value

This property is an integer containing the Left coordinate that was set in the Form Designer when the object was saved.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_X$ value from the ORIGSTRUCT property.

Example

```
// Get the Left coordinate used when the current control was created

OrigLeft = Get_Property( CtrlEntID, "ORIGLEFT" )
```

See also

Common GUI LEFT property, Common GUI ORIGSTRUCT property, Common GUI ORIGX property, SYSTEM CREATE method.

ORIGLIST property

Description

Returns the original "List" attribute from the structure used to create the object at runtime.

Property Value

This property is an @Svm/@Tm-delimited dynamic array containing the data written to the object's List attribute when it was saved in the Form Designer, i.e. @Svm-delimited Rows, with @Tm-delimited Columns (if supported).

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_LIST$ value from the ORIGSTRUCT property.

Note that not all object types use or support a List attribute. Please consult the documentation for the specific control types for more details.

Example

```
// Get the List used when the current control was created

OrigList = Get_Property( CtrlEntID, "ORIGLIST" )
```

See also

Common GUI LIST property, Common GUI ORIGSTRUCT property, SYSTEM CREATE method.

ORIGROWVALUE property

Description

Returns the original data as read into a form or control during a form's READ event.

Property Value

If the object is a data-bound form this property returns the "result row" as constructed during the READ event and used to populate the form's controls. This is not the same as the ATRECORD or RECORD properties and is not guaranteed to have the same structure.

If the object is a data-bound control this property returns the data that that was set in the control by the parent form's READ event.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

The underlying data used for this property is stored in the " OrigResultRow@" common variable in the form's "Window Common Area".

In previous versions of OpenInsight this property was named ORIG_ROWVALUE – this name is still supported for backwards compatibility.

Example

```
// Get the original data value set for the current control's in the READ event.

OrigData = Get_Property( CtrlEntID, "ORIGROWVALUE" )
```

See also

Common GUI COLUMN property, Common GUI TABLE property, WINDOW ATRECORD property, WINDOW RECORD property, WINDOW READ method, WINDOW READ event.

ORIGSIZE property

Description

Returns the original size attributes from the structure used to create the object at runtime.

Property Value

This property is a @Fm-delimited dynamic array containing the position values that were set in the Form Designer when the object was saved.

```
<1> Left (X)
<2> Top (Y)
<3> Width
<4> Height
```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_X$, PSPOS_Y$, PSPOS_WIDE$ and PSPOS_HIGH$ values from the ORIGSTRUCT property.

Note that for a WINDOW (Form) type, the width and height will always be negative values, to indicate to the Presentation Server that these are the desired *client size* values, rather than the normal size values (the latter always include the non-client areas values).

In previous versions of OpenInsight this property was named ORIG_SIZE – this name is still supported for backwards compatibility.

Example

```
// Get the Size used when the current control was created

OrigSize = Get_Property( CtrlEntID, "ORIGSIZE" )
```

See also

Common GUI ORIGSTRUCT property, Common GUI SIZE property, SYSTEM CREATE method.

ORIGSTRUCT property

Description

Returns the original definition structure (dynamic array) used to create an object at runtime.

Property Value

This property is an @Vm/@Svm/@Tm delimited dynamic array that defines the attributes of an object.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

The structure used to create an object is built by the Form compiler at design time. It can be updated and passed to the SYSTEM CREATE method to create a copy of that object at runtime.

Equated constants for use with the structure array are defined in the PS_EQUATES insert record. Type-specific constants are also supplied for each PS type in type specific insert records such as PS_LISTBOX_EQUATES, PS_EDITTABLE_EQUATES and so on.

In previous versions of OpenInsight this property was named ORIG_STRUCT – this name is still supported for backwards compatibility.

Example

```
// Get the structure used to create the current control

CtrlStruct = Get_Property( CtrlEntID, "ORIGSTRUCT" )
```

See also

SYSTEM CREATE method.

ORIGTEXT property

Description

Returns the original "Text" attribute from the structure used to create the object at runtime.

Property Value

This property is a string containing the text that was set in the Form Designer when the object was saved.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_TEXT$ value from the ORIGSTRUCT property.

In previous versions of OpenInsight this property was named ORIG_TEXT – this name is still supported for backwards compatibility.

Example

```
// Get the Text used when the current control was created

OrigText = Get_Property( CtrlEntID, "ORIGTEXT" )
```

See also

Common GUI ORIGSTRUCT property, Common GUI TEXT property, SYSTEM CREATE method.

ORIGVALUE property

Description

Returns the original "Value" attribute from the structure used to create the object at runtime.

Property Value

This property is a string containing the Value that was set in the Form Designer when the object was saved.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_VALUE$ value from the ORIGSTRUCT property.

Note that not all object types use or support a Value attribute. Please consult the documentation for the specific control types for more details.

In previous versions of OpenInsight this property was named ORIG_VALUE – this name is still supported for backwards compatibility.

Example

```
// Get the Value used when the current control was created

OrigValue = Get_Property( CtrlEntID, "ORIGVALUE" )
```

See also

Common GUI ORIGSTRUCT property, Common GUI VALUE property, SYSTEM CREATE method.

ORIGTOP property

Description

Returns the original Top coordinate used when the object was created at runtime.

Property Value

This property is an integer containing the Top coordinate that was set in the Form Designer when the object was saved.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_Y$ value from the ORIGSTRUCT property.

Example

```
// Get the Top coordinate used when the current control was created

OrigTop = Get_Property( CtrlEntID, "ORIGTOP" )
```

See also

Common GUI TOP property, Common GUI ORIGSTRUCT property, Common GUI ORIGY property, SYSTEM CREATE method.

ORIGVISIBLE property

Description

Returns the original "Visible" attribute from the structure used to create the object at runtime.

Property Value

This property is an integer containing the Visible value that was set in the Form Designer when the object was saved.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_VISIBLE$ value from the ORIGSTRUCT property.

In previous versions of OpenInsight this property was named ORIG_VISIBLE – this name is still supported for backwards compatibility.

Example

```
// Get the Visible value when the current control was created

OrigVisible = Get_Property( CtrlEntID, "ORIGVISIBLE" )
```

See also

Common GUI ORIGSTRUCT property, Common GUI VISIBLE property, SYSTEM CREATE method.

ORIGWIDE property

Description

Returns the original Width value used when the object was created at runtime.

Property Value

This property is an integer containing the Width coordinate that was set in the Form Designer when the object was saved.

If this is a negative value it refers to the desired *client-area* width only (e.g. "-400" means an object was created with a client area width of "400" DIPs). Negative values only apply to WINDOW (Form) objects.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property is a synonym for the ORIGWIDTH property.

In previous versions of OpenInsight this property was named ORIG_WIDE – this name is still supported for backwards compatibility.

Example

```
// Get the Width value used when the current control was created

OrigW = Get_Property( CtrlEntID, "ORIGWIDE" )
```

See also

Common GUI ORIGSIZE property, Common GUI ORIGSTRUCT property, Common GUI ORIGIWIDTH property, Common GUI WIDTH property, SYSTEM CREATE method.

ORIGWIDTH property

Description

Returns the original Width value used when the object was created at runtime.

Property Value

This property is an integer containing the Width coordinate that was set in the Form Designer when the object was saved.

If this is a negative value it refers to the desired *client-area* width only (e.g. "-400" means an object was created with a client area width of "400" DIPs). Negative values only apply to WINDOW (Form) objects.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property returns the PSPOS_WIDTH$ value from the ORIGSTRUCT property.

Example

```
// Get the Width value used when the current control was created

OrigW = Get_Property( CtrlEntID, "ORIGWIDTH" )
```

See also

Common GUI ORIGSIZE property, Common GUI ORIGSTRUCT property, Common GUI Common GUI ORIGWIDE property, Common GUI WIDTH property, SYSTEM CREATE method.

ORIGX property

Description

Returns the original Left coordinate used when the object was created at runtime.

Property Value

This property is an integer containing the Left coordinate that was set in the Form Designer when the object was saved.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property is a synonym for the ORIGLEFT property.

In previous versions of OpenInsight this property was named ORIG_X – this name is still supported for backwards compatibility.

Example

```
// Get the X pos used when the current control was created

OrigLeft = Get_Property( CtrlEntID, "ORIGX" )
```

See also

Common GUI LEFT property, Common GUI ORIGLEFT property, Common GUI ORIGSTRUCT property, SYSTEM CREATE method.

ORIGY property

Description

Returns the original Top coordinate used when the object was created at runtime.

Property Value

This property is an integer containing the Top coordinate that was set in the Form Designer when the object was saved.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

This property is a synonym for the ORIGTOP property.

In previous versions of OpenInsight this property was named ORIG_Y – this name is still supported for backwards compatibility.

Example

```
// Get the Y pos used when the current control was created

OrigTop = Get_Property( CtrlEntID, "ORIGY" )
```

See also

Common GUI TOP property, Common GUI ORIGTOP property, Common GUI ORIGSTRUCT property, SYSTEM CREATE method.

PAGENUMBER property

Description

Specifies the page number that the specified control will appear on in a multi-page "Container" parent object such as a panel or form.

Property Value

This property is an integer value specifying the page number. A value of "0" indicates that an object should occur on all pages.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

This property only applies to controls that are direct children of a multi-page object such as a form or a panel.

Example

```
// Ensure that the current control appear on page 2 of it's parent

Call Set_Property_Only( CtrlEntID, "PAGENUMBER", 2 )

// Ensure that the current control appears on all pages of it's parent

Call Set_Property_Only( CtrlEntID, "PAGENUMBER", 0 )
```

See also

Common GUI ALLPAGES property, Container CURRENTPAGE property, Container PAGECOUNT property, Container PAGECHANGED event, WINDOW PAGE event.

PARENT property

Description

Returns the parent of the specified object. A GUI object can have a parent object – if does then it is called a child object. A form that has no parent (i.e. the parent is the Windows desktop), is called a "top-level" form.

Property Value

This property is a string value that contains a valid, fully qualified, control name. Top-level forms will return a null value for this property.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

The PARENT property is implemented internally using the GetParent Windows API function, so please refer to the documentation on the Microsoft website for further information on parent and child relationships in Windows.

Example

```
// Get the PARENT object of the current control

CtrlParent = Get_Property( CtrlEntID, "PARENT" )

// Check if the current form is top-level

IsTopLevel = ( Get_Property( @Window, "PARENT" ) == "" )
```

See also

Common GUI PARENTFORM property, Common GUI GETPARENTFORM method, Common GUI SETPARENT method.

PARENTFORM property

Description

Returns the name of the parent form for the specified object.

Property Value

This property is a string value that contains a valid, fully qualified, control name.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

This property is the same as using the GETPARENTFORM method with a "Flags" parameter of "0".

If the specified object is a top-level form then the object itself is returned.

Example

```
// Get the PARENTFORM object of the current control

ParentForm = Get_Property( CtrlEntID, "PARENTFORM" )

// Check if the current form is top-level

IsTopLevel = ( Get_Property( @Window, "PARENTFORM" ) == @Window )
```

See also

Common GUI PARENT property, WINDOW MDIFRAME property, Common GUI GETPARENTFORM method, Common GUI SETPARENT method

PART property

Description

If the object is data-bound to a key column, then this property specifies the part of the key that it is bound to.

Property Value

This property is an integer value. It returns "0" if the control is not bound to a key column, or if the key column is not multipart.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

N/a.

Example

```
// Get the key part that EDL_REFNO is bound to

KeyPart = Get_Property( @Window : ".EDL_REFNO", "PART" )
```

See also

Common GUI COLUMN property, Common GUI POS property, Common GUI TABLE property, WINDOW ID property.

POS property

Description

Returns the position of the column, relative to the data table structure, of a data-bound control.

Property Value

This property is an integer value. It returns "0" if the control is bound to a key column.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

N/a.

Example

```
// Get a list of all key controls on a form

CtrlMap = Get_Property( @Window, "CTRLMAP" )

Convert @Fm To @Rm In CtrlMap
PosList = Get_Property( CtrlMap, "POS" )

KeyCtrlIDs   = ""
KeyCtrlParts = ""

CtrlIdx = 1
PosIdx  = 1

Loop
   CtrlID   = CtrlMap[CtrlIdx,@Rm,TRUE$]
   CtrlIdx  = BCol2()+1

   CtrlPos  = PosList[PosIdx,@Rm,TRUE$]
   PosIdx   = BCol2()+1

   If ( CtrlPos == 0 ) Then
      // It's a key
      KeyCtrlIDs   := CtrlID                          : @Fm
      KeyCtrlParts := Get_Property( CtrlID, "PART" ) : @Fm
   End
While ( CtrlIdx < BLen( CtrlMap ) )
Repeat

KeyCtrlIDs[-1,1]   = ""
KeyCtrlParts[-1,1] = ""
```

See also

Common GUI COLUMN property, Common GUI PART property, Common GUI TABLE property, WINDOW ATRECORD property, WINDOW ID property, WINDOW RECORD property.

PREVIOUS property

Description

Specifies the previous control in the tab order from the specified control.

Property Value

This property is a string value that should contain a valid, fully qualified, control name.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

This property can be used to dynamically change the tab order of control on a form at runtime.

Example

```
// Skip over the next control in the tab order to the
// one after it

NextCtrl         = Get_Property( CtrlEntid, "NEXT" )
TheCtrlAfterNext = Get_Property( NextCtrl, "NEXT" )

Call Set_Property_Only( CtrlEntId, "NEXT", TheCtrlAfterNext )

// Now update that control so "tabbing back" from it goes to
// this control

Call Set_Property_Only( TheCtrlAfterNext, "PREVIOUS", CtrlEntID )
```

See also

Common GUI NEXT property.

QUALIFIEDWINMSGS property

Description

Returns a list of Window Messages that trigger a WINMSG event for an object.

Property Value

This property is an @Fm-delimited list of Windows messages that have been qualified to trigger a WINMSG event.

Each item in the list has the following structure:

```
<1> Message number
<2> Qualifier string
<3> Event name
<4> SyncFlags
```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

N/a.

Example

```
// Get a list of all qualified window messages for the current
// control

WinMsgList = Get_Property( CtrlEntID, "QUALFIIEDWINMSGS" )
```

See also

Common GUI QUALIFYWINMSG method, Common GUI WINMSG event.

RECT property

Description

Specifies the position and size of a control relative to its parent control using client-area coordinates. (For a top-level form the coordinates are relative to the Windows desktop).

Property Value

For both getting and setting the RECT property, the value is an @Fm-delimited array of integer coordinates:

```
<1> Left
<2> Top
<3> Right
<4> Bottom
```

When setting the RECT property, the following optional members may be applied:

```
<5> Visible flag (-1 to keep an invisible object hidden)
<6> NoSendChange flag (If TRUE$ do not send WM_POSCHANGING)
```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	Yes	No

Remarks

A coordinate may use a value of " 32768" to be left unchanged when setting the RECT property.

By default updating an object via the RECT property will make it visible if it is hidden. Setting the "Visible flag" field to "-1" prevents this.

When container objects (e.g. Forms and Panels) are resized, a SIZE event is normally raised and any child objects informed of the change. This can be prevented by setting the " NoSendChange flag" to TRUE$ which prevents Windows from sending a WM_POSCHANGING message, thereby blocking any notifications.

The RECT property is implemented internally using the SetWindowPos Windows API function, so please refer to the documentation on the Microsoft website for further information on repositioning forms and controls in Windows.

Example

```
// Get the RECT of the current control and move it's Left coordinate 20 DIPs to the
// right, decreasing the width of the control by the same amount.
//
// If we were using the SIZE property we would have to update both the Left and the
// Width coordinates to do this, e.g.:

CtrlSize = Get_Property( CtrlEntID, "SIZE" )

CtrlSize  <1> = CtrlSize  <1> + 20
CtrlSize  <3> = CtrlSize  <3> - 20

Call Set_Property_Only( CtrlEntID, "SIZE", CtrlSize )

// However, with the RECT property we only have to adjust the Left coordinate as the
// Right coordinate stays the same.

CtrlRect = Get_Property( CtrlEntID, "RECT" )

CtrlRect  <1> = CtrlRect  <1> + 20

Call Set_Property_Only( CtrlEntID, "RECT", CtrlRect )
```

See also

Common GUI BOTTOM property, Common GUI CLIENTSIZE property, Common GUI HEIGHT property, Common GUI LEFT property, Common GUI RIGHT property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, Common GUI TOP property, Common GUI WIDTH property, WINDOW SCALEUNITS property, WINDOW SIZE event, Appendix K – High-DPI Programming.

REDRAW property

Description

Specifies if a control is repainted when it is updated,

Property Value

This property is a boolean value. If FALSE$ then any updates to the control that would cause it to be repainted are ignored. The default value is TRUE$.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

Setting the REDRAW property of a parent object to FALSE$ while manipulating its child objects is a common way to increase rendering performance and reducing flicker, as all visual changes will be applied in one operation when the REDRAW property is reset to TRUE$.

When redrawing is turned off Windows removes the WS_VISIBLE style bit from the object but doesn't invalidate it, so it doesn't know that it needs repainting and still appears on the screen. However, because of this it will report that *it is no longer visible*, so always check an object's visibility *before* redrawing is turned off.

It is always best practice to check the state of the REDRAW property before using it, in case an earlier process has turned it off during any recursive programming.

Example

```
// Check if the CHK_SHOWITEMS control is visible - if not we don't want
// to make the EDB_ITEMDETAILS control visible
ShowItemsVis = Get_Property( @Window : ".CHK_SHOWITEMS", "VISIBLE" )

// Save the current state of the REDRAW flag before we turn it off
IsRedraw = Set_Property( @Window, "REDRAW", FALSE$ )

Call Set_Property( @Window : ".CHK_SHOWSTUFF", "VISIBLE", FALSE$ )
Call Set_Property( @WIndow : ".BTN_DELETE", "VISIBLE", FALSE$ )
Call Set_Property( @Window : ".EDB_ITEMDETAILS", "VISIBLE", ShowItemsVis )

// If the redraw as on originally then turn it back on
If IsRedraw Then
   Call Set_Property_Only( @Window, "REDRAW", TRUE$ )
End
```

See also

Common GUI VISIBLE property, Common GUI INVALIDATE method, Common GUI REPAINT method.

REQUIRED property

Description

Specifies if a control is required to contain data.

Property Value

This property is a boolean value. When set to TRUE$ then the user is not allowed to move to another control unless the current control contains data unless the parent form's REQUIREONSAVE property is set (see Remarks below).

For controls that support associated multivalued data, like the EditTable control, this property will return an @Svm-delimited list of boolean values at runtime (one for each column in the control).

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	Yes

Remarks

By default the check to see if the control has data is performed during the LOSTFOCUS event (and POSCHANGED for EditTables) – if the check is failed a message is displayed to the user and the input focus is returned to the control (this is handled by the REQUIRERR event). This behavior can be too intrusive for many applications and may be modified by using the parent Form's REQUIREONSAVE property, where the check is only made when the data is about to be saved.

This property is normally associated with data-bound controls.

Example

```
// Set the REQUIRED flag for the current control

PrevRequired = Set_Property( CtrlEntID, "REQUIRED", TRUE$ )
```

See also

Common GUI VALID property, WINDOW REQUIREONSAVE property, Common GUI LOSTFOCUS event, Common GUI REQUIRERR event, EDITTABLE POSCHANGED event.

RIGHT property

Description

Specifies the right coordinate of an object relative to its parent.

Property Value

This property is an integer value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	Yes	No

Remarks

Setting this property adjusts the width of the object, it does not affect the LEFT property.

Coordinates are always relative to the top left corner of the screen, or, for a child window, the upper left corner of the parent window's client area.

Example

```
// Set the RIGHT of the EDB_NOTES EditBox control to position 300 DIPs

Call Set_Property_Only( @Window : ".EDB_NOTES", "RIGHT", 300 )
```

See also

Common GUI AUTOSIZEWIDTH property, Common GUI CLIENTSIZE property, Common GUI CLIENTWIDTH property, Common GUI RECT property, Common GUI RIGHTANCHOR property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, Common GUI WIDTH property, WINDOW SCALEUNITS property, WINDOW SIZE event, Appendix K – High-DPI Programming.

RIGHTANCHOR property

Description

Specifies if the right coordinate of a control maintains the same distance from right side of its parent object (Form or Panel) when the latter is resized.

For example, if the parent Form's width is increased by 60 pixels, a control with RIGHTANCHOR set to TRUE$ will be moved across by 60 pixels as well.

Property Value

This property is a boolean value. When Set to TRUE$ changing the control's parent width will move the control left or right by the same amount. When set to FALSE$ the control's position is not changed when the parent width is changed.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

If this property is set to TRUE$ the AUTOSIZEWIDTH property is automatically set to FALSE$.

The RIGHTANCHOR property is preserved and updated as necessary if the object is moved or resized.

In previous versions of OpenInsight this property was named ANCHORRIGHT – this name is still supported for backwards compatibility.

Example

```
// Set the RIGHTANCHOR of the EDB_NOTES EditBox control

Call Set_Property_Only( @Window : ".EDB_NOTES", "RIGHTANCHOR", TRUE$ )
```

See also

Common GUI AUTOSIZEWIDTH property, Common GUI CLIENTWIDTH property, Common GUI CLIENTSIZE property, Common GUI RECT property, Common GUI RIGHT property, Common GUI WIDTH property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, Common GUI MOVE method, Common GUI OFFSET method, WINDOW SIZE event.

SCALEFACTOR property

Description

Returns the scale-factor value for the specified object.

Property Value

This property is a numeric value representing the current scale-factor or "magnification" setting for the object.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

The scale-factor is set by the top-level parent form that owns the object. It is combined with the form's DPI settings to give an overall magnification value that is applied when objects are positioned and drawn.

Example

```
// Get the scale factor for the EDL_SURNAME control

ScaleFactor = Get_Property( @Window : ".EDL_SURNAME", "SCALEFACTOR" )
```

See also

Common GUI DPI property, Common GUI SCALEMETRICS property, Common GUI SCALEUNITS property, WINDOW SCALEFACTOR property, WINDOW SCALED event, Appendix K – High-DPI Programming.

SCALEMETRICS property

Description

Returns an array of scaling information for the specified object.

Property Value

This property is an @Fm-delimied array of scaling information:

```
<1> DPI X value
<2> DPI Y value
<3> ScaleFactor value
<4> ScaleUnits value
```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

This property is basically an amalgamation of the DPI, SCALEFACTOR and SCALEUNITS properties, and is intended as an optimization – one property call instead of three.

Equated constants for use with the SCALEMETRICS property can be found in the PS_EQUATES insert record.

Example

```
// Get the scale metrics for the EDL_SURNAME control
$Insert PS_Equates

ScaleMetrics = Get_Property( @Window : ".EDL_SURNAME", "SCALEMETRICS" )

ScaleFactor  = ScaleMetrics<PS_SCM_SCALEFACTOR$>
```

See also

Common GUI DPI property, Common GUI SCALEFACTOR property, Common GUI SCALEUNITS property, WINDOW SCALEFACTOR property, WINDOW SCALED event, Appendix K – High-DPI Programming.

SCALEUNITS property

Description

Returns the scale units value for the specified object. The scale units are a setting that determines how coordinates used in properties, methods events are interpreted – either as DIPs (Device Independent Pixels) or actual pixels

Property Value

This property is a numeric value representing the current scale units used for getting and setting scaled properties for the object. It can be one of the following values:

Value	*Description*
0	Use DIPs (the default).
1	Use pixels.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

The scale-units are set by the top-level parent form that owns the object.

Equated constants for use with the SCALEUNITS property can be found in the PS_EQUATES insert record.

Example

```
// Get the scale units for the EDL_SURNAME control

ScaleUnits = Get_Property( @Window : ".EDL_SURNAME", "SCALEUINTS" )
```

See also

All properties marked as "Scaled", WINDOW SCALEUNITS property, Appendix K – High-DPI Programming.

SCREENRECT property

Description

Specifies the position of an object in *screen* coordinates, i.e. relative to the primary monitor desktop area.

Property Value

The SCREENRECT property is an @Fm-delimited array of integer coordinates:

```
<1> Left
<2> Top
<3> Right
<4> Bottom
```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	Yes	No

Remarks

The SCREENRECT property is implemented internally using the GetWindowRect Windows API function, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Get the SCREENRECT of the current window

ScreenRect = Get_Property( @Window, "SCREENRECT" )
```

See also

Common GUI RECT property, Common GUI SCREENSIZE property, Common GUI SCALEUNITS property, Appendix K – High-DPI Programming.

SCREENSIZE property

Description

Specifies the position and size of an object in *screen* coordinates, i.e. relative to the primary monitor desktop area.

Property Value

The SCREENSIZE property is an @Fm-delimited array of integer coordinates and dimensions:

```
<1> Left
<2> Top
<3> Width
<4> Height
```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	Yes	No

Remarks

The SCREENSIZE property is implemented internally using the GetWindowRect Windows API function, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Get the SCREENSIZE of the current window

ScreenSize = Get_Property( @Window, "SCREENSIZE" )
```

See also

Common GUI SCREENRECT property, Common GUI SIZE property, SCALEUNITS property, Appendix K – High-DPI Programming.

SCROLLBARS property

Description

Specifies which scrollbars are used with an object that supports them.

Property Value

This property is a numeric value representing which scrollbars are visible. It can be one of the following values:

Value	*Description*
0	No scrollbars (the default).
1	Horizontal only.
2	Vertical only.
3	Both horizontal and vertical.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

Not all objects support scrollbars, and of those do some may not support all options. Please refer to the documentation for the individual object type for any differences.

Equated constants for use with the SCROLLBARS property can be found in the PS_EQUATES insert record.

Example

```
// Only show the vertical scrollbar for the EDB_NOTES control
$Insert PS_Equates

Call Set_Property_Only( @Window : ".EDB_NOTES", "SCROLLBARS", PS_SB_VERTICAL$ )
```

See also

Common GUI HSCROLL event, Common GUI VSCROLL event.

SIZE property

Description

Specifies the position and size of a control relative to its parent control using client-area coordinates. (For a top-level form, the coordinates are relative to the Windows desktop).

Property Value

For both getting and setting the SIZE property the value is an @Fm-delimited array of integer coordinates:

```
<1> Left
<2> Top
<3> Width
<4> Height
```

When setting the SIZE property, the following optional members may be applied:

```
<5> Visible flag (-1 to keep an invisible object hidden)
<6> NoSendChange flag (If TRUE$ do not send WM_POSCHANGING)
```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	Yes	No

Remarks

A coordinate may use a value of " 32768" to be left unchanged when setting the SIZE.

By default updating an object via the SIZE property will make it visible if it is hidden. Setting the "Visible flag" field to "-1" prevents this.

When container objects (e.g. Forms and Panels) are resized, a SIZE event is normally raised and any child objects are also informed of the change. This can be prevented by setting the " NoSendChange flag" to TRUE$ which prevents Windows from sending a WM_POSCHANGING message, thereby blocking any notifications.

The SIZE property is implemented internally using the SetWindowPos Windows API function, so please refer to the documentation on the Microsoft website for further information on repositioning forms and controls in Windows.

Example

```
// Get the SIZE of the current form and increase the width by 100 DIPs
// without sending a change notification

WinSize = Get_Property( @Window, "SIZE" )

WinSize<3> = WinSize<3> + 100
WinSize<6> = TRUE$

Call Set_Property_Only( @Window, "SIZE", WinSize )
```

See also

Common GUI BOTTOM property, Common GUI CLIENTSIZE property, Common GUI HEIGHT property, Common GUI LEFT property, Common GUI RIGHT property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, Common GUI TOP property, Common GUI WIDTH property, WINDOW SCALEUNITS property, WINDOW SIZE event, Appendix K – High-DPI Programming.

STYLE property

Description

Gets or sets the "Windows Style" value for a specified object.

Every GUI object in Windows has a set of numeric bit flags that control their behaviour and attributes, and in OpenInsight these are usually exposed as properties. Windows combines these flags (via a bitwise "OR" operation) into a single 32-bit integer value that is called the "Windows Style". At runtime this value can tested against a known bitmask value to see if a flag (i.e. a bit) is set.

Property Value

This property is a string value containing a C-style hexadecimal representation of an unsigned 32-bit integer, i.e. a hex number prefixed by "0X" (This representation generally makes it easier to see which flags are set without needing to use any programs).

e. g.

> The style value 2496135168 is stored as "0X94C80000" in the STYLE property, and from this we can easily see that the following flags are set:
>
> ```
> WS_POPUP        (0x80000000)
> WS_VISIBLE      (0x10000000)
> WS_CLIPSIBLINGS (0x04000000)
> WS_CAPTION      (0x00C00000)
> WS_SYSMENU      (0x00080000)
> ```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

Direct manipulation of Window Styles via this property is not recommended as those flags that can be altered safely are already exposed as object properties (several flags cannot be altered after the object has been created and some attempts may even cause an application to crash).

Equated constants for use with the STYLE property can be found in the MSWIN_WINDOWSTYLE_EQUATES insert record. Type-specific constants can be found in other inserts such a MSWIN_EDIT_EQUATES, MSWIN_LISTBOX_EQUATES and so on.

Further information on Windows Styles can be found om the Microsoft website.

Example

```
// Get the STYLE property of the current form and check to see if
// it has a caption bar
$Insert MSWin_WindowStyle_Equates

WinStyle = Get_Property( @Window, "STYLE" )

// It's in C-hex format so remove the "0X" prefix and convert
// to a decimal integer

WinStyle = Iconv( WinStyle[3,\00\], "MX" )

// Now we can test the style to see if the WS_CAPTION$ bit is
// set

If BitAnd( WinStyle, WS_CAPTION$ ) Then
   // Window has a caption bar …
End
```

See also

Common GUI STYLEEX property, Common GUI STYLEEXN property , Common GUI STYLEN property.

STYLEEX property

Description

Gets or sets the "Windows Extended Style" value for a specified object.

As GUI objects in the system gained more functionality Microsoft found that a 32-bit integer was not enough to store all of the different attributes, so they added a second set of bit flags called the Windows Extended Style.

Property Value

This property is a string value containing a C-style hexadecimal representation of an unsigned 32-bit integer, i.e. a hex number prefixed by "0X". See the STYLE property for more information on how this format is used.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

This property behaves in the same manner as the STYLE property except that it works with a different set of flags.

Direct manipulation of Window Extended Styles via this property is not recommended as those flags that can be altered safely are already exposed as object properties (several flags cannot be altered after the object has been created and some attempts may even cause an application to crash).

In previous versions of OpenInsight this property was named STYLE_EX – this name is still supported for backwards compatibility.

Equated constants for use with the STYLEEX property can be found in the MSWIN_WINDOWSTYLE_EQUATES insert record. Type-specific constants can be found in other inserts such a MSWIN_EDIT_EQUATES, MSWIN_LISTBOX_EQUATES and so on.

Further information on Windows Extended Styles can be found on the Microsoft website.

Example

```
// Get the STYLEEX property of the EDB_NOTES control and check to see if
// it has a "client edge"
$Insert MSWin_WindowStyle_Equates

WinStyleEx = Get_Property( @Window : ".EDB_NOTES", "STYLEEX" )

// It's in C-hex format so remove the "0X" prefix and convert
// to a decimal integer

WinStyleEx = Iconv( WinStyleEx[3,\00\], "MX" )

// Now we can test the style to see if the WS_EX_CLIENTEDGE$ bit is
// set

If BitAnd( WinStyleEx, WS_EX_CLIENTEDGE$ ) Then
   // Control has a sunken edge
End
```

See also

Common GUI STYLE property, Common GUI STYLEEXN property, Common GUI STYLEN property.

STYLEEXN property

Description

Gets or sets the "Windows Extended Style" value for a specified object using a decimal integer format rather than the default C-hexadecimal format.

Property Value

This property is an unsigned integer value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

This property behaves in the same manner as the normal STYLEEX property except that the returned value is a decimal integer, making it easier to use with numeric functions without any conversion – e.g. compare the example code below with the example from the STYLEEX property.

Example

```
// Get the STYLEEX property of the EDB_NOTES control and check to see if
// it has a "client edge"
$Insert MSWin_WindowStyle_Equates

WinStyleEx = Get_Property( @Window : ".EDB_NOTES", "STYLEEX" )

// Now we can test the style to see if the WS_EX_CLIENTEDGE$ bit is
// set

If BitAnd( WinStyleEx, WS_EX_CLIENTEDGE$ ) Then
   // Control has a sunken edge
End
```

See also

Common GUI STYLE property, Common GUI STYLEEX property, Common GUI STYLEN property.

STYLEN property

Description

Gets or sets the "Windows Style" value for a specified object using a decimal integer format rather than the default C-hexadecimal format.

Property Value

This property is an unsigned integer value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

This property behaves in the same manner as the normal STYLE property except that the returned value is a decimal integer, making it easier to use with numeric functions without any conversion – e.g. compare the example code below with the example from the STYLE property.

Example

```
// Get the STYLE property of the current form and check to see if
// it has a caption bar
$Insert MSWin_WindowStyle_Equates

WinStyle = Get_Property( @Window, "STYLEN" )

// Now we can test the style to see if the WS_CAPTION$ bit is
// set

If BitAnd( WinStyle, WS_CAPTION$ ) Then
   // Window has a caption bar …
End
```

See also

Common GUI STYLE property, Common GUI STYLEEX property, Common GUI STYLEEXN property.

TABLE property

Description

If the object is data-bound then this property specifies the database table that it is bound to.

Property Value

This property is a string value and must be a valid database table name.

For controls that support associated multivalued data, like the EditTable control, this property will return an @Svm-delimited list of bound data tables at runtime.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	Yes

Remarks

N/a.

Example

```
// Get the database table name of the EDL_SURNAME control

TableName = Get_Property( @Window : ".EDL_SURNAME", "TABLE" )
```

See also

Common GUI COLUMN property, Common GUI PART property, Common GUI POS property, Common GUI MV property, WINDOW ATRECORD property, WINDOW ID property, WINDOW RECORD property, Common GUI CALCULATE event.

TOP property

Description

Specifies the top coordinate of an object relative to its parent object.

Property Value

This property is an integer value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	Yes	No

Remarks

Setting this property adjusts the top coordinate of the object, it does not affect the height so the object is moved, not resized.

Coordinates are always relative to the top left corner of the screen, or, for a child window, the upper left corner of the parent window's client area.

Example

```
// Set the TOP coordinate of the EDB_NOTES EditBox control to position 10

Call Set_Property_Only( @Window : ".EDB_NOTES", "TOP", 10 )
```

See also

Common GUI AUTOSIZEHEIGHT property, Common GUI BOTTOMANCHOR property, Common GUI CLIENTHEIGHT property, Common GUI CLIENTSIZE property, Common GUI HEIGHT property, Common GUI RECT property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, Common GUI MOVE method, Common GUI OFFSET method, WINDOW SCALEUNITS property, WINDOW SIZE event, Appendix K – High-DPI Programming.

TEXT property

Description

Specifies the text associated with the object.

Property Value

This property is a string value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

The TEXT property of an object is used differently by different types of object, as can be seen in this summary table:

Control Type	*TEXT property description*
Bitmap	BITMAP
CheckBox	The Checkbox label.
ColorDropDown	The contents of the EditLine.
ComboBox	The contents of the EditLine.
DateTime	The date displayed in the control.
EditLine	The contents of the control.
EditBox	The contents of the control.
EditTable	The contents of a cell.
GroupBox	The text displayed in the border of the box.
GroupBoxEx	The text displayed in the border of the box.
Hyperlink	The link text displayed in the control.
ListBox	The text of the currently selected item.
Panel	The text displayed in the control.
ProgressBar	The progress text displayed in the control.
PushButton	The text on the button.
RadioButton	The label for the button.
RichEditBox	The contents of the control.
Static	The text displayed in the control.
TabControl	The text for the active tab.
Window (Form)	The text in the form's title bar.

This list is not exhaustive, so details on the behavior the TEXT property can be found in the documentation for each object type (if supported).

The TEXT property will be affected the SYSTEM CHARMAP property if the latter is set.

For more information on this property please refer to the Windows documentation regarding the GetWindowText and SetWindowText API functions on the Microsoft website.

Example

```
// Update the TEXT of the current form …

Call Set_Property_Only( @Window, "TEXT", "New Entry" )
```

See also

Common GUI DEFPROP property, SYSTEM CHARMAP property.

TIMER property

Description

Starts or stops the generation of TIMER events for an object.

Property Value

This property is an @Fm-delimited array of two numeric values:

```
<1> Delay in milliseconds between each TIMER event
<2> Initial delay (in milliseconds) before the first TIMER event fires
```

The initial delay is optional and defaults to Delay value in field <1>

To stop TIMER events, set a value of "0".

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

In previous versions of OpenInsight only forms supported a TIMER property and a TIMER event. In this version all GUI objects support them.

For more information on Windows Timers please refer to the documentation regarding the SetTimer and KIllTimer functions, and the WM_TIMER message on the Microsoft Website.

Example

```
// Execute the TIMER event once and only once after n millseconds

Call Set_Property_Only( CtrlEntID, "TIMER", 0 : @Fm : n )

// Execute the TIMER event every n millseconds starting after n
// millseconds

Call Set_Property_Only( CtrlEntID, "TIMER", n )

// Execute the TIMER event every n millseconds starting immediately

Call Set_Property_Only( CtrlEntID, "TIMER", n : @Fm : 0 )

// Stop generating TIMER events

Call Set_Property_Only( CtrlEntID, "TIMER", 0 )
```

See also

SYSTEM IDLEPROC property, SYSTEM ADDIDLEPROC method, Common GUI TIMER event.

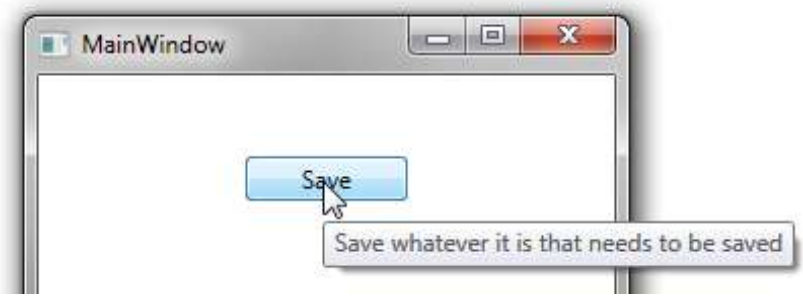
TOOLTIP property

Description

Specifies the tooltip to display for an object.

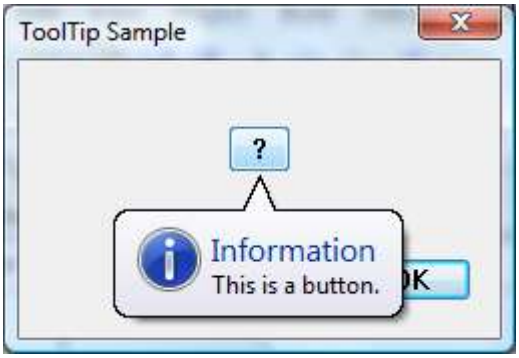
A tooltip is a small message that appears over an object when a user hovers over it with the mouse. It is to offer hints on how the object is used. There are several options available that control the appearance of a tooltip as can be seen in the examples below:

Simple Tooltip:



Balloon Tooltip (rounded corners):

Centered Balloon Tooltip with Title and Large Icon:



Property Value

This property is an @Fm-delimited array structured as follows:

```
<1> Text (multiple lines may be @Tm-delimited)
<2> Maximum width (optional)
<3> Title
<4> Icon ("*", "!", "H", or filename)
<5> Large Icon flag (TRUE$/FALSE$)
<6> Balloon style flag (TRUE$/FALSE$)
<7> Centered style (TRUE$/FALSE$)
```

Only the text (field <1>) is required for a tooltip to be displayed. The other fields are optional.

A tooltip may display an icon if a Title is defined as well. A standard system icon may be used by specifying one of the characters below:

"*"	Information icon
"!"	Warning icon
"H"	Error icon

A filename can also be specified in the same manner as the WINDOW ICON property.

Icons may be displayed in a "Large" format (32x32 pixels), or in a "Small" format (16x16 pixels). Use the boolean "Large Icon" flag in field <5> to specify this.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	Yes	No

Remarks

Some objects, such as ListBoxes and PropertyGrids have other options for displaying tooltips, so they automatically show the contents of an item.

Equated constants for use with tooltip programming can be found in the PS_TOOLTIP_EQUATES insert record.

Further information on ToolTips can be found in the Windows documentation on the Microsoft website.

Example

```
$Insert PS_ToolTip_Equates

// Display a simple tooltip for the current control

TTInfo = ""
TTInfo<PS_TOOLTIP_POS_TEXT$> = "Click here to do stuff"

Call Set_Property_Only( CtrlEntID, "TOOLTIP", TTInfo )

// Display a multiline balloon tooltip for the current control

ToolText =        "Here's a list of the things"
ToolText := @Tm : ""
ToolText := @Tm : "Item One"
ToolText := @Tm : "Item Two"

TTInfo = ""
TTInfo<PS_TOOLTIP_POS_TEXT$>    = ToolText
TTInfo<PS_TOOLTIP_POS_BALLOON$> = TRUE$

Call Set_Property_Only( CtrlEntID, "TOOLTIP", TTInfo )

// Display a centered balloon tooltip for the current control
// with a title and a large icon

TTInfo = ""
TTInfo<PS_TOOLTIP_POS_TEXT$>      = "Don't click this button. Ever."
TTInfo<PS_TOOLTIP_POS_TITLE$>     = "Danger Danger Danger"
TTInfo<PS_TOOLTIP_POS_ICON$>      = "!" ; // Warning
TTInfo<PS_TOOLTIP_POS_LARGEICON$> = TRUE$
TTInfo<PS_TOOLTIP_POS_BALLOON$>   = TRUE$
TTInfo<PS_TOOLTIP_POS_CENTERED$>  = TRUE$

Call Set_Property_Only( CtrlEntID, "TOOLTIP", TTInfo )
```

See also

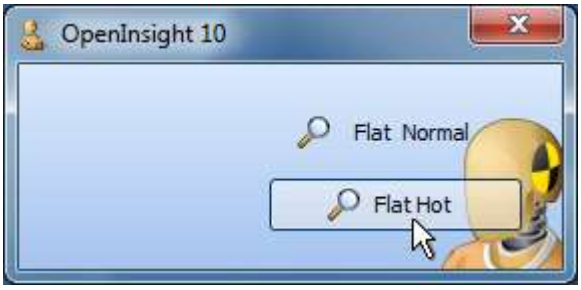
COMBOBOX SHOWITEMTOOLTIPS property, LISTBOX SHOWITEMTOOLTIPS property, PROPERTYGRID SHOWTOOLTIPS property, TABCONTROL SHOWTOOLTIPS property, WINDOW ICON property, Msg stored procedure, Appendix J – System Icons.

TRANSLUCENCY property

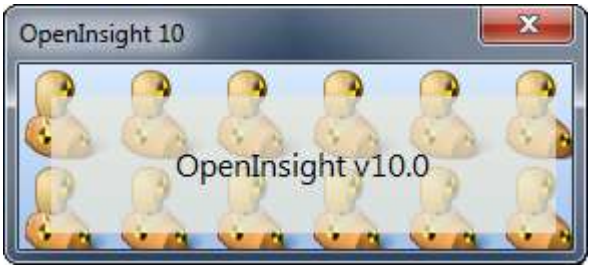
Description

Specifies the degree of transparency applied to an object's background when it is painted. Note that the transparency effect will not apply to an object's non-client area borders, nor to text and glyphs, though it will affect a background image.

Flat PushButton control with 50% TRANSLUCENCY applied:



Static control with 60% TRANSLUCENCY applied:



Property Value

This property is an integer value between 1 and 100, which represents the percentage of transparency applied to the background. A value of 0 means fully opaque, while a value of 100 means fully transparent (i.e. the background will not be drawn).

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

Be aware that the use of translucency does involve extra overhead when drawing controls because there are the extra steps of painting in the parent and then blending it with the control's background – while the PS attempts to mitigate this using cached bitmaps and double-buffering there will always be *some* impact.

Example

```
// Set the TRANSLUCENCY of EDB_NOTES to 60%

PrevVal = Set_Property( @Window : ".EDB_NOTES", "TRANSLUCENCY", 60 )

// Remove the TRANSLUCENCY from EDB_NOTES

PrevVal = Set_Property( @Window : ".EDB_NOTES", "TRANSLUCENCY", 0 )

 // Remove the EDB_NOTES background

PrevVal = Set_Property( @Window : ".EDB_NOTES", "TRANSLUCENCY", 100 )
```

See also

Common GUI BACKCOLOR property, Common GUI VISIBLE property, WINDOW TRANSLUCENCY property.

VALID property

Description

Specifies the Validation Input pattern for a control. This pattern is used to verify user-inputted data and convert it to an "internal format" for storage and processing.

For example, dates in OpenInsight are held internally as simple integer values – so an input conversion needs to be applied to a date string that a user would enter like "01/03/2020".

Property Value

This property is a string value and must be one of the following:

- A null string (no validation or conversion applied).
- One or more OpenInsight input conversion patterns such as "(DE)".
    - Multiple patterns are supported if separated by a delimiter:
        - To OR multiple patterns together use an @Vm delimiter.
        - To AND multiple patterns together use an "_" delimiter.

At design time the following special strings may also be used:

- "<<None>>" – same as a null string, i.e. no conversion applied.
- "<<Default>>" – If the control is data-bound then the database column's input validation pattern is used, otherwise this is treated as a null string.

For controls that support associated multivalued data, like the EditTable control, this property will return an @Svm-delimited list of patterns at runtime (one for each column in the control).

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	Yes

Remarks

By default the check to see if the control passes a validation check is performed during the LOSTFOCUS event (and the POSCHANGED event for EditTables) – if the check is failed a message is displayed to the user and the input focus is returned to the control (this is handled by the VALIDERR event).

This property is normally associated with data-bound controls.

Example

```
// Set the validation pattern for the EDL_DOB control check for a
// European date format:
//
//    "DE" (dd/mm/yyyy)
//
// (you know, the date format that makes sense ;).

PrevVal = Set_Property( @Window : ".EDL_DOB", "VALID", "DE" )

// Set the validation pattern for the EDL_TEST control check for a range
// of numeric values between 0 and 99, OR between 300 and 399

NewVal = "(0,99)" : @Vm : "(300,399)"

PrevVal = Set_Property( @Window : ".EDL_TEST", "VALID", NewVal )

// Set the validation pattern for the EDL_TEST control to verify that
// the data entered is a valid european date (DE) AND passes the checks
// in the AGE_CHECK stored procedure

NewVal = "(DE)_[AGE_CHECK]"

PrevVal = Set_Property( @Window : ".EDL_TEST", "VALID", NewVal )
```

See also

Common GUI CONV property, Common GUI REQUIRED property, Common GUI VALIDMSG property, Common GUI REQUIRERR event, Common GUI VALIDERR event.

VALIDMSG property

Description

Specifies alternative text for a validation message.

When an object fails a validation check the system displays an error message with some default text such as:

```
The input "%1%" does not pass the validation criteria "%2%"
```

The VALIDMSG property can be set and used instead of the default text if desired.

Property Value

This property is a string. Multiple lines may be delimited by "|". It can also contain the following tokens which are replaced at runtime with the appropriate data:

```
%1%    - Replaced with the input data that failed the check
%2%    - Replaced with the pattern used for the check
%3%    - Replaced with the name of the object that failed the check
```

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	Yes

Remarks

The default text for validation messages is stored in the SYSENV TXT_VALIDATION record, and the "UI146" resource string in the SYSENV SYSTEM_RESOURCES record.

Example

```
// Set a custom validation message for the current control

valText  = 'Are you serious? Really?||'
valText := 'How on earth is "%1%" ever going to pass a "%2%" check eh?||'
valText := "Go back to %3% and try again, and this time get it right!"

Call Set_Property_Only( CtrlEntID, "VALIDMSG", valText )
```

See also

Common GUI VALID property, Common GUI VALIDERR event.

VISIBLE property

Description

Specifies if a control is visible or not.

Property Value

The VISIBLE property is an integer value that specifies how the control is displayed. For a standard control it can be one of the following values:

Value	Description
0	The control is hidden.
1	The control is visible.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	No	No

Remarks

The VISIBLE property is implemented internally using the ShowWindow Windows API function and the property value actually corresponds to the function's nCmdShow parameter values. For most controls only the SW_SHOW (1) and SW_HIDE(0) values apply, while other types such as WINDOW (Form) objects support more.

Constants for these values are defined in the MSWIN_SHOWWINDOW_EQUATES insert record.

Example

```
$Insert MsWin_ShowWindow_Equates

// Example - Hiding a control
Call Set_Property_Only( ctrlEntID, "VISIBLE", SW_HIDE$ )
```

See also

WINDOW VISIBLE property.

WIDTH property

Description

Specifies the width of an object.

Property Value

This property is an integer value.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
Get/Set	Get/Set	No	Yes	No

Remarks

The WIDTH property includes an object's non-client area as well the client area.

Example

```
// Set the WIDTH of the EDB_NOTES EditBox control to position 360 DIPs

Call Set_Property_Only( @Window : ".EDB_NOTES", "WIDTH", 360 )
```

See also

Common GUI AUTOSIZEWIDTH property, Common GUI CLIENTSIZE property, Common GUI CLIENTWIDTH property, Common GUI RECT property, Common GUI RIGHTANCHOR property, Common GUI SCREENRECT property, Common GUI SCREENSIZE property, Common GUI SIZE property, WINDOW SCALEUNITS property, WINDOW SIZE event, Appendix K – High-DPI Programming.

Common GUI Methods

These methods apply to most GUI forms and controls except where noted in individual control descriptions later.

Name	Description
ATTACHMENU	Attaches a context menu to an object without displaying it.
CALCULATE	Forces a control bound to a symbolic (calculated) data column to re-evaluate its contents.
GETCHILDLIST	Returns a list of child objects for the specified parent object.
DRAGDETECT	Determines if a user is trying to drag on object.
FINDINITFOCUS	Returns the name of the first child control of a parent object that can accept the input focus.
GETPARENTFORM	Return the parent form for an object.
INVALIDATE	Marks an area of an object as invalid so Windows will repaint it.
MOVE	Moves the control or form to new coordinates
OFFSET	Moves the control or form to new coordinates by an offset amount.
POSTWINMSG	Posts a window message to the specified object.
QUALIFYWINMSG	Qualifies a Window Message so it can be trapped in a WINMSG event.
REPAINT	Repaints a control.
SCALEFONT	Scales an unscaled FONT structure relative to the current scale factor of the specified object.
SCALESIZE	Scales an unscaled SIZE structure relative to the current scale factor of the parent form.
SCALEVALUES	Scales an unscaled array of values relative to the current scale factor of the specified object.
SENDWINMSG	Sends a window message to the specified object.
SETPARENT	Moves the specified object and any child objects to a new parent object.
SETZORDER	Changes the position of the specified object within its parent object's Z-Order.
SHOWDATABINDING	Displays data-binding information for the specified object.
SHOWHELP	Displays help information for the specified object.
SHOWMENU	Displays the context menu for the specified object.
SHOWNOTES	Displays quick-help for the specified object.
SHOWOPTIONS	Displays the input options for the specified control.
TRACKPOPUPMENU	Creates and displays a context menu for the specified object.
UNSCALEFONT	Unscales an scaled FONT structure relative to the current scale factor of the specified object.
UNSCALESIZE	Unscales an scaled SIZE structure relative to the current scale factor of the specified object.
UNSCALEVALUES	Unscales an scaled array of values relative to the current scale factor of the specified object.

ATTACHMENU method

Description

Attaches a context menu to an object without displaying it.

Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "ATTACHMENU", MenuID )
```

Parameters

Name	Required	Description
MenuID	Yes	Fully-qualified repository ID of the context menu to attach.

Returns

Returns TRUE$ if the menu was attached, FALSE$ otherwise. Error information is reported via the Get_Status stored procedure.

Remarks

This method normally used to "pre-create" a context menu for an object rather than waiting until the first time the menu is used. This is necessary when the menu has accelerator keys that might needed before the menu has been displayed.

This method uses the ContextMenu ATTACHMENU function to display the menu.

Example

```
// Example - attach the ITEMS_MENU context menu to the LST_ITEMS control
$Insert RTI_SSP_Equates

MenuID = @AppID<1> : "*CONTEXTMENU**ITEMS_MENU"

Call Set_Status( SETSTAT_OK$ )
IsOK = Exec_Method( @Window : ".ITEMS_MENU", "ATTACHMENU", MenuID )
If IsOK Else
   Call Get_Status( sspErr )
End
```

See also

Common GUI CONTEXTMENU property, Common GUI SHOWMENU method, Common GUI TRACKPOPUPMENU method, Common GUI CONTEXTMENU event, Common GUI INITCONTEXTMENU event, Common GUI MENU event, ContextMenu stored procedure.

CALCULATE method

Description

Forces a control bound to a symbolic (calculated) data column to re-evaluate its contents by triggering the CALCULATE event.

Syntax

```
Status = Exec_Method( CtrlEntID, "CALCULATE", CtrlColumn )
```

Parameters

Name	Required	Description
CtrlColumn	No	If the object is an EditTable this parameter specifies the index of the EditTable column to recalculate. If no index is specified all EditTable columns in the control bound to symbolic data columns will be re-calculated.

Returns

The CALCULATE event status. If this is not null then an error has occurred.

Remarks

N/a.

Example

```
// Example – recalculate the 3rd column in the EDT_INVOICES EditTable
// control

Status = Exec_Method( @Window : ".EDT_INVOICES", "CALCULATE", 3 )
```

See also

Common GUI COLUMN property, Common GUI TABLE property, Common GUI CALCULATE event, Get_EventStatus stored procedure,

DRAGDETECT method

Description

Determines if the user is trying to drag an object. It captures the mouse and tracks its movement until the user releases the button, presses the ESC key, or moves the mouse outside the drag rectangle around the specified point.

This method should be called from an object's BUTTONDOWN event.

Syntax

```
IsDragging = Exec_Method( CtrlEntID, "DRAGDETECT", MouseButton, MouseX, MouseY )
```

Parameters

Name	Required	Description

MouseButton	Yes	Numeric value specifying the mouse button that was clicked down to begin the drag.
		
		`0 Left button`
		`1 Right button`
		`2 Middle button`

MouseX	Yes	Specifies the Left (X) coordinate where the mouse button was clicked down relative to the parent's client area.
MouseY	Yes	Specifies the Top (Y) coordinate where the mouse button was clicked down relative to the parent's client area.

Returns

If the user moved the mouse outside of the drag rectangle while holding down the button this method returns TRUE$. If the button was released inside the drag rectangle, or the ESC key was pressed it returns FALSE$.

Remarks

The system drag rectangle defines the number of pixels on either side of a mouse-down point that the mouse pointer can move before a drag operation begins. This allows the user to click and release the mouse button easily without unintentionally starting a drag operation.

The width and height of the drag rectangle are specified by the SM_CXDRAG and SM_CYDRAG values returned by the GetSystemMetrics function. Please refer to the Windows documentation on the Microsoft website for more details.

Example

```
// Example BUTTONDOWN event code - check if the user wants to "drag"
// the current obejct, and if so capture the mouse messages so that
// all subsequent MOUSEMOVE events will be directed to it.

If Exec_Method( CtrlEntID, "DRAGDETECT", MouseButton, xDown, yDown ) Then
   //  User wants to drag, so capture the mouse...
   Call Set_Property_Only( CtrlEntID, "MOUSECAPTURED", TRUE$ )
End
```

See also

Common GUI CURSOR property, Common GUI MOUSECAPTURED property, Common GUI BUTTONDOWN event, Common GUI BUTTONUP event, Common GUI LOSTCAPTURE event, Common GUI MOUSEMOVE event.

FINDINITFOCUS method

Description

Returns the name of the first child control of a parent object that can accept the input focus. This could be the parent object itself if it can accept the focus.

Syntax

```
InitFocusID = Exec_Method( CtrlEntID, "FINDINITFOCUS" )
```

Parameters

N/a.

Returns

A string value containing the name of a control that can accept the input focus, or null if none are found.

Remarks

This method is generally used with forms to find a control to pass the focus too.

Example

```
// Find a control on the ADM_CONTACTS form to pass the focus too

InitFocusID = Exec_Method( "ADM_CONTACTS", "FINDINITFOCUS" )

If BLen( InitFocusID ) Then
   Call Set_Property_Only( "SYSTEM", "FOCUS", InitFocusID )
End
```

See also

Common GUI FOCUS property, SYSTEM FOCUS property, Common GUI GOTFOCUS event, Common GUI LOSTFOCUS event.

GETCHILDLIST method

Description

This method returns a list of child object IDs for the specified parent object that match the specified filter criteria and options.

Syntax

```
ChildList = Exec_Method( CtrlEntID, "GETCHILDLIST", TypeID, RecurseFlag, |
                         NoSortFlag, VisibleOnlyFlag, PageNumber )
```

Parameters

Name	Required	Description
TypeID	No	If specified then only objects of this type are returned, otherwise objects of any type may be returned.
RecurseFlag	No	Normally, when the ParentID parameter is set only direct children of that parent will be returned. If RecurseFlag is set to TRUE$ then all objects that are descendants of ParentID are returned.
NoSortFlag	No	By default the list of objects returned in alphabetical order – if this parameter is TRUE$ then the list is returned with respect to the Z-order instead.
VisibleOnlyFlag	No	If TRUE$ then only visible objects are returned.
PageNumber	No	If specified then only objects with the same PageNumber property will be returned.

Returns

An @Fm-delimited list of PS objects matching the filter criteria, or null if no matches are found.

Remarks

This method is a thin wrapper around the SYSTEM OBJECTLIST method.

Example

```
// Example - return a list of visible MDI child forms for the current form
// (which we assume is an MDI frame form).

MdiChildren = Exec_Method( @Window : ".MDICLIENT", "GETCHILDLIST", "WINDOW" )
```

See also

SYSTEM OBEJCTLIST property.

GETPARENTFORM method

Description

Returns the name of the parent form for the specified object. Depending on the options passed this may be the direct parent form, the first non-child parent form, or a top-level form.

Syntax

```
ParentForm = Exec_Method( CtrlEntID, "GETAPRENTFORM", Flags )
```

Parameters

Name	Required	Description
Flags	No	Numeric value specifying options for the method: 0 Returns the parent form 1 Returns the first non-child parent form 2 Returns the root top-level parent form

Returns

A string containing the name of the parent form matching the passed Flags.

If the specified object is a top-level form then the object itself is returned.

Remarks

Using this method with a Flags value of "0" is the same as using the PARENTFORM property.

Equated constants for this method can be found in the PS_EQUATES insert record.

Example

```
// Get the top-level form for the current control
$Insert PS_Equates

ParentForm = Exec_Method( CtrlEntID, "GETPARENTFORM", PS_GPF_TOPLEVEL$ )
```

See also

Common GUI PARENT property, Common GUI PARENTFORM property, WINDOW MDIFRAME property, Common GUI SETPARENT method.

INVALIDATE method

Description

Marks an area of an object as invalid so Windows will repaint it.

Syntax

```
Unsued = Exec_Method( CtrlEntID, "INVALIDATE", EraseBkGd,    |
                                               UpdateArea,   |
                                               RepaintNow,   |
                                               IncludeNC )
```

Parameters

Name	Required	Description
EraseBkgd	No	If TRUE$ then Windows will erase the background of the object before painting the contents. Defaults to TRUE$.

UpdateArea	No	@Fm-delimited array specifying the area, in client coordinates, to repaint. This is the same as a normal SIZE property value:
		
		`<1> Left`
		`<2> Top`
		`<3> Width`
		`<4> Height`
		
		If null (the default) then the entire object's client area is repainted.

RepaintNow	No	If TRUE$ then the object is repainted immediately before the method returns. Normally objects are painted by Windows when it processes its message queue, so painting may not be instant of this parameter is FALSE$ (the default).
IncludeNC	No	If TRUE$ then an object's non-client area is repainted too. The default is FALSE$.

Returns

Not used – always returns null.

Remarks

The INVALIDATE method is basically a wrapper around the InvalidateRect and RedrawWindow Windows API functions, please refer to the documentation for these functions on the MSDN website for more details.

Coordinates are interpreted as DIPs or PX based on the parent form's SCALEUNITS property.

Example

```
// Invalidate s section of the current form, erasing the background

UpdateArea = ""
UpdateArea<1> = 10
UpdateArea<2> = 10
UpdateArea<3> = 200
UpdateArea<4> = 100

Call Exec_Method( @Window, "INVALIDATE", TRUE$, UpdateArea )

// Invalidate the entire EDT_STUFF control, including the non-client area and
// update it immediately

Call Exec_Method( @Window : ".EDT_STUFF", "INVALIDATE", TRUE$, "", TRUE$, TRUE$ )
```

See also

Common GUI REDRAW property, Common GUI REPAINT method.

MOVE method

Description

Moves a form or control to new coordinates while preserving its current width and height.

Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "MOVE", Left, Top )
```

Parameters

Name	Required	Description
Left	No	Specifies the new Left (X) coordinate. If null then the current value is kept.
Top	No	Specifies the new Top (Y) coordinate. If null then the current value is kept.

Returns

TRUE$ if the object was moved successfully, or FALSE$ otherwise.

Remarks

This method is equivalent to using the LEFT and TOP properties. All coordinates are relative to the client area of a control's parent, or to the origin of the primary monitor for a form.

Coordinates are interpreted as DIPs or PX based on the parent form's SCALEUNITS property.

Example

```
// Use the MOVE method to place the EDL_NAME control at a new position

Left = 10
Top  = 6
Call Exec_Method( @Window : ".EDL_NAME", "MOVE", Left, Top )
```

See also

Common GUI LEFT property, Common GUI RECT property, Common GUI SIZE property, Common GUI TOP property, Common GUI OFFSET method, Common GUI SIZE event, Appendix K – High DPI Programming.

OFFSET method

Description

Moves a form or control to new coordinates by an offset amount while preserving its current width and height.

Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "OFFSET", LeftOffset, TopOffset )
```

Parameters

Name	Required	Description
LeftOffset	No	Specifies the new Left (X) offset amount. If null then the current value is kept.
TopOffset	No	Specifies the new Top (Y) offset amount. If null then the current value is kept.

Returns

TRUE$ if the object was moved successfully, or FALSE$ otherwise.

Remarks

Offset values are interpreted as DIPs or PX based on the parent form's SCALEUNITS property.

Example

```
// Use the OFFSET method to place the EDL_NAME control at a new position
// 10 DIPs lower and 20 Dips to the left from its current location

LeftOffset = -20
TopOffset  = 10

Call Exec_Method( @Window : ".EDL_NAME", "OFFSET", LeftOffset, TopOffset )
```

See also

Common GUI LEFT property, Common GUI RECT property, Common GUI SIZE property, Common GUI TOP property, Common GUI MOVE method, WINDOW SIZE event, Appendix K – High DPI Programming.

QUALIFYWINMSG method

Description

Enables or disables the processing of a specified Windows message for a form or control. When enabled the object will trigger it's WINMSG event when the message is received.

Syntax

```
PrevQualInfo = Exec_Method( CtrlEntID, "QUALIFYWINMSG", MsgNum, NewQualInfo )
```

Parameters

Name	Required	Description
MsgNum	Yes	Message number to process.
NewQualInfo	Yes	An @Fm delimited array that specifies how to process the Windows message identified by MsgNum. <1> Enable Flag – TRUE$ to track the message, FALSE$ to disable tracking. This field is required. If TRUE$ then the following fields are valid. <2> Qualifier String. Can contain the name of an event qualifier to execute rather than the default (See Remarks below for more details). This field Is optional. <3> Event Name - Can contain the name of an event to execute rather than the default WINMSG event (See Remarks below for more details). This field is optional. <4> SyncFlags – specifies the priority of the event. Can be one of the following values: 0 : Asynchronous – the event is queued and executed as the queue is processed. This is the default. 1 : Basic Sync – the event is executed as soon as it is received. If this is not possible the event will be discarded. 2 : Callback Sync – similar to Synchronous, except that the event will also be executed if the PS is in a "wait state". (See Remarks below for more details)

Returns

The previous event qualifier information before any adjustments are made. This can be used with a subsequent call to QUALIFYWINMSG to reset the default processing.

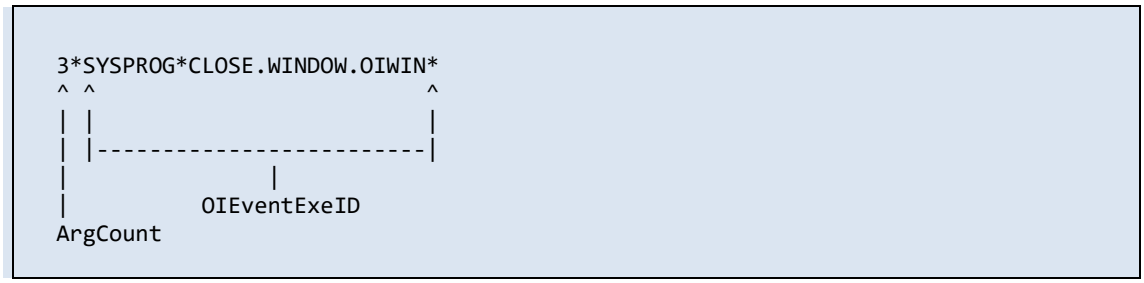
Remarks

Setting the "Qualifier String"

This allows the system to use a different event script at runtime rather than the default one. This can be useful when centralizing code – the script may be written once and then other controls redirected to use it. A qualifier string is defined by two parts, delimited by a "*" character. These are:

1. The number of arguments the event expects (including the object name and class), and
2. The ID of a SYSREPOSEVENTEXES object code record to execute when the event is fired.

E.g.



Setting the "Event Name"

This allows the system to fire a specific named event when the message is received rather than simply calling the WINMSG event. This may help to structure application code in a more meaningful fashion.

For example, when tracking the WM_SETFOCUS message, the Event Configuration dialog may be used to define a custom event called "MYSETFOCUS". If this same name is used with the QUALIFYWINMSG method then the new MYSETFOCUS event is triggered rather than the normal WINMSG event when the object receives the WM_SETFOCUS message.

Setting the "SyncFlags"

Normal PS event processing is executed in an asynchronous fashion, which means it is placed in a queue and run when the PS checks its message queue. However, this can be a problem with handling Windows messages as many of them can pass a pointer to extra data: By the time the event is executed in the PS any pointers will likely be invalid as the code triggering the Windows message will have finished and cleaned up memory. To avoid this problem this event should be handled in one of

the following synchronous ("Basic Sync" or "Callback Sync") manners instead because any pointers passed will still be valid.

In Basic Sync mode the PS attempts to execute the event as soon as it is notified. However, if it is busy processing a previously executed event then the new one cannot be processed and will be discarded.

In Callback Sync mode the PS attempts to execute the event as soon as it is notified. However, if it is busy processing a previously executed event then it checks to see if that one is actually in a "wait-state", i.e. it Basic+ has called back into the PS through use of something like a Set_Property call and is waiting for the PS to respond. This can happen if setting the property generates a Windows notification message which triggers a PS event – the event can be raised before the Set_Property call returns. Using the Callback mode is generally a better idea than the Basic synchronous mode.

Equates for the core Window messages can be found in the MSWIN_WINDOWMESSAGE_EQUATES insert record. Equates for the control-specific message like ComboBoxes and EditLines can be found in their respective MSWIN_<controltype>_EQUATES records.

Example

```
// Example : Track the WM_CAPTURECHANGED message

$Insert MsWin_WindowMessge_Equates
$Insert Logical

NewQualInfo  = TRUE$
PrevQualInfo = Exec_Method( CtrlEntID, "QUALIFYWINMSG", |
                            WM_CAPTURECHANGED$,         |
                            NewQualInfo )

//  Do some processing...

// Stop tracking and reset the event to it's defaults.
Call Exec_Method( CtrlEntID, "QUALIFYWINMSG", |
                  WM_CAPTURECHANGED$,         |
                  PrevQualInfo )
```

See also

Common GUI QUALFIEDWINMSGS property, Common GUI QUALIFYEVENT method, OLECONTROL QUALIFYOLEEVENT method, Common GUI WINMSG event, Appendix C – Event handling.

POSTWINMSG method

Description

Posts a window message to the specified object and returns without waiting for it to be processed.

Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "POSTWINMSG", Msg, wParam, lParam )
```

Parameters

Name	Required	Description
Msg	Yes	Integer specifying the message to be posted.
wParam	No	Message-specific integer value. Defaults to 0.
lParam	No	Message-specific integer value. Defaults to 0.

Returns

TRUE$ if the message was posted successfully, FALSE$ otherwise.

Remarks

It is possible to post a message to an object using a direct call to the Windows PostMessage API function from Basic+. However, the PS and the Basic+ engine operate on different threads and some window messages are thread-sensitive, so it is always better to use this method rather than a direct PostMessage call.

Equated constants for common window message values can be found in the MSWIN_WINDOWMESSAGE_EQUATES insert record.

This method uses the Windows API PostMessage function internally – for further information please see the Microsoft website.

Example

```
// Post a WM_SYSCOMMAND message to the current window to bring up
// the System Menu
$Insert MSWin_WindowMessage equates
$Insert MSWin_SysCommand_Equates
$Insert MSWin_VirtualKey_Equates

PostOK = Exec_Method( @Window, "POSTWINMSG", WM_SYSCOMMAND$, SC_KEYMENU$, VK_SPACE$ )
```

See also

Common GUI QUALIFYWINEVENT method, Common GUI SENDWINMSG method, Common GUI WINMSG event, SYSTEM POSTWINMSG method, SYSTEM PROCESSWINMSGS method, SYSTEM SENDWINMSG method.

REPAINT method

Description

Forces an object to repaint itself according to a set of specified flags.

Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "REPAINT", RepaintFlags, UpdateArea )
```

Parameters

Name	Required	Description
RepaintFlags	No	Specifies a bitmask of flags that control how the repaint request is handled. These flags are the "RDW_" flags from the Windows RedrawWIndow API function: RDW_INVALIDATE (0x0001) RDW_INTERNALPAINT (0x0002) RDW_ERASE (0x0004) RDW_VALIDATE (0x0008) RDW_NOINTERNALPAINT (0x0010) RDW_NOERASE (0x0020) RDW_NOCHILDREN (0x0040) RDW_ALLCHILDREN (0x0080) RDW_UPDATENOW (0x0100) RDW_ERASENOW (0x0200) RDW_FRAME (0x0400) RDW_NOFRAME (0x0800) If null is passed then the default flags applied are: RDW_ERASE RDW_FRAME RDW_INVALIDATE RDW_UPDATENOW RDW_ALLCHILDREN This paints the object and it's children, along with the non-client area immediately without waiting for the next WM_PAINT cycle.
UpdateArea	No	@Fm-delimited SIZE coordinates of the area to update: <1> Left <2> Top <3> Width <4> Height If null is passed then the entire control is repainted.

Returns

TRUE$ if the object was repainted successfully, or FALSE$ otherwise.

Remarks

The REPAINT method is implemented internally using the RedrawWindow Windows API function. More information on this function can be found on the Microsoft website.

Coordinates are interpreted as DIPs or PX based on the parent form's SCALEUNITS property.

Equated constants for the "RDW_" flag values can be found in the MSWIN_REDRAWWINDOW_EQUATES insert record.

Example

```
// Repaint all of the current control and any children immediately without
// waiting for the next WM_PAINT cycle

RepaintOK = Exec_Method( CtrlEntID, "REPAINT" )


// Repaint the part of the current control ignoring any children
// and the non-client area

$Insert MsWin_RedrawWindow_Equates

RepaintFlags = ( RDW_ERASE$ + RDW_INVALIDATE$ + RDW_NOCHILDREN$ )
UpdateArea   = 0 : @Fm : 0 : @Fm : 200 : @Fm : 100

RepaintOK = Exec_Method( CtrlEntID, "REPAINT", RepaintFlags, UpdateArea )
```

See also

Common GUI REDRAW property, Common GUI INVALIDATE method.

SCALEFONT method

Description

Converts a device-independent font structure to a device-dependent font structure by adjusting its height with respect to the specified object's DPI and SCALEFACTOR.

Syntax

```
FontInPX = Exec_Method( CtrlEntID, "SCALEFONT", FontInDIPs )
```

Parameters

Name	Required	Description
FontInDIPs	Yes	Standard OpenInsight @Svm-delimited array representing a font as per the FONT property.

Returns

An @Svm-delimited array representing a font (as per the FONT property) scaled to the same factor and DPI as the specified object.

Remarks

Please refer to the FONT property for more details on the font structure.

Example

```
// Obtain a font from a control and scale it so it matches the scale on
// a different control

// This will be returned in DIPs so it is "unscaled"
MyFont = Get_Property( @Window : ".MY_CTRL", "FONT" )

// Let's see what it would look like if we were going to set it for
// another control
ScaledFont = Exec_Method( "ANOTHER_WINDOW.ANOTHER_CTRL", "SCALEFONT", MyFont )
```

See also

Common GUI DPI property, Common GUI FONT property, Common GUI SCALEFACTOR property, Common GUI SCALEMETRICS property, Common GUI UNSCALEFONT method, Appendix K – High DPI Programming.

SCALESIZE method

Description

Converts a device-independent size array to a device-dependent size array by adjusting its coordinates to the specified object's DPI and SCALEFACTOR.

Syntax

```
SizeInPX = Exec_Method( CtrlEntID, "SCALESIZE", SizeInDIPs )
```

Parameters

Name	Required	Description
SizeInDIPs	Yes	Standard OpenInsight @Fm-delimited array representing size coordinates as per the SIZE property.

Returns

An @Fm-delimited array representing a size (as per the SIZE property) scaled to the same factor and DPI as the specified object.

Remarks

Please refer to the SIZE property for more details on the size array.

Example

```
// Obtain a SIZE from a control and scale it so it matches the scale on
// a different control

// This will be returned in DIPs so it is "unscaled"
MySize = Get_Property( @Window : ".MY_CTRL", "SIZE" )

// Let's see what it would look like if we were going to set it for
// a different control
ScaledSize = Exec_Method( "ANOTHER_WINDOW.ANOTHER_CTRL", "SCALESIZE", MySize )
```

See also

Common GUI DPI property, Common GUI SCALEFACTOR property, Common GUI SCALEMETRICS property, Common GUI SIZE property, Common GUI UNSCALESIZE method, Appendix K – High DPI Programming.

SCALEVALUES method

Description

Converts a @Fm-delimited array of device-independent numeric values to a device-dependent array the values with respect to the specified object's DPI and SCALEFACTOR.

Syntax

```
PXVals = Exec_Method( CtrlEntID, "SCALEVALUES", DIPVals )
```

Parameters

Name	Required	Description
DIPVals	Yes	An @Fm-delimited array of device-dependent numeric values.

Returns

An @Fm-delimited array of values scaled to the same factor and DPI as the specified object.

Remarks

N/a.

Example

```
// Assume we have an array of DIP values that need to be
// converted to Pixels using the same DPI and scaling factor
// as the current form

DIPVals = 120 : @Fm : 160 : @Fm : 20

PXVals = Exec_Method( @Window, "SCALEVALUES", DIPVals )
```

See also

Common GUI DPI property, Common GUI SCALEFACTOR property, Common GUI SCALEMETRICS property, Common GUI UNSCALEVALUES method, Appendix K – High DPI Programming.

SENDWINMSG method

Description

Sends a window message to the specified object and waits for it to be processed before returning.

Syntax

```
lResult = Exec_Method( CtrlEntID, "SENDWINMSG", Msg, wParam, lParam )
```

Parameters

Name	Required	Description
Msg	Yes	Integer specifying the message to be sent.
wParam	No	Message-specific integer value. Defaults to 0.
lParam	No	Message-specific integer value. Defaults to 0.

Returns

An integer with the result from the processed message – this depends on the message sent.

Remarks

It is possible to send a message to an object using a direct call to the Windows SendMessage API function from Basic+. However, the PS and the Basic+ engine operate on different threads and some window messages are thread-sensitive, so it is always better to use this method rather than a direct SendMessage call.

Equated constants for common window message values can be found in the MSWIN_WINDOWMESSAGE_EQUATES insert record.

This method uses the Windows API SendMessage function internally – for further information please see the Microsoft website.

Example

```
// Simulate a click on the BTN_TEST button using a pair of down/up mouse messages
$Insert MSWin_WindowMessage_Equates

lResult = Exec_Method( @window : ".BTN_TEST", "SENDWINMSG", WM_LBUTTONDOWN$, 0, 0 )
lResult = Exec_Method( @window : ".BTN_TEST", "SENDWINMSG", WM_LBUTTONOP$, 0, 0 )
```

See also

Common GUI QUALIFYWINEVENT method, Common GUI POSTWINMSG method, Common GUI WINMSG event, SYSTEM POSTWINMSG method, SYSTEM PROCESSWINMSGS method, SYSTEM SENDWINMSG method.

SETPARENT method

Description

Moves the specified object and any child objects to a new parent object.

Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "SETPARENT", ParentID )
```

Parameters

Name	Required	Description
ParentID	Yes	Specifies the name of the new parent object.

Returns

TRUE$ if the object was moved successfully, or FALSE$ otherwise.

Remarks

If the object is moved to a different form its name and the Window Common area is updated accordingly. The object is also scaled according the destination DPI and scale factor.

Data-bound controls cannot be moved to a different form.

The SETPARENT method is implemented internally using the SetParent Windows API function, so please refer to the documentation on the Microsoft website for further information on parent and child relationships in Windows.

Example

```
// Move the PNL_OPTIONS panel, and its child controls, to a different form

IsOK = Exec_Method( "ZZ_ADM_CONFIG.PNL_OPTIONS", "SETPARENT", "ZZ_ADM_CLIENTS" )

If IsOK Then
   // The PNL_OPTIONS panel now has a name of "ZZ_ADM_CLIENTS.PNL_OPTIONS"
End
```

See also

Common GUI PARENT property Common GUI PARENTFORM property, Common GUI GETPARENTFORM method,

SETZORDER method

Description

Changes the position of the specified object within its parent object's "z-order" by moving it above one of its siblings (every parent object with children has a "z-order" list which determines the order of which objects appear on front of others).

Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "SETZORDER", ObjectBeneath )
```

Parameters

Name	Required	Description
ObjectBeneath	No	The name of the object that will be immediately beneath the specified object in the z-order. If this parameter is null then the specified object is moved to the top of the z-order. If this parameter is "-1" then the specified object is moved to the bottom of the z-order.

Returns

TRUE$ if the object was moved successfully, or FALSE$ otherwise.

Remarks

A better way to ensure that some objects appear in front of others is to use proper parent/child relationships instead of making them all siblings and attempting to control their rendering by setting the z-order (this was common practice in earlier versions of OpenInsight as support for parent/child relationships was very limited).

It is important to ensure that an object that is higher in the z-order does not partially overlap the borders of one beneath it, otherwise it may not be rendered correctly.

The SETZORDER method is implemented internally using the SetWindowPos Windows API function, so please refer to the documentation on the Microsoft website for further information on changing the z-order.

Example

```
// Set the EDL_NAME control above the GRP_MAIN groupbox so the latter doesn't
// paint over it.

IsOK = Exec_Method( @Window : ".EDL_NAME", "SETZORDER", @Window : ".GRP_MAIN" )

// Set the TAB_MAIN tab control to the bottom of the z-order

IsOK = Exec_Method( @Window : ".TAB_MAIN", "SETZORDER", -1 )
```

See also

Common GUI PARENT property, Common GUI SIZE property, WINDOW TOPMOST property, Common GUI SETPARENT method.

SHOWDATABINDING method

Description

Displays a dialog box with column data-binding information for the specified object.

Syntax

```
Call Exec_Method( CtrlEntID, "SHOWDATABINDING" )
```

Parameters

N/a.

Returns

N/a.

Remarks

If this method used against a form (i.e. WINDOW object) a simple list of tables bound to the form is displayed instead of the above dialog. If the specified object is not databound a message is displayed informing the user.

Example

```
// Example - display the data-binding information for the EDL_NAME
// control

Call  Exec_Method( @Window : ".EDL_NAME", "SHOWDATABINDING" )
```

See also

Common GUI COLUMN property, Common GUI TABLE property.

SHOWHELP method

Description

Displays help information for the specified object by triggering its HELP event.

Syntax

```
Status = Exec_Method( CtrlEntID, "SHOWHELP" )
```

Parameters

N/a.

Returns

The HELP event status. If this is not null then an error has occurred.

Remarks

N/a.

Example

```
// Example - display the help information for the EDL_NAME
// control

Status = Exec_Method( @Window : ".EDL_NAME", "SHOWHELP" )
```

See also

Common GUI HELP event, Get_EventStatus stored procedure.

SHOWMENU method

Description

Displays the context menu associated with the specified object.

Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "SHOWMENU", xPos, yPos, bRightAlign )
```

Parameters

Name	Required	Description
xPos	No	The horizontal position in *client* coordinates where the menu should be displayed.
yPos	No	The vertical position in *client* coordinates where the menu should be displayed.
bRightAlign	No	If TRUE$ then the context menu should be right aligned.

Returns

The TRUE$ if the menu was displayed, FALSE$ otherwise. Error information is reported via the Get_Status stored procedure.

Remarks

This method uses the ContextMenu SHOWMENU function to display the menu.

Unlike most of the underlying Context Menu processing this method uses client coordinates rather than screen coordinates and translates them to the latter internally.

Example

```
// Example - display the context menu for the EDL_NAME
// control at coordinates 20, 10
$Insert RTI_SSP_Equates

Call Set_Status( SETSTAT_OK$ )
IsOK = Exec_Method( @Window : ".EDL_NAME", "SHOWMENU", 20, 10 )
If IsOK Else
   Call Get_Status( sspErr )
End
```

See also

Common GUI CONTEXTMENU property, Common GUI ATTACHMENU method, Common GUI TRACKPOPUPMENU method, Common GUI CONTEXTMENU event, Common GUI INITCONTEXTMENU event, Common GUI MENU event, ContextMenu stored procedure, Get_Status stored procedure.

SHOWNOTES method

Description

Displays quick-help information for the specified object by triggering its NOTES event.

Syntax

```
Status = Exec_Method( CtrlEntID, "SHOWNOTES" )
```

Parameters

N/a.

Returns

The NOTES event status. If this is not null then an error has occurred.

Remarks

This event is not related to Lotus Notes functionality that was found in previous versions of OpenInsight.

Example

```
// Example - display the QuickHelp information for the EDL_NAME
// control

Status = Exec_Method( @Window : ".EDL_NAME", "SHOWNOTES" )
```

See also

Common GUI NOTES event, Get_EventStatus stored procedure.

SHOWOPTIONS method

Description

Displays options information for the specified object by triggering its OPTIONS event.

Syntax

```
Status = Exec_Method( CtrlEntID, "SHOWOPTIONS", MoveFocus, ProcessEvents )
```

Parameters

Name	Required	Description
MoveFocus	No	If TRUE$ then move the focus to the object before triggering the OPTIONS event.
ProcessEvents	No	If TRUE$ and MoveFocus is TRUE$ then allow the system to process any pending events before the OPTIONS event is triggered.

Returns

The OPTIONS event status. If this is not null then an error has occurred.

Remarks

When moving the focus to the specified object a check is made to see if the move was successful. If this failed then the OPTIONS event is not triggered.

Example

```
// Example - display the options information for the EDL_NAME control, and ensure the
// focus is moved there first and events processed in case validation is triggered.

Status = Exec_Method( @Window : ".EDL_NAME", "SHOWIOPTIONS", TRUE$, TRUE$ )
```

See also

SYSTEM FOCUS property, SYSTEM PROCESSEVENTS method, Common GUI OPTIONS event, Get_EventStatus stored procedure.

TRACKPOPUPMENU method

Description

Creates and displays a context menu for the specified object at the requested coordinates.

When a context menu item is selected a MENU event is raised and sent to the parent object.

Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "TRACKPOPUPMENU", |
                           MenuStruct,                  |
                           xPos,                        |
                           yPos,                        |
                           Flags )
```

Parameters

Name	Required	Description
MenuStruct	Yes	A dynamic array containing the executable structure of the menu.
xPos	No	The horizontal position in screen coordinates where the menu should be displayed.
yPos	No	The vertical position in screen coordinates where the menu should be displayed.
Flags	No	A numeric bitmask of "TPM_" flags that specify how the menu is positioned horizontally and vertically.

Returns

TRUE$ if the menu is created successfully, FALSE$ otherwise.

Remarks

This method is called by the CONTEXTMENU event to display the context menu associated with an object in its CONTEXTMENU property.

This is considered a low-level method. It is better to set the CONTEXTMENU property and call the SHOWMENU method rather than calling TRACKMENUPOPUP directly.

Equates constants for working with menu structures can be found in the PS_MENU_EQUATES insert record. Equated constants for the "TPM_" values can be found in the MSWIN_MENU_EQUATES insert record.

The TRACKPOPUPMENU method is implemented internally using the TrackPopupMenu Windows API function, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Display a context menu for the current control with two items
// and a separator

$Insert PS_Menu_Equates
$insert MSWin_Menu_Equates

MenuStruct = ""
MenuID     = CtrlEntID : ".CONTEXTMENU"

// Header values...
MenuStruct<0,MENUHDRPOS_NAME$>    = MenuID
MenuStruct<0,MENUHDRPOS_CLASS$>   = MENUCLASS_CONTEXT$  ; // "FLOATING"
MenuStruct<0,MENUHDRPOS_TYPE$>    = MENUTYPE_MENU$      ; // "MENU"
MenuStruct<0,MENUHDRPOS_PARENT$> = CtrlEntID

ItemStruct = ""
ItemStruct<0,0,MENUPOS_TYPE$> = MENUTYPE_ITEM$
ItemStruct<0,0,MENUPOS_END$>  = FALSE$
ItemStruct<0,0,MENUPOS_NAME$> = MenuID : ".OPEN_SESAME"
ItemStruct<0.0,MENUPOS_TEXT$> = "Open Sesame"

MenuStruct<0,-1> = ItemStruct

ItemStruct = ""
ItemStruct<0,0,MENUPOS_TYPE$> = MENUTYPE_SEPARATOR$$
ItemStruct<0,0,MENUPOS_END$   = FALSE$
ItemStruct<0.0,MENUPOS_NAME$  = MenuID : ".SEP101"
ItemStruct<0.0,MENUPOS_TEXT$> = "SEP101"

MenuStruct<0,-1> = ItemStruct

ItemStruct = ""
ItemStruct<0,0,MENUPOS_TYPE$> = MENUTYPE_ITEM$
ItemStruct<0,0,MENUPOS_END$>  = TRUE$
ItemStruct<0,0,MENUPOS_NAME$> = MenuID : ".CLOSE_SESAME"
ItemStruct<0.0,MENUPOS_TEXT$> = "Close Sesame"

MenuStruct<0,-1> = ItemStruct

IsOK = Exec_Method( CtrlEntID, "TRACKPOPUPMENU", MenuStruct, 10, 10, TPM_RIGHTALIGN$ )
```

See also

Common GUI CONTEXTMENU property, Common GUI ATTACHMENU method, Common GUI SHOWMENU method, Common GUI CONTEXTMENU event, Common GUI INITCONTEXTMENU event, Common GUI MENU event, ContextMenu stored procedure.

UNSCALEFONT method

Description

Converts a device-dependent font structure to a device-independent font structure by adjusting its height with respect to the specified object's DPI and SCALEFACTOR.

Syntax

```
FontInDIPs = Exec_Method( CtrlEntID, "UNSCALEFONT", FontInPX )
```

Parameters

Name	Required	Description
FontInPX	Yes	Standard OpenInsight @Svm-delimited array representing a font as per the FONT property. This should contain scaled values.

Returns

An @Svm-delimited array representing a font (as per the FONT property) unscaled (i.e. to 96DPI and a SCALEFACTOR of 1).

Remarks

Please refer to the FONT property for more details on the font structure.

Example

```
// A bit of a contrived example, but... get the font from the MY_CTRL object
// in it's scaled form (i.e. in pixels)

// First make sure we're working with pixels ...
PrevSU = Set_Property( @Window, "SCALEUNITS", PS_SCU_PIXELS$ )

// Now get the font...
MyFontPX = Get_Property( @Window : ".MY_CTRL", "FONT" )

// Reset the scale units
Set_Property( @Window, "SCALEUNITS", PrevSU )

// Let's see what it would look like in DIPs
MyFontDIPs = Exec_Method( @Window : ".MY_CTRL", "UNSCALEFONT", MyFontPX )
```

See also

Common GUI DPI property, Common GUI FONT property, Common GUI SCALEFACTOR property, Common GUI SCALEMETRICS property, Common GUI SCALEFONT property, Appendix K – High DPI Programming.

UNSCALESIZE method

Description

Converts a device-dependent size array to a device-independent size by adjusting its coordinates with respect to the specified object's DPI and SCALEFACTOR.

Syntax

```
SizeInDIPs = Exec_Method( CtrlEntID, "UNSCALESIZE", SizeInPX )
```

Parameters

Name	Required	Description
SizeInPX	Yes	Standard OpenInsight @Fm-delimited array representing size coordinates as per the SIZE property.

Returns

An @Fm-delimited array representing a size (as per the SIZE property) unscaled (i.e. to 96 DPI and a SCALEFACTOR of 1).

Remarks

Please refer to the SIZE property for more details on the size array.

Example

```
// A bit of a contrived example, but... get the size from the MY_CTRL object
// in its scaled form (i.e. in pixels)

// First make sure we're working with pixels ...
PrevSU = Set_Property( @Window, "SCALEUNITS", PS_SCU_PIXELS$ )

// Now get the font...
MySizePX = Get_Property( @Window : ".MY_CTRL", "SIZE" )

// Reset the scale units
Set_Property( @Window, "SCALEUNITS", PrevSU )

// Let's see what it would look like in DIPs
MySizeDIPs = Exec_Method( @Window : ".MY_CTRL", "UNSCALESIZE", MySizePX )
```

See also

Common GUI DPI property, Common GUI SCALEFACTOR property, Common GUI SCALEMETRICS property, Common GUI SCALESIZE method, Common GUI SIZE property, Appendix K – High DPI Programming.

UNSCALEVALUES method

Description

Converts an array of device-dependent numeric values to corresponding device-independent values by adjusting them with respect to the specified object's DPI and SCALEFACTOR.

Syntax

```
DIPValues = Exec_Method( CtrlEntID, "UNSCALEVALUES", PXValues )
```

Parameters

Name	Required	Description
PXValues	Yes	An @Fm-delimited array of device-dependent numeric values.

Returns

An @Fm-delimited array of unscaled values (i.e. to 96 DPI and a SCALEFACTOR of 1).

Remarks

N/a.

Example

```
// Assume we have an array of pixel values that need to be
// converted to DIPs using the same DPI and scaling factor
// as the current form

PXVals  = 120 : @Fm : 160 : @Fm : 20

DIPVals = Exec_Method( @Window, "UNSCALEVALUES", PXVals )
```

See also

Common GUI DPI property, Common GUI SCALEFACTOR property, Common GUI SCALEMETRICS property, Common GUI SCALEVALUES method, Appendix K – High DPI Programming.

Common GUI Events

These events apply to most GUI forms and controls except where noted in individual control descriptions later.

Name	Description
BUTTONDOWN	Occurs when the user presses a mouse button down over an object.
BUTTONUP	Occurs when the user releases a mouse button over an object.
CALCULATE	Occurs when a control bound to a symbolic (calculated) database column re-evaluates its contents.
CHAR	Occurs when a character is entered into a control
CONTEXTMENU	Occurs when the system displays a context menu in response to a right-click.
DROPFILES	Occurs when files are dragged from the Windows Explorer onto an object.
GOTFOCUS	Occurs when control receives input focus.
HELP	Occurs when the user has requested to see online help.
HSCROLL	Occurs when an object is scrolled vertically.
INITCONTEXTMENU	Fired to initialize a context menu in response to a right-click.
LOSTCAPTURE	Occurs when an object loses mouse capture.
LOSTFOCUS	Occurs when a control loses the input focus.
MENU	Occurs when an menu item is selected.
MOUSEMOVE	Occurs when a mouse movement notification is received by an object.
NOTES	Occurs when the user has requested to see QuickHelp.
OMNIEVENT	Generic, developer defined event.
OPTIONS	Occurs when a user wants to see input options for a control.
REQUIRERR	Occurs when a "required" control has no data.
SYSMSG	Occurs when OpenInsight needs to display a system message.
TIMER	Occurs when a specified timer interval for an object has elapsed.
VALIDERR	Occurs when a control fails its validation check.
VSCROLL	Occurs when an object is scrolled vertically.
WINMSG	Occurs when a qualified window message is received by an object.

BUTTONDOWN event

Description

Occurs when the user presses a mouse button down over an object.

Syntax

```
bForward = BUTTONDOWN( CtrlEntID,
                       CtrlClassID,
                       xDown,
                       yDown,
                       xUp,
                       yUp,
                       CtrlKey,
                       ShiftKey,
                       MouseButton )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
xDown	Client X-coordinate (Left) where the mouse button was pressed down.
yDown	Client Y-coordinate (Top) where the mouse button was pressed down.
xUp	Client X-coordinate (Left) where the mouse button was released. This is always zero for a BUTTONDOWN event.
yUp	Client Y-coordinate (Top) where the mouse button was released. This is always zero for a BUTTONDOWN event.
CtrlKey	TRUE$ if the Ctrl key was pressed down when the mouse button was pressed, FALSE$ otherwise.
ShiftKey	TRUE$ if the Shift key was pressed down when the mouse button was pressed, FALSE$ otherwise.
MouseButton	Integer specifying which mouse button was pressed: "0" - Left button "1" - Right button "2" - Middle button

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

There is no system-level event handler for BUTTONDOWN.

Equated constants for the MouseButton parameter can be found in the PS_EQUATES insert record.

In previous versions of OpenInsight only forms supported a BUTTONDOWN event. In this version all GUI objects support them.

For more information on this event please refer to the Windows documentation regarding the WM_LBUTTONDOWN, WM_RBUTTONDOWN and WM_MBUTTONDOWN window messages on the Microsoft website.

Example

```
Function BUTTONDOWN( CtrlEntID, CtrlClassID,  xDown, yDown, xUp, yUp, ShiftKey, |
                                                        CtrlKey, MouseButton )

  // Example BUTTONDOWN event code - check if the user wants to "drag"
  // the current obejct, and if so capture the mouse messages so that
  // all subsequent MOUSEMOVE events will be directed to it.

  $Insert PS_Equates
  $Insert Logical

  // Only drag if it's the left button!
  If ( MouseButton == MBUTTON_LEFT$ ) Then
     If Exec_Method( CtrlEntID, "DRAGDETECT", MouseButton, xDown, yDown ) Then
        // User wants to drag, so set a UDP flag that we are dragging and
        // capture the mouse...
        Call Set_Property_Only( CtrlEntID, "@DRAGGING", TRUE$ )
        Call Set_Property_Only( CtrlEntID, "MOUSECAPTURED", TRUE$ )
     End
  End

Return TRUE$
```

See Also

Common GUI CURSOR property, Common GUI MOUSECAPTURED property, Common GUI BUTTONUP event, Common GUI LOSTCAPTURE event, Common GUI MOUSEMOVE event.

BUTTONUP event

Description

Occurs when the user releases a mouse button over an object.

Syntax

```
bForward = BUTTONUP( CtrlEntID,
                     CtrlClassID,
                     xDown,
                     yDown,
                     xUp,
                     yUp,
                     CtrlKey,
                     ShiftKey,
                     MouseButton )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
xDown	Client X-coordinate (Left) where the mouse button was pressed down.
yDown	Client Y-coordinate (Top) where the mouse button was pressed down.
xUp	Client X-coordinate (Left) where the mouse button was released.
yUp	Client Y-coordinate (Top) where the mouse button was released.
CtrlKey	TRUE$ if the Ctrl key was pressed down when the mouse button was released, FALSE$ otherwise.
ShiftKey	TRUE$ if the Shift key was pressed down when the mouse button was released, FALSE$ otherwise.
MouseButton	Integer specifying which mouse button was released: "0" - Left button "1" - Right button "2" - Middle button

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

There is no system-level event handler for BUTTONUP.

Equated constants for the MouseButton parameter can be found in the PS_EQUATES insert record.

In previous versions of OpenInsight only forms supported a BUTTONUP event. In this version all GUI objects support them.

For more information on this event please refer to the Windows documentation regarding the WM_LBUTTONUP, WM_RBUTTONUP and WM_MBUTTONUP window messages on the Microsoft website.

Example

```
Function BUTTONUP( CtrlEntID, CtrlClassID,  xDown, yDown, xUp, yUp, ShiftKey, |
                                                    CtrlKey, MouseButton )

 // Example BUTTONUP event code - check if we were dragging when the user released
 // the Left mouse button, and if so move it.
 $Insert Ps_Equates
 $Insert Logical

 If ( MouseButton = MBUTTON_LEFT$ ) Then

    IsDragging = Get_Property( CtrlEntID, "@DRAGGING" )
    If IsDragging then
       // We are flagged for dragging so remove the flag
       Call Set_Property_Only( CtrlEntID, "@DRAGGING", FALSE$ )
    End

    // If we are still captured then release the capture...
    If Get_Property( CtrlEntID, "MOUSECAPTURED" ) Then
       Call Set_Property_Only( CtrlEntID, "MOUSECAPTURED", FALSE$ )
    End Else
       // If the mouse wasn't captured then the drag must have been aborted
       IsDragging = FALSE$
    End

    If IsDragging Then
       // Move, but only if the coordinates are within the parent's client area
       Parent       = Get_Property( CtrlEntID, "PARENT" )
       ParentClient = Get_Property( Parent, "CLIENTSIZE" )

       If ( xUp > 0 ) And ( yUp > 0 ) Then
             If ( xUp < ParentClient<1> ) And ( yUp < ParentClient<2> ) Then
                   xOffset = ( xUp - xDown )
                   yOffset = ( yUP - yDown )
                   Call Exec_Method( CtrlEntID, "OFFSET", xOffset, yOffset )
                End
             End
          End
       End
    End
  End

Return TRUE$
```

See Also

Common GUI CURSOR property, Common GUI MOUSECAPTURED property, Common GUI BUTTONDOWN event, Common GUI LOSTCAPTURE event, Common GUI MOUSEMOVE event.

CALCULATE event

Description

Occurs when a control bound to a symbolic (calculated) database column re-evaluates its contents.

Syntax

```
bForward = CALCULATE( CtrlEntID, CtrlClassID, CtrlColumn )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
CtrlColumn	If the control is an EditTable this parameter identifies the index of the EditTable column to be re-evalulated. If no index is specified all EditTable columns in the control bound to symbolic data columns will be re-calculated.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

The system-level event handler handles the re-evaluation of the data and updates the control's contents.

- @Record is set to the RECORD property of the parent form.
- @ID is set to the ID property of the form.
- Any dependent controls are also updated (identified at compile-time)

For forms bound to multiple tables, only controls bound to the columns in the primary table will be re-calculated.

Example

```
Function CALCULATE( CtrlEntID, CtrlClassID, CtrlColumn )

  // Example - set some data in the EDL_DATE property using INVALUE – this will update
  // the RECORD property so that the CALCULATE event uses this data in its processing

  // Update EDL_DATE with the current date
  Call Set_Property( @Window : ".EDL_DATE", "INVALUE", Date() )

Return TRUE$ ; // Now allow the system to perform the calculation...
```

See Also

Common GUI COLUMN property, Common GUI TABLE property, WINDOW ID property, WINDOW RECORD property, Common GUI CALCULATE method, Get_EventStatus stored procedure.

CHAR event

Description

Occurs when a character is entered into a control with the input focus.

Syntax

```
bForward = CHAR( CtrlEntID, CtrlClassID, VirtCode, ScanCode, Ctrl, Shift, Alt )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
VirtCode	Contains the character entered if it is a printable character.
ScanCode	Contains the virtual key code of the character entered if it is not a printable character.
Ctrl	Boolean value set to TRUE$ if a Control key was pressed down when the character was entered or FALSE$ otherwise.
Shift	Boolean value set to TRUE$ if a Shift key was pressed down when the character was entered or FALSE$ otherwise.
Alt	Boolean value set to TRUE$ if an Alt key was pressed down when the character was entered or FALSE$ otherwise.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

There is no system-level event handler for the CHAR event.

Not all keystrokes will generate a CHAR event because some objects consume them internally, e.g. the Tab and Enter keys usually do not trigger a CHAR event. A common workaround for this is to create hidden menu items with an ACCELERATOR property that matches the desired keystroke and us ethe MENU event to respond to them.

Virtual key codes are numeric constants Windows uses to identify keystrokes. Equated constants for these can be found in the MSWIN_VIRTUAL_EQUATES insert record.

Example

```
Function CHAR( CtrlEntID, CtrlClassID, VirtCode, ScanCode, Ctrl, Shift, Alt )

  $Insert MSWin_VirtualKey_Equates
  $Insert Logical

  RetVal = FALSE$

  Begin Case
     Case ( VirtCode == " " )
        // If the user enters a space character then treat this as
        // a Click operation
        Call Exec_Method( CtrlEntID, "CLICK" )

     Case ( ScanCode == VK_DOWN$ )
        // If the user hits the "Down" key then move to the next
        // control

        NextCtrl = Get_Property( CtrlEntID, "NEXT" )
        Call Set_Property( "SYSTEM", "FOCUS". NextCtrl )

     Case ( ScanCode == VK_UP$ )
        // If the user hits the "Up" key then move to the previous
        // control

        PrevCtrl = Get_Property( CtrlEntID, "PEEVIOUS" )
        Call Set_Property( "SYSTEM", "FOCUS". PrevCtrl )

     Case OTHERWISE$
        RetVal = TRUE$

  End Case

Return RetVal
```

See Also

Common GUI ECHO property.

CONTEXTMENU event

Description

Occurs when the system is about to display a context menu in response to a right click, giving the application chance to modify the structure if desired.

Syntax

```
bForward = CONTEXTMENU( CtrlEntID,
                        CtrlClassID,
                        MenuID,
                        MenuStruct,
                        xPos,
                        yPos,
                        RightAlign,
                        AttachOnly )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
MenuID	ID of the ContextMenu entity to display.
MenuStruct	A dynamic array containing the executable structure of the menu.
xPos	Client area X-coordinate where the user clicked.
yPos	Client area Y-coordinate where the user clicked.
RightAlign	TRUE$ if the menu is to be right-aligned to the object.
AttachOnly	If TRUE$ then the TRACKPOPUPMENU method is called with the AttachOnly flag so that the menu is parsed, created and attached, but not displayed. *This argument should not be changed by a CONTEXTMENU event handler*.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

This event has a system-level handler which performs the following tasks:

- Calling an CONTEXTMENU quick event, if defined.
- Displaying the context menu via the TRACKPOPUPMENU method.

A CONTEXTMENU quick event handler can prevent the context menu from being displayed by setting the Event Status flag to TRUE$.

Equates constants for working with menu structures can be found in the PS_MENU_EQUATES insert record.

For more information on this event and context menu processing in general please refer to the Windows documentation regarding the WM_CONTEXTMENU window message on the Microsoft website.

Example

```
Function CONTEXTMENU( CtrlEntID, CtrlClassID, MenuID, MenuStruct, xPos, yPos, |
                                                      RightAlign, AttachOnly )

  // Example CONTEXTMENU event code - check to see which clipboard items should
  // be enabled
  $Insert PS_Menu_Equates
  $insert PS_Equates

  EditState = Get_Property( CtrlEntID, "EDITSTATEFLAGS" )

  xCount = FieldCount( MenuStruct, @Vm )
  For X = 5 to XCount
     If ( MenuStruct<0,X>[1,1] == "@" ) Then
        Null ; // Ignore - it's an imagelist header
     End Else
        If ( MenuStruct<0,X,MENUPOS_TYPE$> == MENUTYPE_ITEM$ ) Then
           ItemName    = MenuStruct<0,X,MENUPOS_NAME$>[-1,"B."]
           DisableItem = FALSE$
           Begin Case
              Case ( ItemName = "UNDO" )
                 DisableItem = ( EditState<PS_ESF_CANUNDO$> != TRUE$ )
              Case ( ItemName = "CUT" )
                 DisableItem = ( EditState<PS_ESF_CANCUT$> != TRUE$ )
              Case ( ItemName = "COPY" )
                 DisableItem = ( EditState<PS_ESF_CANCOPY$> != TRUE$ )
              Case ( ItemName = "PASTE" )
                 DisableItem = ( EditState<PS_ESF_CANPASTE$> != TRUE$ )
           End Case

           If ( DisableItem ) Then
              MenuStruct<0,X,MENUPOS_GREY$> = TRUE$
           End
        End
     End
  Next

Return TRUE$
```

See Also

Common GUI CONTEXTMENU property, Common GUI SHOWMENU method, Common GUI TRACKPOPUPMENU method, Common GUI INITCONTEXTMENU event, Common GUI MENU event, ContextMenu stored procedure.

DROPFILES event

Description

Occurs when files are dragged from the Windows Explorer and dropped onto an object.

Syntax

```
bForward = DROPFILES( CtrlEntID,
                      CtrlClassID,
                      FileList,
                      xDrop,
                      yDrop )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
FileList	@Vm-delimited list of files names that have been dropped onto the object.
xDrop	Client X-coordinate where the files were dropped onto the object.
yDrop	Client Y-coordinate where the files were dropped onto the object.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

This event will only be fired if the object's ACCEPTDROPFILES property is TRUE$.

There is no system-level event handler for DROPFILES.

For more information on this event please refer to the Windows documentation regarding the WS_EX_ACCEPTFILES extended window style and the WM_DROPFILES message on the Microsoft website.

Example

```
Function DROPFILES( CtrlEntID, CtrlClassID, FileList, xDrop, yDrop )

  // Example DROPFILES event for a ListBox called LST_WEBFILES that accepts
  // htm, html, js or css filenames and loads them into its list of items

  $Insert Logical

  // Clear out any current items
  Call Set_Property_Only( CtrlEntID, "LIST", "" )

  FileCount = FieldCOunt( FileList, @Vm )
  For FileCtr = 1 To FileCount

     FileName = FileList<FileCtr>

     // Check the extension
     FileExt  = FileName[-1,"B."]
     Convert @UPPER_CASE To @lower_case In FileExt

     Locate FileExt In "htm,html,js,css" Using "," Setting Pos Then
        // It's OK so add it to the ListBox
        Call Exec_Method( CtrlEntID, "INSERT", -1, FileName )
     End

  Next

Return TRUE$
```

See Also

Common GUI ACCEPTDROPFILES property.

GOTFOCUS event

Description

Occurs when a control receives the input focus.

Syntax

```
bForward = GOTFOCUS( CtrlEntID, CtrlClassID, PrevFocusID )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the control receiving the event.
CtrlClassID	Type of object receiving the event.
PrevFocusID	Name of the control that previously had the input focus on the same parent form. (i.e. the form's GOTFOCUSCONTROL property). This parameter can be null.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

This event has a system-level handler which performs the following tasks:

- If the control is an EDITTABLE automatic row insertion is processed if needed.
- Check if QBF Mode is on – if so then prevent the next steps from executing.
- Update the control's GOTFOCUSVALUE property.
- Process the DEFVALUE property.

Setting input focus to a control by using its FOCUS property will not trigger this event, while using the SYSTEM FOCUS property will.

For more information on this event please refer to the Windows documentation regarding WM_SETFOCUS window message and the SetFocus function on the Microsoft website.

Example

```
Function GOTFOCUS( CtrlEntID, CtrlClassID, PrevFocusID )

  // Example GOTFOCUS event script - skip the current control if the
  // value of another is null

  RetVal  = TRUE$
  DataVal = Trim( Get_Property( @Window : ".EDL_DATA", "TEXT" ) )

  If BLen( DataVal ) Then
     // All good, proceed as normal
     Null
  End Else
     // Find the NEXT control in the Tab order and move to
     // that

     NextCtrl = Get_Property( CtrlEntID, "NEXT" )
     Call Set_Property( "SYSTEM", "FOCUS", NextCtrl )

     // Stop the system-level gotfocus event...
     RetVal = FALSE$

  End

Return RetVal
```

See Also

Common GUI CANGETFOCUS property, Common GUI DEFVALUE property, Common GUI FOCUS property, Common GUI GOTFOCUSVALUE property, SYSTEM FOCUS property, WINDOW GOTFOCUSCONTROL property, Common GUI LOSTFOCUS event, WINDOW ACTIVATED event, WINDOW INACTIVATED event.

HELP event

Description

Occurs when the user requests online help for an object.

Syntax

```
bForward = HELP( CtrlEntID, CtrlClassID, ItemID, MouseX, MouseY )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
ItemID	Specifies the Item that was the source of the help request. This may not be the same as the control that is handling it.
MouseX	If the HELP event was raised from a WM_HELP message this parameter contains the X (Left) coordinate of the mouse in screen coordinates.
MouseY	If the HELP event was raised from a WM_HELP message this parameter contains the Y (Top) coordinate of the mouse in screen coordinates.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

The system-level event handler uses the SYSMSG event to display a "No Help Available" message.

If a QuickEvent is defined for this event the system level event handler will not execute.

For more information on this event please refer to the Windows documentation regarding the WM_HELP window message on the Microsoft website.

Example

```
Function HELP( CtrlEntID, CtrlClassID, ItemID, MouseX, MouseY )

   // If the control is data bound use the help from the dictionary
   $Insert Dict_Equates
   $insert Msg_Equates
   $Insert Logical

   RetVal = TRUE$

   ColName   = Get_Property( CtrlEntID, "COLUMN" )
   TableName = Get_Property( CtrlEntID, "TABLE" )

   If Index( ColName, @Svm, 1 ) Then
      // Assume EditTable - get the current column index.
      CaretPos  = Get_Property( CtrlEntID, "CARETPOS" )
      ColName   = ColName<0,0,CaretPos<1>>
      TableName = TableName<0,0,CaretPos<1>>
   End

   If BLen( ColName ) Then
      HelpText = Xlate( "DICT." : TableName, ColName, DICT_DESC$, "X" )
      Convert @VM:@Tm to "||" in HelpText

      If BLen( HelpText ) Then
         MsgArray = ""
         MsgArray<MTEXT$>     = HelpText
         MsgArray<MICON$>     = "*"
         MsgArray<MJUST$>     = "C"
         MsgArray<MCPATION$> = ColName : " help "

         Call Msg( @Window, MsgArray )

         RetVal = FALSE$

      End

   End

Return RetVal
```

See Also

Common GUI SHOWHELP method.

HSCROLL event

Description

Occurs when an object is scrolled horizontally.

Syntax

```
bForward = HSCROLL( CtrlEntID, CtrlClassID, Value )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
Value	Specifies the position that object has been scrolled to.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

There is no system-level event handler for HSCROLL.

For more information on this event please refer to the Windows documentation regarding the WM_HSCROLL window message on the Microsoft website.

Example

```
Function HSCROLL( CtrlEntID, CtrlClassID, Value )

  // Example HSCROLL event for an UPDOWN control to update the LBL_VALUE
  // STATIC control with the current position when an arrow is clicked.
  $Insert Logical

  Call Set_Property_Only( @Window : ".TXT_VALUE", "TEXT", Value )

Return TRUE$
```

See Also

Common GUI HPOSITION property, Common GUI SCROLLBARS property, Common GUI VPOSITION property, Common GUI VSCROLL event.

INITCONTEXTMENU event

Description

Fired in response to a right click (actually a WM_CONTEXTMENU) message from Windows) and initializes the object for displaying an attached context menu

Syntax

```
bForward = INITCONTEXTMENU( CtrlEntID,
                            CtrlClassID,
                            MenuID,
                            xPos,
                            yPos,
                            RightAlign,
                            CustomStruct )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
MenuID	ID of the ContextMenu entity to display.
xPos	Client area X-coordinate where the user clicked.
yPos	Client area Y-coordinate where the user clicked.
RightAlign	TRUE$ if the menu is to be right-aligned to the object.
CustomStruct	Structure for menus constructed at runtime via the ContextMenu stored procedure.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

This event has a system-level handler which performs the following tasks:

- Calling the Yield stored procedure to clear any pending events
- Calling an INITCONTEXTMENU quick event, if defined
- Reading the context menu definition from the repository (if not already cached)
- Converting the structure into v10+ format if needed

- Compiling it into an "executable" format and caching it
- Firing the subsequent CONTEXTMENU event

The intent of the INITCONTEXTMENU event is as a place for the Presentation Server to begin the context menu process, so as such it is a system tool – it is not really intended that developers have to interact with this event, although there's nothing to prevent this if desired.

An INITCONTEXTMENU quick event handler can prevent the context menu from being displayed by setting the Event Status flag to TRUE$.

For more information on this event and context menu processing in general please refer to the Windows documentation regarding the WM_CONTEXTMENU window message on the Microsoft website.

Example

N/a.

See Also

Common GUI CONTEXTMENU property, Common GUI SHOWMENU method, Common GUI TRACKPOPUPMENU method, Common GUI CONTEXTMENU event, Common GUI MENU event, ContextMenu stored procedure.

LOSTCAPTURE event

Description

Occurs when an object capturing mouse messages releases the capture.

Syntax

```
bForward = LOSTCPATURE( CtrlEntID, CtrlClassID, CaptureID )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
CaptureID	Name of the object that has released the capture.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

There is no system-level event handler for LOSTCAPTURE.

For more information on this event please refer to the Windows documentation regarding the WM_CAPTURECHANGED window message and the ReleaseCapture function on the Microsoft website.

Example

```
Function LOSTCAPTURE( CtrlEntID, CtrlClassID,  CaptureID )

   // Example LOSTCAPTURE - check to see if the capture object is being dragged,
   // and if so clear it (see examples for the BUTTONDOWN and BUTTONUP events )

   If Get_Property( CaptureID, "@DRAGGING" ) Then
      Call Set_Property_Only( CaptureID, "@DRAGGING", FALSE$ )
   End

Return TRUE$
```

See Also

Common GUI CURSOR property, Common GUI MOUSECAPTURED property, Common GUI BUTTONDOWN event, Common GUI BUTTONUP event, Common GUI MOUSEMOVE event.

LOSTFOCUS event

Description

Occurs when a control loses the input focus.

Syntax

```
bForward = LOSTFOCUS( CtrlEntID, CtrlClassID, Flag, FocusID )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the control receiving the event.
CtrlClassID	Type of object receiving the event.
Flag	Numeric value specifying why the control lost the focus: 0 - The focus has moved to a control on another (OI or non-OI) form. 1 - The focus has moved to another control on the same form. 2 - The LOSTFOCUS event was raised from a MENU event.
FocusID	Name of the control that has received the focus. This parameter can be null if the focus has moved to a non-OI object.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

This event has a system-level handler which performs the following tasks:

- Verifies that the control contains data if the REQUIRED property is TRUE$.
- Validates the data in the control if the VALID property is set.
- Updates the parent form's SAVEWARN property the control's data has changed.

Setting input focus to a control by using its FOCUS property will not trigger this event, while using the SYSTEM FOCUS property will.

For more information on this event please refer to the Windows documentation regarding WM_KILLFOCUS window message and the SetFocus function on the Microsoft website.

Example

```
Function LOSTFOCUS( CtrlEntID, CtrlClassID, Flag, FocusID )

   // Example LOSTFOCUS event script - check to see if the contents of the current
   // control matches a know code - if not report the error via a message and reset
   // the focus
   $Insert Logical

   RetVal = TRUE$
   Code   = Get_Property( CtrlEntID, "TEXT" )

   Locate Code In "AX,RY,SE" using "," Setting Pos Else

      Call Msg( @Window, "Invalid Code" )

      // Reset the focus without triggering any more focus events

      Call Set_Property( "SYSTEM", "BLOCKEVENTS", TRUE$ )
      Call Set_Property( "SYSTEM", "FOCUS", CtrlEntID )
      Call Set_Property( "SYSTEM", "BLOCKEVENTS", FALSE$ )

      RetVal = FALSE$

    End

 Return RetVal
```

See Also

Common GUI CANGETFOCUS property, Common GUI FOCUS property, Common GUI GOTFOCUSVALUE property, SYSTEM FOCUS property, Common GUI LOSTFOCUS event, WINDOW ACTIVATED event, WINDOW INACTIVATED event.

MENU event

Description

Occurs when a menu item is selected from a context menu or a form dropdown menu.

Syntax

```
bForward = MENU( CtrlEntID, CtrlClassID )
```

Parameters

Name	Description

CtrlEntID	Fully qualified name of the menu object receiving the event.
	
	For a form menu item CtrlEntID has the format:
	
	`<formName> ".MENU." <itemName>`
	
	E,g.
	
	`RTI_IDE.MENU.FILE.SAVE`
	`RTI_IDE.MENU.FILE.CLOSE`
	
	For a context menu CtrlEntID has the format:
	
	`<controlName> ".CONTEXTMENU." <itemName>`
	
	E.g.
	
	`RTI_OUTPUT_PANEL.LST_ITEMS.CONTEXTMENU.COPY`
	`RTI_OUTPUT_PANEL.LST_ITEMS.CONTEXTMENU.SELECT_ALL`

CtrlClassID	Type of object receiving the event.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

There is no system-level event handler for MENU.

Context menus cannot have Event Script handlers, they may only have QuickEvent handlers defined.

Example

```
// Example: Assume this is a MENU handler subroutine within in a Commuter Module

OnMenu:
   // Is this a Form menu or a context menu item

   CtrlID = Field( CtrlEntID, ".", 2 )

   OwnerID = CtrlID[1,"."]
   MenuID  = CtrlID[Col2()+1,\00\]

   If ( OwnerID == "MENU" ) then
      // It's a form menu
      OwnerID = @Window

   End Else
      OwnerID = CtrlID[1,"."]
      MenuID  = CtrlID[Col2()+1,\00\]

      If ( MenuID[1,"."] == "CONTEXTMENU" ) Then

         MenuID = MenuID[Col2()+1,\00\]
         CtrlID = @Window : "." : OwnerID

         Begin Case
            Case ( OwnerID = "LST_MENUDES_ITEMS" )
               GoSub LstMenuDesItems_OnMenu
         End Case
      End
   End

Return TRUE$

LstMenuDesItems_OnMenu:

   Begin Case
      Case ( MenuID = "SELECT_ALL" )
         Call Exec_Method( CtrlID, "SELECT_ALL" )
   End Case

Return
```

See Also

Common GUI CONTEXTMENU property, Common GUI SHOWMENU method, Common GUI TRACKPOPUPMENU method, Common GUI INITCONTEXTMENU event, ContextMenu stored procedure.

MOUSEMOVE event

Description

Occurs when one of the following mouse messages is sent to an object by Windows:

- Enter – The mouse cursor has begun moving over the client area of an object.
- Move – The mouse cursor is moving over an object's client area.
- Hover – The mouse cursor is hovering over a spot in an object's client area.
- Leave – The mouse cursor has left the client area of an object.

Syntax

```
bForward = MOUSEMOVE( CtrlEntID,
                      CtrlClassID,
                      MouseEvent,
                      MouseX,
                      MouseY,
                      CtrlKey,
                      ShiftKey,
                      MouseButton )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
MouseEvent	Specifies the type of MOUSEMOVE event. Can be one of the following: "ENTER" "MOVE" "HOVER" "LEAVE"
MouseX	Client X-coordinate (Left) of the mouse cursor.
MouseY	Client Y-coordinate (Top) of the mouse cursor.
CtrlKey	TRUE$ if the Ctrl key is pressed down, or FALSE$ otherwise.
ShiftKey	TRUE$ if the Shift key is pressed down, or FALSE$ otherwise.
MouseButton	Integer specifying which mouse button pressed down : "0" - Left button "1" - Right button "2" - Middle button If no buttons are pressed down this argument is null.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

For a "LEAVE" MouseEvent the coordinates, keys and button parameters are always null.

If the mouse is captured by an object then all mouse events are directed to the capturing object, *even if the mouse is moving over a different object*.

There is no system-level event handler for MOUSEMOVE.

Equated constants for the MOUSEMOVE event may be found in the PS_EQUATES insert record.

For more information on this event please refer to the Windows documentation regarding the WM_MOUSEMOVE, WM_MOUSEHOVER and WM_MOUSELEAVE window messages on the Microsoft website.

Example

```
Function MOUSEMOVE( CtrlEntID, CtrlClassID, MouseEvent, MouseX, MouseY, |
                                        CtrlKey, ShiftKey, MouseButton )

   // Update the Form text with the current position of the mouse over the current
   control
   $Insert PS_Equates
   $Insert Logical

   TrackText = ""

   Begin Case
      Case ( MouseEvent = MMEVENT_ENTER$ )
         TrackText = "The mouse is in the house"
      Case ( MouseEvent = MMEVENT_HOVER$ )
         TrackText = "And.... what are you waiting for?"
      Case ( MouseEvent = MMEVENT_LEAVE$ )
         TrackText = "The mouse has left the building"
      Case OTHERWISE$
         TrackText = "Mouse is on the move"
   End Case

   If ( MouseEvent = MMEVENT_LEAVE$ ) Else
      TrackText := " @(" : MouseX : "," : MouseY : ")"
   End

   Call Set_Property( @Window, "TEXT", TrackText )

 Return RetVal
```

See Also

Common GUI CURSOR property, Common GUI MOUSECAPTURED property, Common GUI BUTTONDOWN event, Common GUI BUTTONUP event, Common GUI LOSTCAPTURE event.

NOTES event

Description

Occurs when the user requests to see QuickHelp for an object.

Syntax

```
bForward = NOTES( CtrlEntID, CtrlClassID )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

The system-level event handler performs the following:

- Checks to see if there is a QuickHelp (APPNOTE) entity with the same entity ID as the current control, and If so this is executed.
- If not a new, blank QuickHelp entity is created with the name entity as the current control and is then executed, allowing the user to begin entering their own notes if they wish.

If a QuickEvent is defined for this event the system level event handler will not execute.

Example

```
Function NOTES( CtrlEntID, CtrlClassID )

   // If the control is data bound use the help from the dictionary
   $Insert Dict_Equates
   $insert AppNote_Equates
   $Insert Logical

   RetVal = TRUE$

   ColName   = Get_Property( CtrlEntID, "COLUMN" )
   TableName = Get_Property( CtrlEntID, "TABLE" )

   If Index( ColName, @Svm, 1 ) Then
      // Assume EditTable - get the current column index.
      CaretPos  = Get_Property( CtrlEntID, "CARETPOS" )
      ColName   = ColName<0,0,CaretPos<1>>
      TableName = TableName<0,0,CaretPos<1>>
   End

   If BLen( ColName ) Then
      HelpText = Xlate( "DICT." : TableName, ColName, DICT_DESC$, "X" )
      Convert @VM:@Tm to "||" in HelpText
      Swap "|" with "<br/>" in HelpText

      If BLen( HelpText ) Then

         NoteArray<AHTMLTEXT$> = HelpText
         NoteArray<AREADONLY$> = TRUE$
         NoteArray<ATITLE$>    = ColName : " help "

         Call AppNote( @Window, "", NoteArray, "" )

         RetVal = FALSE$

      End

   End

Return RetVal
```

See Also

Common GUI SHOWNOTES method, AppNote stored procedure

OMNIEVENT event

Description

This is generic user-defined event that is only triggered by developer code – the system itself will not trigger it. It is intended to be a simple way to add user defined events to an application.

Syntax

```
bForward = OMNIEVENT( CtrlEntID, CtrlClassID, Message, Param1, Param2, |
                      Param3, Param4, Param5, Param6, Param7, Param8 )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
Message	Identifies the function within the OMNIEVENT to call. This is used to branch to different sub-handlers with the OMNIEVENT handler and is determined by the developer,
Param1	Generic parameter to pass to the event. Its meaning is specific to the Message being processed.
Param2	Generic parameter to pass to the event. Its meaning is specific to the Message being processed.
Param3	Generic parameter to pass to the event. Its meaning is specific to the Message being processed.
Param4	Generic parameter to pass to the event. Its meaning is specific to the Message being processed.
Param5	Generic parameter to pass to the event. Its meaning is specific to the Message being processed.
Param6	Generic parameter to pass to the event. Its meaning is specific to the Message being processed.
Param7	Generic parameter to pass to the event. Its meaning is specific to the Message being processed.
Param8	Generic parameter to pass to the event. Its meaning is specific to the Message being processed.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

There is no system-level event handler for OMNIEVENT.

Example

```
////////////////////////////////////////////////////////////////////////////
// Example: Assume a READ operation on an MDiChild form wants to run an event
// on its parent MDI Frame to notify it that a record has been opened so that
// it can synchronize its menu items and buttons etc.

////////////////////////////////////////////////////////////////////////////
// The MDI Child READ handler uses code like this to invoke an OMNIEVENT on
// its parent, passing it a message called "CHILDREAD" along with the the ID
// it has just read and its own name...

MDIFrame = Get_Property( @Window, "MDIFRAME" )
RecordID = Get_Property( @Window, "ID" )

Call Exec_Method( MDIFrame, "DISPATCHEVENT", "OMNIEVENT", "CHILDREAD", |
                                                    @Window, RecordID )

////////////////////////////////////////////////////////////////////////////
// And in the MDIFrame an OMNIEVENT handler is defined to process this:

Function OMNIEVENT( CtrlEntID, CtrlClassID, Message, Param1, Param2, Param3, |
                    Param4, Param5, Param6, Param7, Param8 )

   $Insert Logical

   Begin Case
      Case ( Message == "CHILDREAD" )

         ChildID  = Param1
         RecordID = Param2

         // Do whatever ....

   End Case

Return TRUE$
```

See Also

N/a.

OPTIONS event

Description

Occurs when the user requests to see available data-entry options for an object.

Syntax

```
bForward = OPTIONS( CtrlEntID, CtrlClassID )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

The system-level event handler performs the following:

- Checks to see if the targeted object is a dropdown control, such as a ComboBox, and if so displays its drop-down list.
- Otherwise the SYSMSG event is triggered to display a "No Options Available" message (If a QuickEvent is defined for this event the SYSMSG event will not be fired).

Example

```
Function OPTIONS( CtrlEntID, CtrlClassID )

   // Display a popup of PART records for the user to select from
   $Insert Popup_Equates
   $Insert Logical

   PartID = Popup( @Window, "", "PARTS_LIST" )
   If Blen( PartID ) Then
      Call Set_Property( CtrlEntID, "DEFPROP", PartID )
   End

Return FALSE$
```

See Also

Common GUI SHOWOPTIONS method.

REQUIRERR event

Description

Occurs when a control with a required property has no data and OpenInsight intends to display a warning message to the user.

Syntax

```
bForward = REQUIRERR( CtrlEntID, CtrlClassID, RequireInfo )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.

RequireInfo	Depends on the object type.
	
	If the type is a WINDOW (form) this parameter contains an @Fm delimited list of controls that are missing data.
	
	Each entry in the list is an @Vm-delimited array of control information:
	
	`<0,1> Name of the control missing data`
	`<0,2> Column index of missing data (for EditTables)`
	`<0,3> Row index of missing data (for EditTables)`
	
	(If the list if blank the system scans the form to build this list itself)
	
	If the type is not a WINDOW then this parameter may be the name of a single control that is missing data. This defaults to CtrlEntID.
	
	In both cases the system attempts to resolve control names to something more meaningful to an end user:
	
	• If the control is an EditTable the heading text for the appropriate column is used in place of a control name, or
	
	• If the control is data-bound then the name is replaced with the display heading text (DICT_DISPLAY$) from the associated dictionary column.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

By default the check to see if the control has data is performed during the LOSTFOCUS event (and POSCHANGED for EditTables) – if the check is failed a message is displayed to the user and the input focus is returned to the control (this is handled by the REQUIRERR event). However, this behavior can be too intrusive for many applications and may be modified by using the parent Form's REQUIREONSAVE property, where the check is only made when the data is about to be saved.

This event has a system-level handler which performs the following tasks:

- Calling a REQUIRERR quick event, if defined.
- Displaying a default "Required" error message to the user.
- Reset the focus to the problem control.

A REQUIRERR quick event handler can prevent the system message from being displayed by setting the Event Status flag to TRUE$.

Example

```
Function REQUIRERR( CtrlEntID, CtrlClassID, RequireInfo )

  // Example - Process the list to use a "NameCapped" version of the control name
  // the text for any non-edit table controls
  Declare Function NameCap
  $Insert Logical

  If BLen( RequireInfo ) Else RequireInfo = CtrlEntID

  ReqCount = FieldCount( RequireInfo, @Fm )
  For ReqCtr = 1 To ReqCount
     If ( RequireInfo<ReqCtr,2> ) Then
        // Assume an EditTable - let the default handler use the
        // column header
        Null
     End Else
        CtrlName = Field( RequireInfo<ReqCtr,1>, ".", 2, 999 )
        Convert "_" To " " In CtrlName
        RequireInfo<ReqCtr,1> = NameCap( CtrlName )
     End
  Next

Return TRUE$
```

See Also

Common GUI REQUIRED property, Common GUI LOSTFOCUS event, Common GUI EDITTABLE POSCHANGED event, WINDOW REQUIREONSAVE property.

SYSMSG event

Description

Occurs when OpenInsight attempts display a standard system message to the user.

Syntax

```
bForward = SYSMSG( CtrlEntID, CtrlClassID, MsgCode, CancelFlag, StatCode )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
MsgCode	Numeric code of the message to display: 1. SaveWarn* : Data may be lost, save first warning 2. DeleteWarn* : Verify delete message 3. <unused> : N/a 4. <unused> : N/a 5. ReadErr* : Error reading row 6. ReadSubErr* : Error reading subsidiary row 7. LockErr* : Error locking row 8. <unused> : N/a 9. WriteLockErr* : No lock for writing 10. NoHelpInfo : No help available 11. NoOptInfo : No options available 12. NewRowInfo* : New row informational (null message) 13. <unused> : N/a 14. QBFInitOff* : Can't execute QBF - not in QBFInit mode 15. NullKeyErr* : Null key - cannot read or write 16. NoLockErr* : Row not locked - cannot save or delete 17. OverWrite* : Record exists on write-without-read 18. DeleteErr* : Error deleting row 19. <unused> : N/a 20. ChangeWarn* : Data may be lost warning – no save option 21. SaveWarnInfo* : SAVEWARN changed informational (null message) Any other value in this argument is treated as an MSG repository entity name. Items marked (*) are only handled for data-bound forms.
CancelFlag	Boolean "cancel flag". By default this is FALSE$, but the system event handler will set this the TRUE$ if the user attempts to cancel the operation.
StatCode	If passed this is assumed to be one or more status codes as retrieved from the Get_Status SSP. These will be translated into text format and displayed as the message text.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

The system-level event handler for SYSMSG displays the requested system message.

In previous versions of OpenInsight the Required and Validation Error messages were handled by the SYSMSG event. In version 10.1 these have been moved to their respective system event handlers (REQUIRERR and VALIDERR respectively) to conform to the standard event flow.

Example

```
Function SYSMSG( CtrlEntID, CtrlClassID, MsgCode, CancelFlag, Auxiliary )

  // Example - if we get a SAVEWARN notification then
  // put it in the System Monitor as a tracing program

  $Insert PS_Equates
  $Insert Logical

  Begin Case
     Case( MsgCode = SYSMSG_SAVEWARNINFO$ )

        SaveWarn = Get_Property( CtrlEntID, "SAVEWARN" )
        LogText  = "* SAVEWARN -> " : SaveWarn |
                 : " :" : CtrlEntID            |
                 : " [" : Auxiliary : "]"
        Call Exec_Method( "SYSTEMMONITOR", "OUTPUT", LogText )

  End Case

Return TRUE$
```

See Also

Common GUI HELP event, Common GUI OPTIONS event, Common GUI REQUIREER event, Common GUI VALIDERR event, WINDOW SAVEWARN property, WINDOW CLEAR event, WINDOW DELETE event, WINDOW READ event, WINDOW WRITE event.

TIMER event

Description

Occurs when the amount of time specified by the TIMER property has passed.

Syntax

```
bForward = TIMER( CtrlEntID, CtrlClassID )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

There is no system level event handler for the TIMER event.

In previous versions of OpenInsight only forms supported a TIMER property and a TIMER event. In this version all GUI objects support them.

For more information on Windows Timers please refer to the documentation regarding the SetTimer and KIllTimer functions, and the WM_TIMER message on the Microsoft Website.

Example

```
Function TIMER( CtrlEntID, CtrlClassID )

   // Check to see of a record is present in a file - if so stop the TIMER
   $Insert Logical

   Open "PENDING_TRANSFERS" To hPendingTransfers Then
      Read TransferRec From hPendingTransfers, "%DONE%" Then
         Call Set_Property_Only( CtrlEntID, "TIMER", 0 ) ; // Stop
      End
   End

Return TRUE$
```

See Also

Common GUI TIMER property, SYSTEM IDLEPROC property, SYSTEM ADDIDLEPROC method.

VALIDERR event

Description

Occurs when a control fails a data validation check and OpenInsight intends to display a warning message to the user.

Syntax

```
bForward = VALIDERR( CtrlEntID, CtrlClassID, ValidInfo )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
ValidInfo	Contains an @Fm-delimited array of validation and pattern information: <1> The data that caused the validation failure <2> The validation pattern(s) that were used to check the data <3> Custom validation message from the VALIDMSG property. (The custom validation message can be null to use the default text)

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

By default a check to see if the control passes a validation check is performed during the LOSTFOCUS event (and POSCHANGED event for EditTables) – if the check is failed the VALIDERR event is raised to display a warning.

This event has a system-level handler which performs the following tasks:

- Calling an VALIDERR quick event, if defined.
- Displaying a default "Validation failed" error message to the user.
- Reset the focus to the problem control.

A VALIDERR quick event handler can prevent the system message from being displayed by setting the Event Status flag to TRUE$.

Example

```
Function VALIDERR( CtrlEntID, CtrlClassID, ValidInfo )

   // Example VALIDERR event - Change the validation message and
   // let the system handler display it
   $Insert Logical

   ValidInfo<3> = 'Bad news – "%1%" doesn't compute - computer says No'

   // And pass it into the system handler to display

 Return TRUE$
```

See Also

Common GUI VALID property, Common GUI VALIDMSG property, Common GUI LOSTFOCUS event, EDITTABLE POSCHANGED event.

VSCROLL event

Description

Occurs when an object is scrolled vertically.

Syntax

```
bForward = VSCROLL( CtrlEntID, CtrlClassID, Value )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
Value	Specifies the position that object has been scrolled to.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

There is no system-level event handler for VSCROLL.

For more information on this event please refer to the Windows documentation regarding the WM_VSCROLL window message on the Microsoft website.

Example

```
Function VSCROLL( CtrlEntID, CtrlClassID, Value )

  // Example VSCROLL event for an UPDOWN control to update the LBL_VALUE
  // STATIC control with the current position when an arrow is clicked.
  $Insert Logical

  Call Set_Property_Only( @Window : ".TXT_VALUE", "TEXT", Value )

Return TRUE$
```

See Also

Common GUI HPOSITION property, Common GUI SCROLLBARS property, Common GUI VPOSITION property, Common GUI HSCROLL event.

WINMSG event

Description

Occurs when a qualified window message is received by an object.

Syntax

```
bForward = WINMSG( CtrlEntID, CtrlClassID, hWnd, Message, wParam, lParam )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of object receiving the event.
hWnd	Handle of the object that received the message.
Message	Integer specifying the message that has been received.
wParam	Message-specific integer value.
lParam	Message-specific integer value.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

There is no system-level event handler for WINMSG.

Equates for the core Window messages can be found in the MSWIN_WINDOWMESSAGE_EQUATES insert record. Equates for the control-specific message like ComboBoxes and EditLines can be found in their respective MSWIN_<controltype>_EQUATES records.

For more information window messaging please refer to the Windows documentation on the Microsoft website.

Example

```
Function WINMSG( CtrlEntID, CtrlClassID, hWnd, Message, wParam, lParam )

   // Assume the WM_PARENTNOTIFY message has been qualified and we are listening
   // for a child control being created
   Declare Function RTI_LOWORD
   $Insert MSWin_WindowMessage_Equates
   $Insert Logical

   Begin Case
      Case ( Message = WM_PARENTNOTIFY$ )
         // From the MS docs:
         //
         // The LOWORD of wParam is the notification code
         // The HIWORD of wParam is the
         // lParam contains the hWnd of the control that has
         // been created

         Code = RTI_LOWORD( wParam )

         Begin Case
           Case ( Code = WM_CREATE$ )
              // A child was created - if it's an OI object we
              // can get its ID.
              CtrlID = Exec_Method( "SYSTEM", "OBJECTID", lParam )

              If BLen( CtrlID ) Then
                 // An OI Control was created...
              End

         End Case

   End Case

Return TRUE$
```

See Also

Common GUI QUALFIEDWINMSGS property, Common GUI POSTWINMSG method, Common GUI QUALIFYWINMSG method, Common GUI SENDWINMSG method,

.