

# WINDOW (Form) object

---

The WINDOW object (more commonly referred to as a Form) is the core object type used to build an OpenInsight desktop user interface. It represents a visual surface on which other GUI controls (such as EditLines, Listboxes, Checkboxes and so on) are placed and arranged which allow the user to interact with an application.



## Developer Notes

1. Forms support an IMAGE sub-object which is used to specify their background image.
2. Forms support a MENU sub-object as an interface to the form's menu bar (if present).

## WINDOW Properties

The WINDOW object supports the following properties in addition to the Common GUI Object and Container Object properties, except where noted below:

Name	Description
<b>ACCELERATORFORM</b>	Specifies the name of form that menu accelerator keystrokes on the current form are redirected to.
<b>ACTIVE</b>	Specifies if the form is active.
<b>ALLOWFORMSTATEEVENTS</b>	Specifies if a form triggers the FORMSTATECHANGED and MDICHILDSTATECHANGED events.
<b>ALLOWSELFLOCKS</b>	Specifies if a data-bound form ignores locks placed on the same record.
<b>ALLOWSEQKEYRESET</b>	Specifies if a user is allowed to reset a data-bound form's sequential key counter.
<b>ARRANGEICONS</b>	Specifies if an MDI Frame form automatically arranges its minimized MDI child forms.
<b>ATRECORD</b>	Gets or sets data associated with the primary table of a data-bound form and updates data-bound controls.
<b>AUTOCOMPOSITED</b>	Specifies if the form uses system double-buffering during sizing operations.
<b>COMMUTERMODULE</b>	Contains the name of the stored procedure used to handle events for the specified form.
<b>CTRLMAP</b>	Returns a list of controls hosted by the specified form.
<b>DESTROYFLAG</b>	Specifies if the form is flagged for destruction during SYSTEM QUERYEND processing.
<b>DPI</b>	Returns the current DPI (dots-per-inch) setting for the specified form.
<b>DWMANIMATION</b>	Specifies if the form uses Desktop Window Manager (DWM) animations.
<b>FIRSTFOCUS</b>	Returns the name of the first control that receives the input focus when the specified form is created.
<b>FORMBORDERSTYLE</b>	Specifies the border style of a form.
<b>GOTFOCUSCONTROL</b>	Returns the last control on the form with a GOTFOCUS event handler that had the input focus.
<b>HELPBUTTON</b>	Specifies if a help button is displayed on a form's caption bar.
<b>HIDEEFFECT</b>	Specifies the animation used with a form's HIDE method.
<b>ICON</b>	Specifies the name of the icon displayed in the form's caption bar.
<b>ID</b>	Returns the current row key associated with the primary table data-row in a data-bound form.
<b>INITIALFOCUS</b>	Returns the control that receives the input focus when the specified form is activated.
<b>INITIALPOSITION</b>	Specifies the starting location of the specified form.
<b>IOPTIONS</b>	Specifies the data-binding options for a form.

<b>LOADPREVALWAYS</b>	Specifies if the form's PREVROWVAL property is updated during a read operation as well as a write operation.
<b>LOCKCOORDINATION</b>	Specifies if table-lock coordination is used by the form in conjunction with record locking.
<b>LOCKTYPE</b>	Specifies the type of row locking used by a form.
<b>MAXIMIZEBUTTON</b>	Specifies if a maximize button is displayed on a form's caption bar.
<b>MDIACTIVE</b>	Activates an MDI child form or returns the name of the currently active MDI child form for the specified MDI frame form.
<b>MDIFRAME</b>	Returns the name of the parent MDI frame form for the specified MDI child form.
<b>MINIMIZEBUTTON</b>	Specifies if a maximize button is displayed on a form's caption bar.
<b>MULTIINSTANCE</b>	Specifies if multiple instances of the same form may be executed in the same session at runtime.
<b>NEWROW</b>	Returns TRUE\$ if the specified form has loaded a new (blank) data row.
<b>NOCLEARONWRITE</b>	Specifies if a data bound form clears its contents after a successful write operation.
<b>NUMERICCOMPARE</b>	Specifies the type of comparison the form write operation uses to determine if a column needs updating.
<b>OVERLAYICON</b>	Specifies a small icon that may be used to overlay the normal icon on a form's taskbar button.
<b>OWNER</b>	Gets or sets the owner of the specified form.
<b>PLACEMENTDATA</b>	Get or sets the show state and the restored, minimized, and maximized positions of the specified form.
<b>PREVROWVAL</b>	Returns the values that were held in the specified form's controls for a previously loaded data row.
<b>PROGRESSSTATE</b>	Sets the state of the current progress information on the specified form's taskbar button.
<b>PROGRESSVALUE</b>	Displays progress information on the specified form's taskbar button.
<b>QBFLIST</b>	Gets or sets the list of data keys used by the current QBF (Query-By-Form) session for the specified form.
<b>QBFPOS</b>	Gets or sets the index of the row to display when the specified form has a valid QBFLIST loaded.
<b>QBFSTATUS</b>	Returns the status of the QBF session for the specified form.
<b>QBFREADMODE</b>	Specifies if a form triggers the READ event when loading data during QBF processing.
<b>RECORD</b>	Gets or sets the cached copy of the data row associated with the specified form.
<b>REQUIREONWRITE</b>	Specifies when a form performs "required data" checks.
<b>RESTORESIZE</b>	Returns the position and size of the specified form in its non-maximized/minimized state.

<b>ROW</b>	Gets or sets data associated with the primary table of a data-bound form and updates data-bound controls.
<b>ROWLOCKED</b>	Returns TRUE\$ if the form has loaded and locked a data row.
<b>SAVEWARN</b>	Specifies if a data-bound form contains changed data that has not been saved.
<b>SCALEFACTOR</b>	Specifies the custom magnification factor for a form.
<b>SCALEUNITS</b>	Specifies how size and position coordinates are interpreted by a form.
<b>SHOWCAPTION</b>	Specifies if a form displays a caption bar.
<b>SHOWEFFECT</b>	Gets or sets the animation used with the SHOW method for the specified form.
<b>SIZINGMODE</b>	Specifies if a form can be resized with the mouse.
<b>STATUSLINE</b>	Identifies the control that receives "status" messages from stored procedures when the specified form is active.
<b>STYLESHEET</b>	Specifies the name of a form to use as a "styling template" when adding controls to a form at design-time.
<b>SUPPRESSSAVEWARN</b>	Specifies if a data-bound form checks the SAVEWARN property before clearing its contents or closing.
<b>SYSTEMMENU</b>	Specifies if a "System Menu" is allowed for the specified form.
<b>TABLE</b>	Returns the name of the primary table that the specified form is bound to.
<b>TASKBARBUTTON</b>	Returns a flag denoting if Windows has created a taskbar button for the specified form.
<b>TASKBARID</b>	Specifies a text string used to group or ungroup forms on the Windows taskbar.
<b>TRACKINGSIZE</b>	Specifies the minimum and maximum sizes that a user may resize a form to.
<b>TRANSLUCENCY</b>	Specifies the degree of transparency applied to a form when it is painted.
<b>TOPMOST</b>	Specifies if the form appears above all other non-TOPMOST forms in the system z-order.
<b>VISIBLE</b>	Specifies if a form is visible, hidden, maximized, or minimized.
<b>WRITEATRECORD</b>	Specifies how data set with the ATRECORD property is saved during a write operation.
<b>WRITEMODE</b>	Specifies how data set with the ROW property is saved during a write operation.

The following Common GUI Object properties are not supported:

- ALLPAGES
- AUTOSCALE
- AUTOSIZEHEIGHT
- AUTOSIZEWIDTH
- BOTTOMANCHOR
- COLUMN
- CONV
- DEFPROP
- DEFPROPPOS
- DEFPOSPROPID
- DEFPROPRAW
- DEFVALUE
- DESIGNSELECTED
- ECHO
- EDGESTYLE
- FORECOLOR
- GOTFOCUSVALUE
- IMAGELIST
- INVALUE
- NEXT
- ORIGARRAY
- ORIGINLABEL
- ORIGINLIST
- ORIGINVALUE
- PAGENUMBER
- PART
- POS
- PREVIOUS
- REQUIRED
- RIGHTANCHOR
- VALID
- VALIDMSG

## ACCELERATORFORM property

### Description

Gets or sets the name of a different form (with a menu) to redirect menu accelerator keystrokes to when they are detected on the current form.

### Property Value

This is string value containing the name of a running form instance. When an accelerator keystroke is detected for the current form it is sent to the other nominated form instance for processing.

### Property Traits

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

### Remarks

This property is useful for owned/child forms (e.g. dialog boxes) which traditionally do not have their own menu. Rather than creating and duplicating a hidden menu themselves (for common functions like Help and Options) they can simply redirect the accelerator keystrokes to their parent for processing instead.

### Example

```
// Example - in the CREATE event of a dialog box redirect the accelerator  
// keystrokes to the parent form.  
  
ParentForm = Get_Property( CtrlEntID, "PARENT" )  
Call Set_Property_Only( CtrlEntID, "ACCELERATORFORM", ParentForm )
```

### See Also

MENU object.

## ACTIVE property

### Description

Specifies if the form is active – i.e. if it is the top-level form that the user is currently working with.

### Property Value

This is a boolean value. It returns TRUE\$ if the form is the active form, or FALSE\$ otherwise.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get	No	No	No

### Remarks

This property is updated during the WM\_ACTIVATE window message processing. Please see the Microsoft website for more information on this message and on the topic of Keyboard Focus and Activation.

### Example

```
// Example - Determine if the current form is the active form.  
IsActive = Get_Property( @Window, "ACTIVE" )
```

### See Also

SYSTEM FOCUS property, WINDOW ACTIVATED event, WINDOW INACTIVATED event.

## ALLOWSELFLOCKS property

### Description

Specifies if a data-bound form ignores locks placed on the same row if it is loaded into other forms in the same PS instance at the same time.

### Property Value

This is a boolean value. It returns TRUE\$ if the form ignores "self-locks", or FALSE\$ otherwise.

### Property Traits

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

### Remarks

When a row is read into a form the system checks to see if it can place an exclusive lock on it first. If it cannot, the default behavior is to display a message to the user, warning them that they can view a read-only version of a row or abort the read attempt.

If the ALLOWSELFLOCKS property is TRUE\$ and the exclusive lock attempt fails, then no warning will be shown if the lock is held by *another form in the current PS instance* – in this case the row will be loaded and editing allowed as normal.

This facility is intended to allow forms to edit different parts of the same row at the same time without a locking conflict. For this activity to be safe forms that use this property *must* ensure that their controls do not bind to the same columns on different forms, otherwise this may result in unexpected data integrity issues due to the possibility of overwriting updates from other forms containing the same row.

### Example

```
// Example - Determine if the current form is allowing self-Locks
AllowSelfLocks = Get_Property( @Window, "ALLOWSELFLOCKS" )
```

### See Also

WINDOW IOOPTIONS property, WINDOW READROW method, WINDOW READ event.



## ALLOWFORMSTATEEVENTS property

### Description

Specifies if a form triggers the FORMSTATECHANGED and MDICHILDSTATECHANGED events.

### Property Value

This is a boolean value. It returns TRUE\$ if the form triggers the events, or FALSE\$ otherwise.

### Property Traits

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

### Remarks

This property is FALSE\$ for existing forms and must be enabled "manually" to preserve backwards compatibility. It is set to TRUE\$ by default for new forms.

### Example

```
// Example - Determine if the current form is allowing "FormState" events  
AllowFormStateEvents = Get_Property( @Window, "ALLOWFORMSTATEEVENTS" )
```

### See Also

WINDOW IOOPTIONS property, WINDOW FORMSTATECHANGED event WINDOW MDICHILDSTATECHANGED event.

## ALLOWSEQKEYRESET property

### Description

Specifies if the user can reset the sequential key counter for a data-bound form by entering a "=" character in the control bound to the key column.

### Property Value

This is a boolean value. It returns TRUE\$ if the user is allowed to reset the sequential key counter, or FALSE\$ otherwise.

### Property Traits

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

### Remarks

This property only applies to forms bound to a single table with a numeric single part key. When a user enters a "=" character in the control bound to the key column they are presented with the option of setting the sequential key counter for the table to a new value.

By default the sequential key is stored in the "%SK%" record in the dictionary table, but a different record may be chosen via the DEFVALUE property.

If the sequential key has not been initialized before it defaults to a value of "1".

### Example

```
// Example - Determine if the current form is allowing a sequential key reset  
AllowSeqKeyReset = Get_Property( @Window, "ALLOWSEQKEYRESET" )
```

### See Also

Common GUI DEFVALUE property, WINDOW IOOPTIONS property.

## ARRANGEICONS property

### Description

Specifies if an MDI Frame form automatically arranges its minimized MDI child forms. It does not affect MDI child forms that are not minimized.

### Property Value

This is a boolean value. Set to TRUE\$ to if an MDI Frame automatically arranges its minimized MDI child forms, or FALSE\$ otherwise.

### Property Traits

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

### Remarks

In Windows 3.1 minimized MDI Child forms were represented as icons in the MDI client area. From Windows 95 onwards they appear as small caption bars instead, but the term "icons" is still used to refer to them in this context.

Minimized MDI child forms are auto-arranged under the following circumstances:

- When the MDI Frame is resized.
- When an MDI child is activated or resized.
- When this property is first set to TRUE\$.

This property uses the WM\_MDIICONARRANGE window message internally. Please see the Microsoft website for more information on this message and on the topic of Windows MDI programming.

### Example

```
// Example - Set the current form to auto arrange its MDI child icons.  
PrevVal = Set_Property( @Window, "ARRANGEICONS", TRUE$ )
```

### See Also

WINDOW MDIICONARRANGE method, WINDOW ARRANGEICONS event, Appendix E – MDI Programming.

## ATRECORD property

### Description

Gets or sets data associated with the primary table of a data-bound form and updates data-bound controls.

When getting the property, the data is extracted directly from the controls on the form and merged with a cached version of the data row that was read from disk during the READ event (i.e. the RECORD property).

When setting the ATRECORD property, the passed data row is set as per the form's RECORD property and then the data-bound controls are automatically populated from this. The SAVEWARN property is also set to TRUE\$.

### Property Value

This property is a dynamic array that represents a database row for the primary table bound to the form. Its structure is determined by dictionary of the table.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get/Set	No	No	Yes

### Remarks

By default the WRITE event of a form only updates the columns in the database table that are bound directly to controls on the form – *any data columns that are not bound to a control are ignored by the write processor*. This means that you cannot update these non-control columns using ATRECORD *unless* the form's WRITEMODE property is set to "WriteEntireRow" - in this case the entire data row held in the RECORD property is updated with the contents of the bound controls and is then written to disk.

The intent behind the default of only updating bound controls is to prevent data corruption in cases where different forms load different columns from the same row at runtime. If each form updated the non-bound columns during the write process it would be possible to overwrite new data with stale data. The WRITEMODE property allows this safety feature to be circumvented, but it expects to be used with forms that do not use different columns with the same data row.

(Note: This property was designed to emulate the functionality of setting the @Record variable for a data-bound form in an Advanced Revelation application which would automatically populate the data-bound prompts on screen.)

This property is considered deprecated in favor of the ROW property. New applications should use ROW to work with the cached version of the data row for more consistent and expected results when using a "single form – single row" model.

### Example

```
// Example - a table has five columns:
//
//  CUST_ID      (key)
//  TITLE        <1>
//  FORENAME     <2>
//  SURNAME      <3>
//  DATE_OF_BIRTH <4>
//
// It is bound to a form that has the following three controls:
//
//  EDL_CUST_ID  -> CUST_ID (key)
//  EDL_FORENAME -> FORENAME
//  EDL_SURNAME  -> SURNAME
//
// The form is loaded with record "C1234" which has the following data:
//
// <1> MR
// <2> REN
// <3> HOEK
// <4> 65478
//
// Enter "STIMPSON J" in EDL_FORENAME

AtRecord = Get_Property( @Window, "ATRECORD" )

// AtRecord contains:
// <1> MR
// <2> STIMPSON J
// <3> HOEK
// <4> 65478

AtRecord<1> = "SIR"
AtRecord<3> = "CAT"

Call Set_Property_Only( @Window, "ATRECORD", AtRecord )
Call Exec_Method( @Window, "WRITEROW" )

// If we were to look at the record stored in the table we would now see this:
//
// <1> MR                <-- Not "SIR"!
// <2> STIMPSON J
// <3> CAT
// <4> 65478
//
// Because the TITLE column (field <1>) is not bound to a control on the form
// the value set by ATRECORD in field <1> ("SIR") does NOT get written unless
// the WRITEATRECORD property is set to TRUE$.
```

### **See Also**

WINDOW RECORD property, WINDOW ROW property, WINDOW SAVEWARN property, WINDOW WRITEATRECORD property, WINDOW WRITEENTIREROW property, WINDOW READROW method, WINDOW WRITEROW method, WINDOW READ event, WINDOW WRITE event.

## AUTOCOMPOSITED property

### Description

Specifies if the form uses system double-buffering during sizing operations to achieve smoother rendering.

### Property Value

This is a boolean value. When set to TRUE\$ the WS\_EX\_COMPOSITED extended window style is applied to the form during a resize operation and then removed afterwards,

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	No	No

### Remarks

Results of using this property may vary depending on whether Windows is using the Desktop Window Manager (DWM) for rendering. If the DWM is active (i.e. on Windows Vista/7 running full Aero, or Windows 8 onwards) then using this property may actually degrade the rendering operation.

### Example

```
// Example - set AUTOCOMPOSITED for the current form.  
Call Set_Property_Only( @Window, "AUTOCOMPOSITED", TRUE$ )
```

### See Also

Common GUI COMPOSITED property.

## COMMUTERMODULE property

### Description

Returns the name of a stored procedure that contains event handling code for the form.

### Property Value

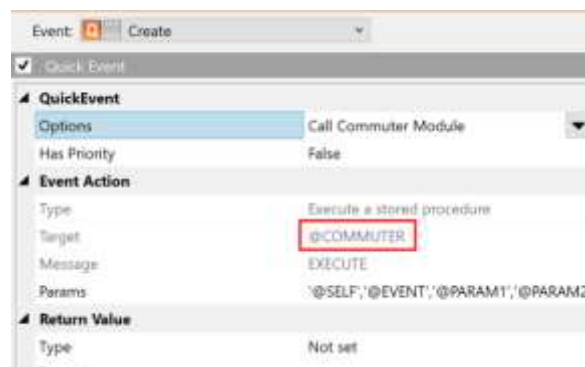
This is a string value containing the name of a valid stored procedure (Note this is *not* a fully qualified STPROCEDURE repository ID).

### Property Traits

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

### Remarks

The contents of this property are used by the QuickEvent processor to call a stored procedure to pass the event parameters onto at runtime. The QuickEvent processor replaces the token "@COMMUTER" in the event handler definition with the stored procedure name and then invokes it:



### Example

```
// Example - get the name of the current form's commuter module  
CommuterModule = Get_Property( @Window, "COMMUTERMODULE" )
```

### See Also

Appendix XXX – Event processing.



## CTRLMAP property

### Description

Returns a list of controls hosted by the specified form.

### Property Value

This an @Fm-delimited array of fully-qualified control names.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get	No	No	Yes

### Remarks

The list returned by this control is the list that is compiled into the form template by the Form Designer, and subsequently used to create the form at runtime.

If controls are later added to the form at runtime by using the SYSTEM CREATE method this list will not be updated. It can be updated "manually" by a developer by manipulating the form's "Window Common Area" if desired.

If a control is destroyed using the SYSTEM DESTROY method there is an optional flag that will remove the control from this list.

The list is held in the form's Window Common Area in the ControlMap@ variable which can be accessed by using the OIWIN\_COMM\_INIT insert record.

### Example

```
// Example - get the list of controls for the current form  
CtrlMap = Get_Property( CtrlEntID, "CTRLMAP" )
```

### See Also

SYSTEM CREATE method, SYSTEM DESTROY method, SYSTEM OBJECTLIST method, OIWIN\_COMM\_INIT insert record.

## DESTROYFLAG property

### Description

Specifies if the form is flagged for destruction during QUERYEND processing.

### Property Value

This is a boolean value. When TRUE\$ this means that the form has been marked for destruction during a QUERYEND process.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get	No	No	No

### Remarks

When the SYSTEM QUERYEND property returns TRUE\$, forms that are destroyed will have DESTROYFLAG set, but the forms themselves will not be destroyed.

Often, the HANDLE property is used to test if a form still exists, but if a form is closed (by using End\_Window for example) when QUERYEND is true, the HANDLE will still exist but DESTROYFLAG will be set.

In earlier versions of OpenInsight this property was called DESTROY\_FLAG. This name is deprecated but can still be used.

### Example

```
// Test to see if the current form is flagged for destruction  
IsFlagged = Get_Property( @Window, "DESTROYFLAG" )
```

### See Also

Common GUI HANDLE property, SYSTEM QUERYEND property.

## DPI property

### Description

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

### Property Value

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

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

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get	No	No	No

### Remarks

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. Beginning with Windows 8.1 individual monitors can have their own DPI settings - 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

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

## DWMANIMATION property

### Description

Specifies if the form uses Desktop Window Manager (DWM) animations when being displayed.

### Property Value

This is a boolean property. If TRUE\$ then DWM animations are enabled, or FALSE\$ if they are disabled.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get/Set	No	No	No

### Remarks

This property is only effective when Windows is using the DWM for rendering, i.e. on Windows Vista/7 running full Aero, or Windows 8 onwards.

The property implements the DWMWA\_TRANSITIONS\_FORCEDISABLED attribute of the DwmGetWindowAttribute and DwmSetWindowAttribute Windows API functions. Please see the Microsoft website for further information on these functions.

### Example

```
// Example -turn off DWM animations for the current window.  
Call Set_Property_Only( @Window, "DWMANIMATION", FALSE$ )
```

### See Also

N/a.

## FIRSTFOCUS property

### Description

Returns the name of the first control that receives the input focus when the specified form is created.

### Property Value

This is a string value containing the fully qualified name of a control, or null if no controls on the form could accept the input focus.

### Property Traits

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

### Remarks

When a form is created the Presentation Server scans through its controls in tab-order looking for the first control that can accept the input focus. This property returns the name of that control.

### Example

```
// Example - get the first control for the current form.  
FocusCtrl = Get_Property( @Window, "FIRSTFOCUS" )
```

### See Also

Common GUI FOCUS property, SYSTEM FOCUS property, WINDOW GOTFOCUSCONTROL property, WINDOW INITIALFOCUS property.

## FORMBORDERSTYLE property

### Description

Gets or sets the border style of the form.

### Property Value

This is an integer value that specifies the appearance and behavior of a form's border. It can be one of the following:

Value	Name	Description
0	None	The form has no border and cannot have a caption bar or be resized by the user.
1	Fixed	The form has a thin border and cannot be resized by the user.
2	Sizeable	The form has a normal border and can be resized by the user.
3	Dialog	The form has a normal border but cannot be resized by the user.
4	FixedTool	The form has a normal border with a thinner caption bar but cannot be resized by the user.
5	SizeableTool	The form has a normal border with a thinner caption bar and can be resized by the user.

Note that the "Tool" styles also have the following restrictions:

- Minimize, maximize and help buttons are not allowed.
- Tool-style forms do not appear on the Windows taskbar.
- Tool-style forms will not display an icon.

### Property Traits

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

### Remarks

The descriptions of the various border styles given above can vary depending on the current Windows visual style. For example, on Windows 10 all borders are a single pixel wide and the Tool form styles have the same caption bar height as the other styles. The differences in earlier versions of Windows were more pronounced.

Equated constants for this property can be found in the PS\_WINDOW\_EQUATES insert record.

## Example

```
// Example - set current form's border style to "none".  
$Insert PS_Window_Equates  
  
PrevStyle = Set_Property( @Window, "FORMBORDERSTYLE", PS_FORMBORDERSTYLE_NONE$ )
```

## See Also

WINDOW HELPBUTTON property, WINDOW ICON property, WINDOW MAXIMIZEBUTTON property, WINDOW MINIMIZEBUTTON property, WINDOW SHOWCAPTION property, WINDOW SIZINGMODE property, WINDOW SYSTEMMENU property, WINDOW TASKBARID property.

## GOTFOCUSCONTROL property

### Description

Returns the last control on the form with a GOTFOCUS event handler that had the input focus.

### Property Value

This is a string value containing the fully qualified name of a control, or null if no controls on the form with a GOTFOCUS event handler have had the input focus yet.

### Property Traits

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

### Remarks

N/a.

### Example

```
// Example - get the last control on the current form with a GOTFOCUS handler that  
// had the input focus
```

```
GotFocusCtrl = Get_Property( @Window, "GOTFOCUSCONTROL" )
```

### See Also

Common GUI FOCUS property, SYSTEM FOCUS property, WINDOW FIRSTFOCUS, WINDOW INITIALFOCUS property.



## HELPBUTTON property

### Description

Specifies if a help button is displayed on the form's caption bar.



### Property Value

This is a boolean value. It returns TRUE\$ if a button should be displayed, or FALSE\$ otherwise.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	No	No

### Remarks

When the help button is clicked the cursor changes to an arrow with a question mark. Clicking on an object with this cursor will trigger a HELP event for the object.

A help button can only be displayed if the follow criteria are met:

- The form has a caption bar.
- The form does not have a minimize or maximize button in its caption bar.
- The form does not have one of the "tool" styles set for its FORMBORDERSTYLE property.

This property implements the WS\_EX\_CONTEXTHELP extended window style. Please see the Microsoft website for more details.

### Example

```
// Example - Show a help button on the form's caption bar  
Call Set_Property_Only( @Window, "HELPBUTTON", TRUE$ )
```

### **See Also**

WINDOW FORMBORDERSTYLE property, WINDOW MAXIMIZEBUTTON property, WINDOW MINIMIZEBUTTON property, WINDOW SHOWCAPTION property, Common GUI HELP event.

## HIDEEFFECT property

### Description

Gets or sets the animation used with the HIDE method for the specified form.

### Property Value

This is an integer value that can be one of the following:

<b>Value</b>	<b>Name</b>	<b>Description</b>
<b>0</b>	None	No animation effect when hidden. This is the default.
<b>1</b>	Fade	The form fades until is no longer visible.
<b>2</b>	Slide down	The form's top edge slides down to its bottom edge until the form is no longer visible.
<b>3</b>	Slide up	The form's bottom edge slides up to its top edge until the form is no longer visible.
<b>4</b>	Slide right	The form's left edge slides to its right edge until it the form is no longer visible.
<b>5</b>	Slide left	The form's right edge moves to its left edge until the form is no longer visible.
<b>6</b>	Slide down and right	The form's top-left corner slides down to its bottom-right corner until the form is no longer visible.
<b>7</b>	Slide down and left	The form's top-right corner slides down to the bottom-left corner until the form is no longer visible.
<b>8</b>	Slide up and right	The form's bottom-left corner slides up to its top-right corner until the form is no longer visible.
<b>9</b>	Slide up and left	The form's bottom-right corner slides up to its top-left corner until the form is no longer visible.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	No	No

### Remarks

This property does not apply when used with an MDI Child form.

Equated constants for the HIDEEFFECT property value can be found in the PS\_WINDOW\_EQUATES insert record.

## Example

```
// Example - set the HIDEFFECT to "Slide Up" and hide the form  
$Insert PS_Window_Equates  
  
Call Set_Property_Only( @Window, "HIDEFFECT", PS_SHE_SLIDE_UP$ )  
Call Exec_Method( @Window, "HIDE" )
```

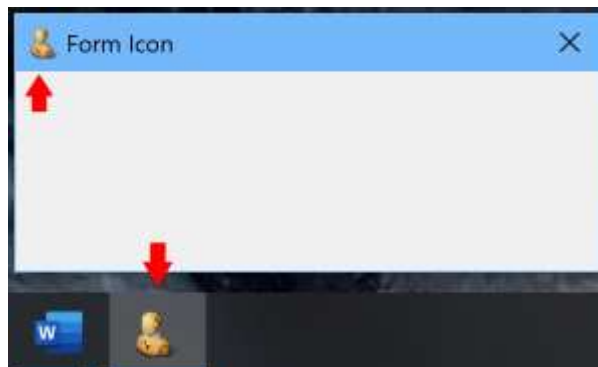
## See Also

WINDOW SHOWNEFFECT property, WINDOW TRANSLUCENCY property, WINDOW VISIBLE property, WINDOW HIDE method, WINDOW SHOW method.

## ICON property

### Description

Specifies the name of the icon displayed in the form's caption bar, and if appropriate, the Windows TaskBar. The icon is also used to access the form's System Menu.



### Property Value

This can be one of three formats:

- A path and file name of an icon (.ico) file.
- A path and file name to a resource file (such as a DLL) containing the icon, along with its resource ID. The latter component is separated from the file name by a "#" character.

E.g.

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

- The name of a Windows system icon:
  - "APPLICATION"
  - "ERROR"
  - "INFORMATION"
  - "QUESTION"
  - "WARNING"
  - "SHIELD"

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	No	No

## Remarks

A form must have a System Menu if it wants to display an icon on the caption bar.

Equated constants for using System Icons can be found the PS\_SYSICON\_EQUATES insert record.

## Example

```
Declare Function Repository
$insert PS_SysIcon_Equates

// Example 1 - setting an ICON property with a normal filename
IcoFile = ".\icons\rti_ide.res\rti_test_dummy.ico"
Call Set_Property_Only( @Window, "ICON", IcoFile )

// Example 2 - setting an ICON property with a resource from a DLL.
// (the icon with an ID of "1" from Oengine.dll)

IcoFile = "oengine.dll#1"
Call Set_Property_Only( @Window, "ICON", IcoFile )

// Example 3 - setting an ICON property with a System Icon
IcoFile = "WARNING"
Call Set_Property_Only( @Window, "ICON", IcoFile )

// Example 4 - setting an ICON property using a repository ID
//
// IMAGE entities always return the filename in field 1
IcoFile = Repository( "ACCESS", PS_REP_SYSICON_QUESTION$ )<1>
Call Set_Property_Only( @Window, "ICON", IcoFile )
```

## See Also

WINDOW SHOWCAPTION property, WINDOW SYSTEMMENU property, WINDOW TASKBARBUTTON property, WINDOW TASKBARID property, Appendix J – System Icons.

## ID property

### Description

Returns the current row key associated with the primary table data row in a data-bound form.

### Property Value

This property is string value containing the data row key if a row is loaded into the form *and* it is successfully locked by the form. If the form has not locked the row (i.e. it is in "View-Only" mode) then the ID property returns a null string instead.

### Property Traits

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

### Remarks

This property does not return the keys associated with secondary tables in a multi-table form.

### Example

```
// Example - assume the current form has loaded a row from the database with a key
// of "UK007" and the row has been Locked successfully by the form.

RowID = Get_Property( @Window, "ID" )

// RowID contains "UK007"

// Example - assume the current form has loaded a row from the database with a key
// of "UK007" and the row has NOT been Locked successfully by the form.

RowID = Get_Property( @Window, "ID" )

// RowID now contains ""
```

### See Also

WINDOW ATRECORD property, WINDOW ROW property, WINDOW RECORD property, WINDOW TABLE property, WINDOW READROW method, WINDOW READ event.

## INITIALFOCUS property

### Description

Specifies the control that receives the input focus when a form is activated.

### Property Value

This is a string value containing the fully qualified name of a control, or null if no controls on the form could accept the input focus.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get/Set	No	No	No

### Remarks

This property is set when a control receives the input focus and is constantly updated as the user moves between controls on a form. This is to ensure that a user who switches to another form is returned to the same control when the first form is activated once more.

(Contrast this with the FIRSTFOCUS property which is only set when the form is first created and remains constant.)

### Example

```
// Example - get the "initial-focus" control for the current form.  
FocusCtrl = Get_Property( @Window, "INITIALFOCUS" )
```

### See Also

Common GUI FOCUS property, SYSTEM FOCUS property, WINDOW GOTFOCUSCONTROL property, WINDOW FIRSTFOCUS property.



## INITIALPOSITION property

### Description

Specifies the starting location of the specified form.

### Property Value

This is an integer value that can be one of the following:

Value	Name	Description
0	As designed	The form is displayed using the form's Left and Top property values.
1	Center on desktop	The form is displayed centered on the desktop.
2	Center on parent	The form is displayed centered on its parent.

### Property Traits

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

### Remarks

This property does not apply when used with an MDI Child form.

Equated constants for the INITIALPOSITION property value can be found in the PS\_WINDOW\_EQUATES insert record.

### Example

```
// Get the current form's INITIALPOSITION setting  
InitPos = Get_Property( @Window, "INITIALPOSITION" )
```

### See Also

Common GUI SIZE property, Common GUI SYSTEMSIZE property, Common GUI RECT property, Common GUI LEFT property, Common GUI TOP property, WINDOW CENTER method.

## IOOPTIONS property

### Description

Specifies the data-binding options for the form.

### Property Value

This value is an @Fm-delimited array of data-binding options. See each individually named property for more information.

Field	PropertyName	Description
<1>	Reserved	N/a.
<2>	LockType	Specifies the type of row locking used by the form.
<3>	LockCoordination	Specifies if table-lock coordination is used by the form in conjunction with record locking.
<4>	AllowSelfLocks	Specifies if attempts to lock rows that are already locked by the current session are allowed.
<5>	Reserved	N/a.
<6>	NoClearOnWrite	Specifies if a form clears its contents after a successful write operation.
<7>	Reserved	N/a.
<8>	Reserved	N/a.
<9>	Reserved	N/a.
<10>	RequireOnWrite	Specifies if "data-required" checks are performed before a write operation.
<11>	QBFRReadMode	Specifies if a form's READ event is triggered when loading a data row in QBF mode.
<12>	NumericCompare	Specifies how comparisons for column updates are performed during a write operation.
<13>	WriteMode	Specifies how data set via the ROW property is saved during a write operation.
<14>	LoadPrevAlways	Specifies if the "previous data row" is updated on a read operation as well as a write operation.
<15>	SuppressSaveWarn	Specifies if a data-bound form ignores the SAVEWARN property during CLEAR and CLOSE processing.

<16>	AllowFormStateEvents	Specifies if a form triggers the FORMSTATECHANGED and MDICHILDSTATECHANGED events.
------	----------------------	--

### Property Traits

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

### Remarks

This property is available for backwards compatibility with earlier versions of OpenInsight and has been deprecated. From version 10 onwards each of the options above is exposed by its own individual property and these should be used in preference to the IOOPTIONS property.

Equates constants for the IOOPTIONS property can be found in the OIWIN\_EQUATES insert record.

### Example

```
// Example - set the "RequireOnWrite" option for the current form
$insert OIWin_Equates

Options = Get_Property( @Window, "IOOPTIONS" )

Options<FIO_REQONWRITEONLY$> = TRUE$

Call Set_Property_Only( @Window, "IOOPTIONS", Options )
```

### See Also

WINDOW ALLOWSELFLOCKS property, WINDOW ALLOWFORMSTATECHANGEDEVENTS, WINDOW LOADPREVALWAYS property, WINDOW LOCKCOORDINATION property, WINDOW LOCKLEVEL property, WINDOW LOCKMODE property, WINDOW NOCLEARONWRITE property, WINDOW NUMERICCOMPARE property, WINDOW QBFREADMODE property, WINDOW REQUIREONWRITE property, WINDOW SUPPRESSSAVEWARN property, WINDOW WRITEMODE property.

## LOADPREVALWAYS property

### Description

Specifies if the form's PREVROWVAL (previous data row) property is updated during a read operation as well as a write operation.

### Property Value

This is boolean value – if TRUE\$ then PREVROWVAL is updated during a read operation otherwise it is only updated during a write operation. The default is FALSE\$.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	No	Yes

### Remarks

When a row is written the system keeps a cached version of the data that was updated in a property called PREVROWVAL. This data is then used with subsequent calls to the READPREVROW method to populate data-bound controls with data from the previous row. By default, the PREVROWVAL is only updated from a write operation, but setting this property to TRUE\$ ensures that it is updated from a read operation too.

### Example

```
// Example - Set the current form's LOADPREVALWAYS property.  
Call Set_Property_Only( CtrlEntID, "LOADPREVALWAYS", TRUE$ )
```

### See Also

WINDOW IOOPTIONS property, WINDOW PREVROWVAL property, WINDOW READPREVROW method, WINDOW READ event, WINDOW WRITE event.

## LOCKCOORDINATION property

### Description

Specifies if table-lock coordination is used by a form in conjunction with record locking.

### Property Value

This is an integer value that can be one of the following:

Value	Name	Description
0	Normal	No table locks are used. This is the default.
1	WithTableLocks	A shared table lock will be applied in concert with the row lock. A shared table lock will conflict with any exclusive table locks which are already on the table or with any exclusive table locks which are attempted on the table after the shared table lock is implicitly applied. Table locks will always be attempted before row locks are applied.

### Property Traits

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

### Remarks

Equated constants for use with the LOCKCOORDINATION property may be found in the PS\_WINDOW\_EQUATES insert record.

### Example

```
// Example - set the LOCKCOORDINATION property to use table locks.
$insert PS_Window_Equates

PrevVal = Set_Property( @Window, "LOCKCOORDINATION", PS_LKCOORD_WITHTABLE$ )
```

### See Also

WINDOW IOOPTIONS property, WINDOW LOCKLEVEL property.

## LOCKTYPE property

### Description

Specifies the type of row locking used by a form.

### Property Value

This is an integer value that can be one of the following:

Value	Name	Description
0	Exclusive	<p>Exclusive locks disallow other users from gaining exclusive or shared locks on the same rows (This is the default option.)</p> <p>When a READ event is triggered exclusive locks will be applied to the primary table row and rows from subsidiary tables before any read operations are attempted.</p> <p>Failure of any lock will cause a locking error to be presented and the user will be allowed to continue in view-only mode or to cancel the read operation altogether.</p> <p>Locks will be removed when the form data is cleared through any mechanism, (for example, CLEAR, DELETE, CLOSE, or QBF operation).</p>
1	Shared	<p>This type is similar to the Exclusive type except that a shared lock is applied instead of an exclusive lock.</p> <p>A shared lock disallows other exclusive locks but allows other shared locks to be gained against the same rows.</p> <p>Write and delete operations should be disabled for this locking option.</p> <p>Shared locking is not supported on all networks – in this case this option behaves exactly as does the Exclusive option.</p>
2	No Locking	<p>No locking is performed on any rows, either from the primary or subsidiary tables.</p> <p>Write and delete operations should be disabled for this locking option.</p> <p>This setting may be appropriate for View only forms or for forms which perform custom locking beforehand.</p>

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get	No	No	Yes

### Remarks

Equated constants for use with the LOCKTYPE property may be found in the PS\_WINDOW\_EQUATES insert record.

### Example

```
// Example - set the LOCKTYPE property to use No Locking.  
$Insert PS_Window_Equates  
  
PrevVal = Set_Property( @Window, "LOCKTYPE", PS_LKCOORD_WITHTABLE$ )
```

### See Also

WINDOW IOOPTIONS property, WINDOW LOCKCOORDINATION property.

## MAXIMIZEBUTTON property

### Description

Specifies if a maximize button is displayed on the form's caption bar.



### Property Value

This is a boolean value. It returns TRUE\$ if a button should be displayed, or FALSE\$ otherwise.

### Property Traits

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

### Remarks

Clicking this button maximizes the form. When this form is maximized this button changes to a "restore" button – clicking this returns the form to its pre-maximized size.

A maximize button can only be displayed if the follow criteria are met:

- The form has a caption bar.
- The form does not have a help button in its caption bar.
- The form does not have one of the "tool" styles set for its FORMBORDERSTYLE property.

This property implements the WS\_MAXIMIZEBUTTON window style. Please see the Microsoft website for more details.

A minimize button is always displayed with a maximize button, even if the former is disabled (i.e. the form's MINIMIZEBUTTON property is set to FALSE\$).

### Example

```
// Example - Show a maximize button on the form's caption bar  
Call Set_Property_Only( @Window, "MAXIMIZEBUTTON", TRUE$ )
```



### **See Also**

WINDOW FORMBORDERSTYLE property, WINDOW HELPBUTTON property, WINDOW MINIMIZEBUTTON property, WINDOW SHOWCAPTION property, WINDOW VISIBLE property.

## MDIACTIVE property

### Description

Activates an MDI child form or returns the name of the currently active MDI child form for the specified MDI frame form.

### Property Value

This is a string value containing the fully qualified name of an active MDI child form. If an MDI frame form contains no MDI child forms this property returns a null string.

### Property Traits

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

### Remarks

This property applies to MDI frame forms only.

### Example

```
// Example - activate the CUSTOMERS MDI child form in the current MDI
// frame form
PrevActiveID = Set_Property( @Window, "MDIACTIVE", "CUSTOMERS" )
```

### See Also

WINDOW MDIFRAME property, WINDOW STARTMDICHILDFORM method, Start\_MDICChild stored procedure.

## MDIFRAME property

### Description

Returns the name of the parent MDI frame form for the specified MDI child form.

### Property Value

This is a string value containing the fully qualified name of an MDI frame form. If used with a non-MDI child form it returns a null string.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get	No	No	No

### Remarks

This property mainly applies to MDI child forms. If used on an MDI frame form it returns its own name, while if used on a non-MDI form it returns a null string.

### Example

```
// Example - get the MDI frame form for the current MDI child form  
MDIFrame = Get_Property( @Window, "MDIFRAME" )
```

### See Also

WINDOW MDIACTIVE property, WINDOW STARTMDICHILDFORM method, Start\_MDICChild stored procedure.

## MINIMIZEBUTTON property

### Description

Specifies if a minimize button is displayed on the form's caption bar.



### Property Value

This is a boolean value. It returns TRUE\$ if a button should be displayed, or FALSE\$ otherwise.

### Property Traits

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

### Remarks

Clicking this button minimize the form.

A minimize button can only be displayed if the follow criteria are met:

- The form has a caption bar.
- The form does not have a help button in its caption bar.
- The form does not have one of the "tool" styles set for its FORMBORDERSTYLE property.

This property implements the WS\_MINIMIZEBUTTON window style. Please see the Microsoft website for more details.

A maximize button is always displayed with a minimize button, even if the former is disabled (i.e. the form's MAXIMIZEBUTTON property is set to FALSE\$).

### Example

```
// Example - Show a minimize button on the form's caption bar  
Call Set_Property_Only( @Window, "MINIMIZEBUTTON", TRUE$ )
```

### **See Also**

WINDOW FORMBORDERSTYLE property, WINDOW HELPBUTTON property, WINDOW MAXIMIZEBUTTON property, WINDOW SHOWCAPTION property, WINDOW VISIBLE property.

## MULTIINSTANCE property

### Description

Specifies if multiple instances of the same form may be executed in the same session at runtime.

### Property Value

This is a boolean value. It returns TRUE\$ if the multiple instances are allowed, or FALSE\$ otherwise.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get	No	No	No

### Remarks

N/a.

### Example

```
// Example - Determine if the current form is multi-instance.  
IsMulti = Get_Property( @Window, "MULTIINSTANCE" )
```

### See Also

WINDOW STARTFORM method, WINDOW STARTMDICHILDFORM method.

## NEWROW property

### Description

Returns TRUE\$ if the specified data-bound form has loaded a new (blank) data row.

### Property Value

This is a boolean value. It returns TRUE\$ if the form has loaded a new row, or FALSE\$ otherwise.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get	No	No	Yes

### Remarks

N/a.

### Example

```
// Example - Determine if the current form is working with a new data row:  
IsNewRow = Get_Property( @Window, "NEWROW" )
```

### See Also

WINDOW ID property, WINDOW ROW property, WINDOW READROW method, WINDOW READ event.

## NOCLEARONWRITE property

### Description

Specifies if a data bound form clears its contents after a successful write operation.

### Property Value

This is a boolean value. It returns TRUE\$ if the form keeps its contents after a successful write operation, or FALSE\$ (the default) if it clears them,

### Property Traits

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

### Remarks

N/a.

### Example

```
// Example - Determine if the current form is set to clear after a write  
IsClearForm = Not( Get_Property( @Window, "NOCLEARONWRITE" ) )
```

### See Also

WINDOW IOOPTIONS property, WINDOW WRITEROW method, WINDOW CLEAR event, WINDOW WRITE event.



## NUMERICCOMPARE property

### Description

Specifies the type of comparison the form write operation uses to determine if a column needs updating.

### Property Value

This is a boolean value. It returns FALSE\$ (the default) if the form always uses a string comparison operation when scanning for changed columns, or TRUE\$ if it can use a numeric one where appropriate.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get	No	No	Yes

### Remarks

When a form executes a write operation it compares the data in each column bound to a control with the version currently on disk to see if an update is required. In early versions of OpenInsight this was performed as a simple equality test which could lead to some issues if the value "0" was compared to a null value (they would appear equal to the simple logical test, which is probably not the right answer). Later versions of OpenInsight switched to forcing a full string comparison instead so that the tests were more accurate. As a result of this the NUMERICCOMPARE property was introduced to allow for backwards compatibility.

### Example

```
// Example - Set the current form to use the simple numeric compare  
Call Set_Property_Only( @Window, "NUMERICCOMPARE", TRUE$ )
```

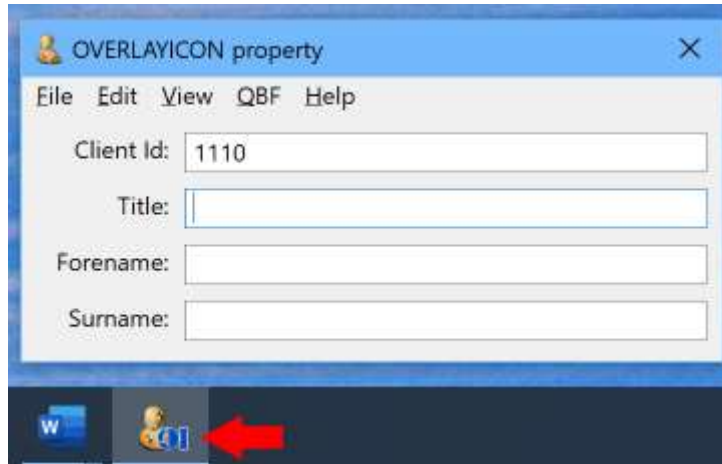
### See Also

WINDOW IOOPTIONS property, WINDOW WRITEROW method, WINDOW WRITE event.

## OVERLAYICON property

### Description

Specifies a small icon that may be used to overlay the normal icon on a form's taskbar button.



### Property Value

This is string value containing the name and path of an icon file.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get/Set	No	No	No

### Remarks

If you wish to use an icon stored in the OpenInsight repository then use the Repository stored procedure along with the ACCESS method to return the icon details. The file and path name for the icon file is specified in the first field.

Note that an overlay icon may only be set once Windows has created a taskbar button for the form, so it is necessary to check if the TASKBARBUTTON property returns TRUE\$ first.

## Example

```
// Example - set an overlay icon for the current form using the  
// file name and path from a repository entity  
Declare Function Repository  
  
IconRec = Repository( "ACCESS", @AppID : "*IMAGE*ICO*OI10" )  
IconFile = IconRec<1>  
  
Call Set_Property_Only( @Window, "OVERLAYICON", IconFile )
```

## See Also

WINDOW ICON property, WINDOW TASKBARBUTTON property, WINDOW TASKBARID property.

## OWNER property

### Description

Gets or sets the owner of the specified form.

### Property Value

This is string value containing the name of a running top-level form.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get/Set	No	No	No

### Remarks

When a form is created one of the parameters passed is the name of a parent form, which becomes the "owner" of the new form. When a form is owned by another form it exhibits the following characteristics:

- An owned form is always above its owner in the z-order.
- The system automatically destroys an owned form when its owner is destroyed.
- An owned form is hidden when its owner is minimized.
- An owned form does not have a button on the taskbar.

With `Get_Property` the `OWNER` and the `PARENT` property usually return the same value. However, it is not possible to change a form's owner with the `PARENT` property or the `SETPARENT` method because that will make it a child object which means it will no longer be a top-level form. Only the `OWNER` property can change a form's owner while maintaining its top-level state.

### Example

```
// Example - Set the owner of a the current form to MAIN_APP_FORM  
Call Set_Property_Only( @Window, "OWNER", "MAIN_APP_FORM" )
```

### See Also

Common GUI `CHILD` property, Common GUI `PARENT` property, Common GUI `SETPARENT` method, `WINDOW STARTFORM` method, `WINDOW SHOWDIALOG` method, `WINDOW SHOWMESSAGE` method, `WINDOW SHOWPOPUP` method.

## PLACEMENTDATA property

### Description

Gets or sets the show state and the restored, minimized, and maximized positions of the specified form.

### Property Value

This value is an @Fm-delimited array of placement information:

Field	PropertyName	Description
<1>	ShowCmd	Specifies the show state of the form – this is equivalent to its VISIBLE property.
<2>	NormalPosition	Specifies size of the form in its normal position:  <0,1> Left <0,2> Top <0,3> Width <0,4> Height
<3>	MinPosition	Specifies the coordinates of the form's upper-left corner when minimized. This is an @Vm-delimited array:  <0,1> Left <0,2> Top
<4>	MaxPosition	Specifies the coordinates of the form's upper-left corner when maximized. This is an @Vm-delimited array:  <0,1> Left <0,2> Top

### Property Traits

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

### Remarks

If the form is a top-level form that does not have a "Tool"-style FORMBORDERSTYLE property, then the coordinates in the structure above are in workspace coordinates, otherwise these are in screen coordinates. Therefore these coordinates should only be used with the PLACEMENTDATA property and not with the normal SIZE property.

(Workspace coordinates differ from screen coordinates in that they take the locations and sizes of application toolbars (including the taskbar) into account. Workspace coordinate (0,0) is the upper-left corner of the workspace area, the area of the screen not being used by application toolbars).

For more information on this property please refer to the Windows documentation regarding the `GetWindowPlacement` and `SetWindowPlacement` on the Microsoft website.

Equates constants for the `PLACEMENTDATA` property can be found in the `PS_WINDOW_EQUATES` insert record.

### Example

```
// Example - set the "RequireOnWrite" option for the current form  
$Insert OIWin_Equates  
  
Options = Get_Property( @Window, "IOOPTIONS" )  
  
Options<FIO_REQONWRITEONLY$> = TRUE$  
  
Call Set_Property_Only( @Window, "IOOPTIONS", Options )
```

### See Also

Common GUI MONITOR property, Common GUI RECT property, Common GUI SIZE property, SYSTEM MONITORLIST property, WINDOW VISIBLE property.

## PREVROWVAL property

### Description

Returns the values that were held in the specified form's controls for a previously loaded data row.

### Property Value

This property is a dynamic array that contains the data extracted from the form's controls from a previously loaded row (The format of the array is determined by the "row-map" which is a structure built by the form compiler at design-time).

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get	No	No	Yes

### Remarks

By default, when data is saved in a form, a copy of the data from the data-bound controls is cached and exposed via the PREVROWVAL property. This data may later be loaded back into the controls by using the form's READPREVROW method, thereby allowing easy duplication of previously entered values into the form's current data row.

If the form's LOADPREVALWAYS property is set to TRUE\$ then the PREVROWVAL is set when a row is read into the form as well as when it has been saved.

(Note: This property was designed to emulate the "Alt-C" functionality for a data-bound form as found in Advanced Revelation applications - this feature would automatically populate the data-bound prompts on screen from previously loaded data).

### Example

```
// Example - get the current form's previously saved data in "row-map" format.  
PrevRow = Get_Property( @Window, "PREVROWVAL" )
```

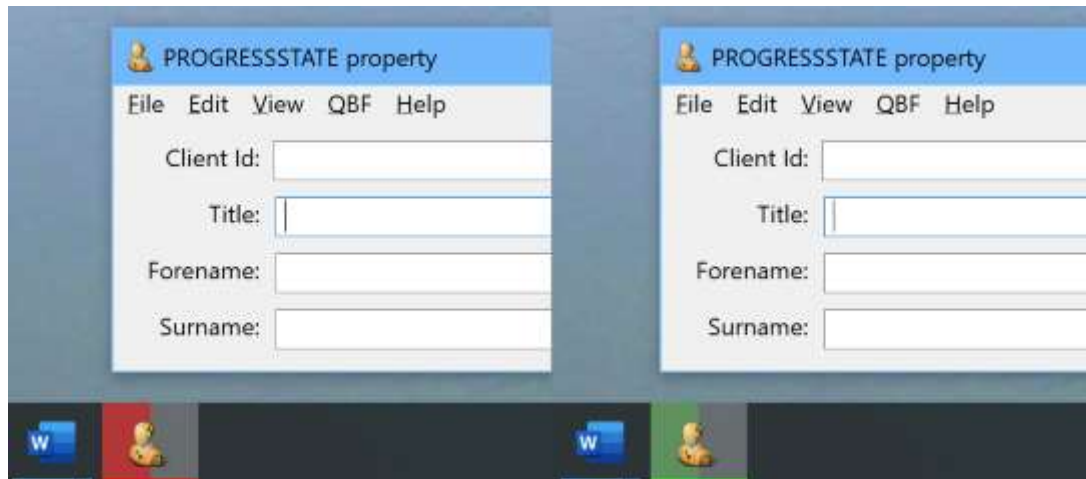
### See Also

WINDOW LOADPREVALWAYS property, WINDOW READPREVROW method, WINDOW READROW method, WINDOW WRITEROW method, WINDOW READ event, WINDOW WRITE event.

## PROGRESSTATE property

### Description

Sets the state of the current progress information on the specified form's taskbar button.



### Property Value

This is an integer value that can be one of the following:

Value	Name	Description
0	No Progress	Removes progress information from the taskbar button.
1	Normal	Sets the color of the taskbar button progress information to Green.
2	Error	Sets the color of the taskbar button progress information to Red.
3	Paused	Sets the color of the taskbar button progress information to Amber.
4	Indeterminate	Sets the taskbar button progress information to a green marquee effect.

### Property Traits

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

### Remarks

If the form does not have its own taskbar button and is grouped with other forms, the taskbar button for the group displays the progress information.



## Example

```
// Example - simple Loop to show progress information on the taskbar
$insert PS_Window_Equates

XCount = 100
For X = 1 To XCount

    // Update the Progress value
    Call Set_Property_Only( @Window, "PROGRESSVALUE", X : @Fm : XCount )

    // Call a function
    Call SomeFunction()

    // If there's an error then set the progress state to an error state
    If Get_Status( ErrorText ) Then
        Call Set_Property_Only( @Window, "PROGRESSSTATE", PS_PGS_ERROR$ )

        // Handle Error and assume fixed so set the state back to normal
        Call Set_Property_Only( @Window, "PROGRESSSTATE", PS_PGS_NORMAL$ )

    End

Next

// Remove the progress information
Call Set_Property_Only( @Window, "PROGRESSVALUE", 0 : @fm : 0 )
```

## See Also

PROGRESSBAR SYNCTASKBAR property, WINDOW PROGRESSVALUE property, WINDOW TASKBARBUTTON property, WINDOW TASKBARID property.

## PROGRESSVALUE property

### Description

Displays progress information on the specified form's taskbar button.



### Property Value

This is an @Fm delimited array containing progress information:

Field	Name	Description
<1>	CurrentValue	This is an integer specifying the current progress position. It is used with the MaximumValue field to calculate a percentage value which is then used to display the width of progress indicator on the taskbar button.  It cannot be greater than the MaximumValue.
<2>	MaximumValue	This is an integer specifying the maximum progress value. It is used with the CurrentValue field to calculate a percentage value which is then used to display the width of the progress indicator on the taskbar button.  It cannot be less than the CurrentValue.

### Property Traits

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

## Remarks

Setting the CurrentValue and MaximumValue fields to 0 removes the progress information from the taskbar button.

If the form does not have its own taskbar button and is grouped with other forms, the taskbar button for the group displays the progress information.

## Example

```
// Example - simple loop to show progress information on the taskbar
$insert PS_Window_Equates

XCount = 100
For X = 1 To XCount

    // Update the Progress value
    Call Set_Property_Only( @Window, "PROGRESSVALUE", X : @Fm : XCount )

    // Call a function
    Call SomeFunction()

    // If there's an error then set the progress state to an error state
    If Get_Status( ErrorText ) Then
        Call Set_Property_Only( @Window, "PROGRESSSTATE", PS_PGS_ERROR$ )

        // Handle Error and assume fixed so set the state back to normal
        Call Set_Property_Only( @Window, "PROGRESSSTATE", PS_PGS_NORMAL$ )

    End

Next

// Remove the progress information
Call Set_Property_Only( @Window, "PROGRESSVALUE", 0 : @fm : 0 )
```

## See Also

PROGRESSBAR SYNCTASKBAR property, WINDOW PROGRESSSTATE property, WINDOW TASKBARBUTTON property, WINDOW TASKBARID property.

## QBFLIST property

### Description

Gets or sets the QBF result list, i.e. the array of data keys used by the current QBF (Query-By-Form) session for the specified form

### Property Value

This is an @Fm-delimited list of keys to use with the current QBF session.

### Property Traits

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

### Remarks

When setting the QBFLIST property the first key in the list is automatically loaded into the form via the SHOWQBFFIRST method. If the QBF session has not been initialized via the QBFINITSESSION method, this is called internally first.

Setting an empty list will close the QBF session via the QBFCLOSESESSION method.

### Example

```
// Example - select data from a table and load the list of keys as a QBFLIST
$insert RList_Equates

Call RList( "SELECT CUSTOMERS WITH STATE EQ 'CA'", TARGET_ACTIVELIST$, "", "", "" )

QBKeys = ""
Eof     = FALSE$
Loop
  ReadNext Key Else Eof = TRUE$
Until Eof
  QBKeys := Key : @fm
Repeat
QBKeys[-1,1] = ""

Call Set_Property_Only( @Window, "QBFLIST", QBList )
```

### See Also

WINDOW QBFSHOWFIRST method, WINDOW QBFCLOSESESSION method, WINDOW QBFSHOWTABLE method, WINDOW QBFFIRST event, WINDOW QBFCLOSE event.

## QBFPOS property

### Description

Gets or sets the index of the row to display when the specified form has a valid QBF result list (QBFLIST property) loaded.

### Property Value

This is an integer value that must contain a valid position index for the list of keys in the QBFLIST property.

### Property Traits

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

### Remarks

Setting this property triggers the form's QBFABS event.

### Example

```
// Example - Assume we have been given a key that is in the QBFLIST
//           and we wish to load it into the form.

QBFList = Get_Property( @Window, "QBFLIST" )

Locate CustKey In QBFList Using @Fm Setting Pos Then
    Call Set_Property_Only( @Window, "QBFPOS", Pos )
End
```

### See Also

WINDOW QBFSHOWFIRST method, QBFSHOWLAST method, WINDOW QBFSHOWNEXT method, WINDOW QBFSHOWPREV method, WINDOW QBFSHOWTABLE method, WINDOW QBFABS event.

## QBFSTATUS property

### Description

Returns the status of the QBF session for the specified form.

### Property Value

This is an integer value that can be one of the following:

Value	Name	Description
0	QBFInactive	No QBF session is active for the form.
1	QBFInitialize	The form is ready to accept query data into its controls.
2	QBFActive	A QBF query has been executed and the QBF result list (QBFLIST property) has been populated.

### Property Traits

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

### Remarks

Equated constants for the QBF status values can be found in the RTI\_QBF\_EQUATES insert record.

### Example

```
// Example - Get the QBFSTATUS so we have set the UI controls accordingly
$insert RTI_QBF_Equates

QBFStatus = Get_Property( @Window, "QBFSTATUS" )

Begin Case
  Case ( QBFStatus = QBFSTAT_OFF$ )
    // Disable all QBF buttons except the QBF init session

  Case ( QBFStatus = QBFSTAT_INIT$ )
    // Use is entering a query - enable the QBF close session and
    // run query buttons, but all other QBF buttons are disabled.

  Case ( QBFStatus == QBFSTAT_ACTIVE$ )
    // A QBFLIST has been loaded - enable all QBF buttons except
    // the QBF init session and run query

End Case
```

### **See Also**

WINDOW QBFCLOSESESSION method, WINDOW QBFINITSESSION method, WINDOW QBFRUNQUERY method, WINDOW QBFCLOSE event, WINDOW QBFINIT event, WINDOW QBFRUN event.

## QBFREADMODE property

### Description

Specifies if and how a form triggers the READ event when loading data during QBF processing.

### Property Value

This is an integer value that can be one of the following:

Value	Name	Description
0	OnlyQBF	The custom QBF form loading process is used. A READ event is not triggered. This is the default for backwards compatibility.
1	QBFThenRead	The standard QBF form-load process is used, followed by a READ event. This causes the data to be read and loaded twice.  <i>This option is available for backwards compatibility purposes only and should be considered deprecated.</i>
2	OnlyRead	The standard READ event process is used to load rows during QBF processing.  <i>This is the preferred option.</i>

### Property Traits

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

### Remarks

By default (and to preserve backwards compatibility) the QBF events that load data into controls (QBFFIRST, QBFNEXT, QBFPREV, QBFLAST, QBFABS) do not use the normal READ event handler to accomplish this because they have their own internal methods (see the *OnlyQBF* option above). This means that any custom READ event processing required by the form when loading data will not be executed.

In previous versions of OpenInsight it was possible to set a flag in the form's IOOPTIONS property to trigger a READ event after the QBF load (this is exposed as the *QBFThenRead* option above), but this not an optimal solution because the data in the form will be loaded *twice*: once in the QBF event, and once in the READ event which, of course, is not very efficient.

This version of OpenInsight introduces the *OnlyRead* option instead, which means that the QBF processor uses the standard READ process to load data records,



thereby ensuring that any custom pre/post READ event processing will be executed. This is the preferred option and should be adopted where possible.

### Example

```
// Example - Determine if the current form is set to trigger a READ event after QBF  
// Loading.
```

```
QBReadMode = Get_Property( @Window, "QBFREADMODE" )
```

### See Also

WINDOW IOOPTIONS property, WINDOW QBFABS event, WINDOW QBFFIRST event, WINDOW QBFNEXT event, WINDOW QBFPREV event, WINDOW QBFLAST event, WINDOW READ event.

## RECORD property

### Description

Gets or sets the cached copy of the data row associated with the specified form.

### Property Value

This property is a dynamic array that represents a database row for the primary table bound to the form. Its structure is determined by dictionary of the table.

### Property Traits

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

### Remarks

When the data for the form's primary table is read at runtime a cached copy is stored in memory. This cached copy is updated from the controls as the user interacts with the form (usually during LOSTFOCUS and POSCHANGED events). The cached copy is then used to populate the @Record global variable before any CALCULATE events are executed on the controls in the form.

Setting this property (and therefore the cached copy) will not refresh the data held in the data-bound controls, nor will it allow an update of any non-control bound columns when the form's WRITE event is executed. The ROW property should be used for this purpose instead.

This property is considered deprecated in favor of the ROW property. New applications should use ROW to work with the cached version of the data row for more consistent and expected results when using a "single form – single row" model.

### Example

```
// RECORD example - update the cached version of the data row - note this will only  
// affect subsequent CALCULATE events - consider using the ROW property instead to  
// see changes in the controls and updates written to disk.
```

```
MyRecord<1> = "Mr"  
PrevVal = Set_Property( @Window, "RECORD". MyRecord )
```

### See Also

WINDOW ATRECORD property, WINDOW ROW property, WINDOW WRITEMODE property.

## REQUIREONWRITE property

### Description

Specifies when a form performs "required data" checks.

### Property Value

This is a boolean value. When set to TRUE\$ the form only performs "required data" checks just prior to a write operation and blocks the write if any controls have missing data. When set to FALSE\$ (the default) the checks are performed as the user moves between controls on the form (via the LOSTFOCUS and POSCHANGED events).

### Property Traits

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

### Remarks

N/a.

### Example

```
// Example - Determine if the current form is set to check for missing data
// just before a write operation

IsRequiredOnWrite = Get_Property( @Window, "REQUIREONWRITE" )
```

### See Also

Common GUI REQUIRED property, WINDOW IOOPTIONS property, WINDOW WRITEROW method, WINDOW WRITE event.

## RESTORESIZE property

### Description

Returns the position and size of the specified form in its non-maximized/minimized state.

### Property Value

This property value is an @Fm-delimited array of integer coordinates in DIPs (Device Independent Pixels):

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

### Property Traits

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

### Remarks

Each form keeps a copy of the "restore size", which is its position and size before it was maximized or minimized.

This value is updated when the form is resized or moved when not minimizing or maximizing.

This value is always returned as Device Independent Pixels (DIPs).

### Example

```
// Get the RESTORESIZE of the current form
RestoreSize = Get_Property( @Window, "RESTORESIZE" )
```

### See also

Common GUI RECT property, Common GUI SIZE property, WINDOW VISIBLE property, WINDOW SIZE event.

## ROW property

### Description

Gets or sets data associated with the primary table for the specified data-bound form and updates its data-bound controls.

When getting the property, the data is extracted directly from the controls on the form and merged with a cached version of the data row that was read from disk during the READ event.

When setting the ROW property, the cached copy of the form's data row is replaced and then the data-bound controls are automatically populated from that. The form's WRITEMODE property is automatically set to "WriteEntireRow" and the SAVEWARN property is also set to TRUE\$.

### Property Value

This property is a dynamic array that represents a database row for the primary table bound to the form. Its structure is determined by dictionary of the table.

### Property Traits

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

### Remarks

By default the WRITE event of a form only updates the columns in the database table that are bound directly to controls on the form – *any data columns that are not bound to a control are ignored by the write processor*. The intent behind this default behavior is to prevent data corruption in cases where different forms load different columns from the same row at runtime. If each form updated the non-bound columns during the write process it would be possible to overwrite new data with stale data.

However, this "multiple forms per single row" model is not as common as a "single form – single row" model, and there is an expectation that when setting the contents of the entire row at runtime the following is true:

- Any data-bound controls are updated from the contents of the new row.
- Any columns not bound to controls will still be written back to disk.

The ROW property fulfills both expectations in a single operation, whereas previous versions of OpenInsight would need to use the ATRECORD property and the WRITEATRECORD properties together.

(This emulates the functionality of setting the @Record variable for a data-bound form in an Advanced Revelation application which would automatically populate the data-bound prompts on screen.)

## Example

```
// Example - a table has five columns:
//
//  CUST_ID      (key)
//  TITLE        <1>
//  FORENAME     <2>
//  SURNAME      <3>
//  DATE_OF_BIRTH <4>
//
// It is bound to a form that has the following three controls:
//
//  EDL_CUST_ID  -> CUST_ID (key)
//  EDL_FORENAME -> FORENAME
//  EDL_SURNAME  -> SURNAME
//
// The form is loaded with record "C1234" which has the following data:
//
// <1> MR
// <2> REN
// <3> HOEK
// <4> 65478
//
// Enter "STIMPSON J" in EDL_FORENAME

Row = Get_Property( @Window, "ROW" )

// Row contains:
// <1> MR
// <2> STIMPSON J
// <3> HOEK
// <4> 65478

Row<1> = "SIR"
Row<3> = "CAT"

Call Set_Property_Only( @Window, "ROW", Row )
Call Exec_Method( @Window, "WRITEROW" )

// If we were to look at the record stored in the table we would now see this:
//
// <1> SIR          <-- Not bound to a control but still updated
// <2> STIMPSON J
// <3> CAT
// <4> 65478
//
// Because the ROW property automatically sets the WRITEMODE property to
// "WriteEntireRow" the TITLE column (field <1>) is written back even though
// it is not bound to a control on the form.
```

### **See Also**

WINDOW ATRECORD property, WINDOW RECORD property, WINDOW SAVEWARN property, WINDOW WRITEMODE property, WINDOW READROW method, WINDOW WRITEROW method, WINDOW READ event, WINDOW WRITE event.

## ROWLOCKED property

### Description

Returns TRUE\$ if the specified data-bound form has loaded a data row and has locked it for update.

### Property Value

This is a boolean value. It returns TRUE\$ if the form has locked a data row for update, or FALSE\$ otherwise.

### Property Traits

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

### Remarks

N/a.

### Example

```
// Example - Determine if the current form has a Locked row  
IsLocked = Get_Property( @Window, "ROWLOCKED" )
```

### See Also

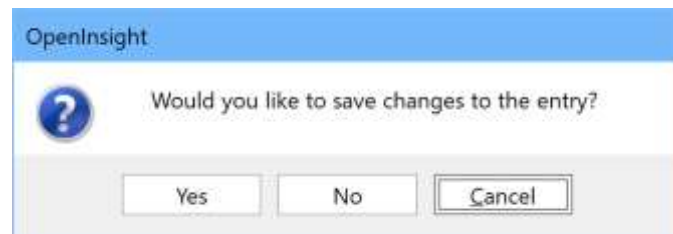
WINDOW ID property, WINDOW LOCKCOORDINATION property, WINDOW LOCKTYPE property, WINDOW ROW property, WINDOW READROW method, WINDOW READ event.



## SAVEWARN property

### Description

Specifies if a data-bound form contains changed data that has not been saved. This property is checked by the form's default CLEAR and CLOSE event handlers to decide if a warning message should be displayed to the user that changes will be lost unless the data is saved first.



### Property Value

This is a boolean value. It is set to TRUE\$ when data contained in the form has been changed from when it was originally loaded. It returns FALSE\$ if no changes have been made.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get/Set	No	No	Yes

### Remarks

By default, the SAVEWARN property is updated during the following events:

- It is set to FALSE\$ after data has been read and loaded into the form's controls.
- It is set to TRUE\$ during a data-bound control's LOSTFOCUS event when the user has changed data in the control.
- It is set to TRUE\$ during a data-bound EditTable control's POSCHANGED event when the user has changed data in the control.
- It is set to TRUE\$ when the DEFPROP property is used to change data in a data-bound control.

To ensure that the SAVEWARN property is updated properly, both the form's CLEAR and CLOSE events trigger a LOSTFOCUS event on the current control before they decide if a warning message needs to be shown to the user.

Setting the SAVEWARN property triggers the form's SYSMMSG event with a "SAVEWARNINFO" code (21) to allow applications to track when this has been set.

The SetDebugger stored procedure may also be used as a debugging tool to track when SAVEWARN is set.

### Example

```
// Example - Determine if data in the current form has changed  
IsChanged = Get_Property( @Window, "SAVEWARN" )
```

### See Also

Common GUI DEFPROP property, WINDOW SUPPRESSSAVEWARN property, Common GUI LOSTFOCUS event, EDITABLE POSCHANGED event, WINDOW CLEAR event, WINDOW CLOSE event, WINDOW SYSMMSG event, SetDebugger stored procedure.

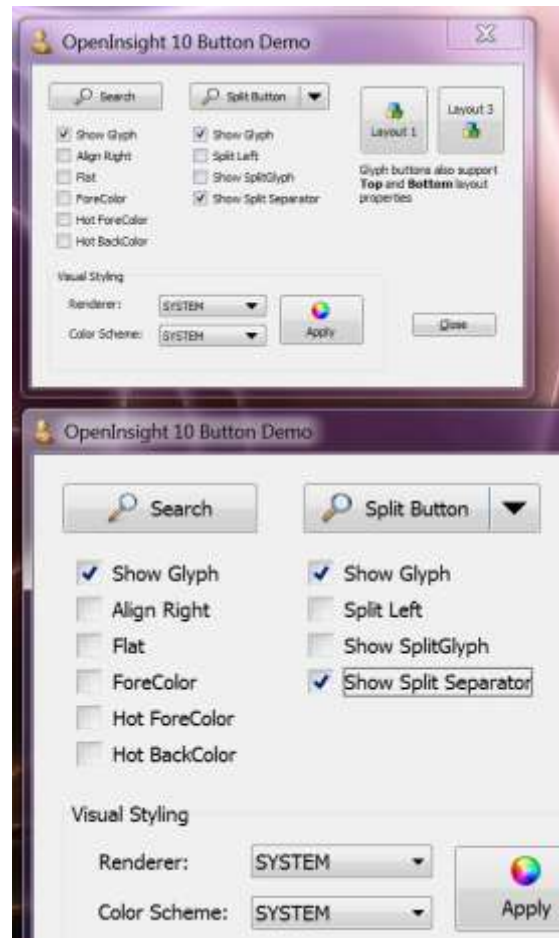
## SCALEFACTOR property

### Description

Specifies a custom scaling value for a form, allowing it to appear larger or smaller than normal.



Comparison of forms with a SCALEFACTOR of 0.5 and 1.0



Comparison of forms with a SCALEFACTOR of 1.0 and 1.7

### Property Value

This value is an @Fm-delimited array of scale factor attributes:

Field	Name	Description
<1>	ScaleFactor	<p>This is a number that specifies how much to scale the form by. A value of 1 means that the form has no custom scaling applied, a value of 1.5 scales the form to one-and-a-half times its normal size and so on.</p> <ul style="list-style-type: none"> <li>This value cannot be set at design time.</li> <li>This value can be set on its own at runtime without having to specify the other fields.</li> </ul>

<2>	MinScaleFactor	This specifies the minimum value that the ScaleFactor can be set to. By default it is set to "0.5", and can be a value between "0.1" and "1.0" inclusive.
<3>	MaxScaleFactor	This specifies the maximum value that the ScaleFactor can be set to. By default it is set to "5.0", and can be a value between "1.0" and "5.0" inclusive.
<4>	ScaleFactor Increment	If this field is set to a value other than 0 (the default) it allows the ScaleFactor to be adjusted via the Mouse-wheel /Ctrl-key combination, or with a "pinch-zoom" gesture if running with a touch screen. The increment value controls the rate at which the form grows or shrinks.

### Property Traits

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

### Remarks

The ScaleFactor is applied *after* any scaling is applied for the monitor DPI. For example, if the form runs on a 144 DPI monitor (150%) and has a ScaleFactor of 2 applied the *actual* scaling factor used is 3.0 (1.5 x 2.0).

Note that the minimum and maximum sizes that a form can be rescaled to is restricted by the minimum and maximum form sizes as defined by Windows. As a general rule this size is usually slightly larger than the size of the entire desktop across *all* monitors (See the GetSystemMetrics() Windows API function on the Microsoft website for more details, specifically with reference to the SM\_CXMAXTRACK, SM\_CXMINTRACK, SM\_CYMAXTRACK, and SM\_CYMINTRACK indexes). This restriction can, however, override this behaviour if the TRACKINGSIZE property is adjusted for the form, specifying values large enough to handle the desired scaling range.

Equates constants for the SCALEFACTOR property can be found in the PS\_WINDOW\_EQUATES insert record.

### Example

```
// Example - Set the scaling factor of the current form - the min/max/inc
// members can be ignored if they are unchanged.

Call Set_Property_Only( @Window, "SCALEFACTOR", 2 )

// Example - Turn on mouse-wheel/pinch-zoom scaling for the current form
$insert PS_Window_Equates

ScaleFactor = Get_Property( @Window, "SCALEFACTOR" )
ScaleFactor<PS_SCF_POS_INCREMENT$> = 0.1
Call Set_Property_Only( @Window, "SCALEFACTOR", ScaleFactor )
```

**See Also**

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

## SCALEUNITS property

### Description

Specifies how size and position coordinates are interpreted by a form. The scale units are a setting that determines how coordinates used in properties, methods and events are interpreted – either as DIPs (Device Independent Pixels) or as 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/Set	No	No	No

### Remarks

All child controls on a form use the same SCALEUNITS value.

Equated constants for use with the SCALEUNITS property can be found in the PS\_EQUATES insert record.

### Example

```
// Example - Set the current form's SCALEUNITS to pixels while it is
// positioned and reset them afterwards
$insert Ps_Equates

OrigScaleUnits = Set_Property( @Window, "SCALEUNITS", PS_SCU_PIXELS$ )

Call Set_Property_Only( @Window, "SIZE", NewSizeInPixels )

Call Set_Property_Only( @Window, "SCALEUNITS", OrigScaleUnits )
```

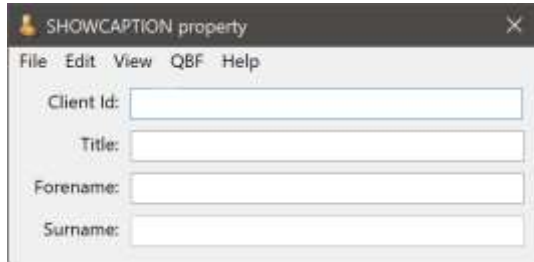
### See also

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

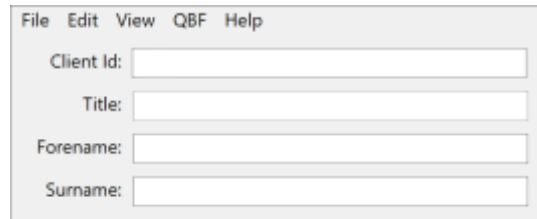
## SHOWCAPTION property

### Description

Specifies if a form displays a caption bar.



*Form with caption bar*



*Form without caption bar*

### Property Value

This is a boolean value. It returns TRUE\$ if the caption bar should be displayed, or FALSE\$ otherwise.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	No	No

### Remarks

This property implements the WS\_CAPTION window style. Please see the Microsoft website for more details.

### Example

```
// Example - Remove the current form's caption bar  
Call Set_Property_Only( @Window, "SHOWCAPTION", FALSE$ )
```

### See Also

WINDOW FORMBORDERSTYLE property, WINDOW HELPBUTTON property, WINDOW MAXIMIZEBUTTON property, WINDOW MINIMIZEBUTTON property, WINDOW VISIBLE property.

## SHOWEFFECT property

### Description

Gets or sets the animation used with the SHOW method for the specified form.

### Property Value

This is an integer value that can be one of the following:

Value	Name	Description
0	None	No animation effect when shown. This is the default.
1	Fade	The form fades in until it is fully visible.
2	Slide down	The form's bottom edge slides down from its top edge until the form is fully visible.
3	Slide up	The form's top edge slides up from its bottom edge until the form is fully visible.
4	Slide right	The form's right edge slides out from its left edge until it the form is fully visible.
5	Slide left	The form's left edge slides out from its right edge until the form is fully visible.
6	Slide down and right	The form's bottom-right corner slides down from its top-left corner until the form is fully visible.
7	Slide down and left	The form's bottom-left corner slides down from its top-right corner until the form is fully visible.
8	Slide up and right	The form's top-right corner slides up from its bottom-left corner until the form is fully visible.
9	Slide up and left	The form's top-left corner slides up from its bottom-right corner until the form is fully visible.

### Property Traits

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

### Remarks

This property does not apply when used with an MDI Child form.

Equated constants for the SHOWEFFECT property value can be found in the PS\_WINDOW\_EQUATES insert record.



## Example

```
// Example - set the SHOWEFFECT to "Slide Down" and show the current form  
$Insert PS_Window_Equates  
  
Call Set_Property_Only( @Window, "SHOWEFFECT", PS_SHE_SLIDE_DOWN$ )  
Call Exec_Method( @Window, "SHOW" )
```

## See Also

WINDOW HIDEFFECT property, WINDOW TRANSLUCENCY property, WINDOW VISIBLE property, WINDOW HIDE method, WINDOW SHOW method.

## SIZINGMODE property

### Description

Specifies if a form can be resized with the mouse. Usually the FORMBORDERSTYLE property determines if the mouse can be used to resize a form. The SIZINGMODE property may be used to override this behaviour.

### Property Value

This is an integer value that may be one of the following values:

Value	Name	Description
0	Default	Resizing the form is controlled by the FORMBORDERSTYLE property.
1	Always	The form may always be resized with the mouse.
2	Never	The form cannot be resized with the mouse.

### Property Traits

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

### Remarks

Equated constants for this property can be found in the PS\_WINDOW\_EQUATES insert record.

### Example

```
// Example - set current form's SIZINGMODE property to "Never".
$insert PS_Window_Equates

PrevVal = Set_Property( @Window, "SIZINGMODE", PS_SIZINGMODE_NEVER$ )
```

### See Also

WINDOW FORMBORDERSTYLE property.

## STATUSLINE property

### Description

Identifies the control that receives "status" messages from stored procedures when the specified form is active. Status messages are text strings sent to the Presentation Server via the Send\_Info stored procedure when code is executed in event context.

### Property Value

This is string value containing the name of a valid control instance. The control's TEXT property is updated with the status message.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	No	No

### Remarks

When processing a status message the Presentation Server first attempts to use the nominated STATUSLINE control on the active form. If this is not possible it then attempts to use the last valid STATUSLINE control from a previously active form. If no STATUSLINE control is available the message is sent to the results text box in the System Monitor instead.

### Example

```
// Example set the current form's TXT_STATUS STATIC control to receive status  
// messages from the Send_info stored procedure.
```

```
Call Set_Property_Only( @Window, "STATUSLINE", @Window : ".TXT_STATUS" )
```

### See Also

Common GUI TEXT property, SYSTEM RECEIVER property, System Monitor chapter, Send\_Info stored procedure.

## STYLESHEET property

### Description

Specifies the name of a form to use as a "styling template" when adding controls to a form at design-time.

When a new control is added to a form at design time the stylesheet form is scanned to see if it contains a control with the same type. If so, the following properties are duplicated for the new control:

- BACKCOLOR
- FORECOLOR
- FONT
- HEIGHT
- WIDTH

### Property Value

This is string value containing the name of a valid form.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	N/a	No	No	Yes

### Remarks

N/a.

### Example

N/a.

### See Also

N/a.

## SUPPRESSSAVEWARN property

### Description

Specifies if a data-bound form checks the SAVEWARN property to see if data has changed before clearing its contents or closing.

### Property Value

This is a boolean value. When set to TRUE\$ the form will not warn the user about unsaved changes in data-bound controls when it is cleared or closed. When set the FALSE\$ (the default) then the SAVEWARN property will be processed as normal.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	No	Yes

### Remarks

Setting this property to TRUE\$ allows a form to be closed unconditionally.

### Example

```
// Example - Determine if the current form is set to check for unsaved changes  
// just before its contents are cleared or it is closed.
```

```
NoSaveWarning = Get_Property( @Window, "SUPPRESSSAVEWARN" )
```

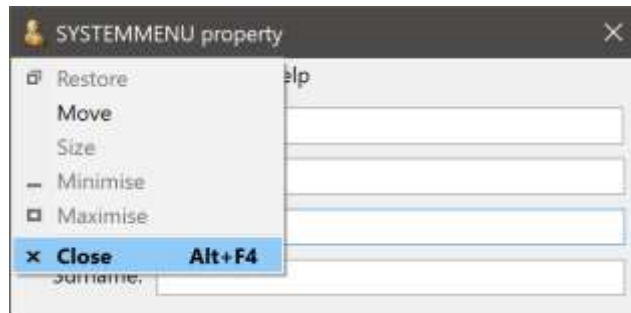
### See Also

WINDOW IOOPTIONS property, WINDOW SAVEWARN property, WINDOW CLEAR event, WINDOW CLOSE event, WINDOW SYMSG event.

## SYSTEMMENU property

### Description

Specifies if a "System Menu" is allowed for the specified form. A System Menu is the default menu normally added to a form to allow some basic windowing operations:



### Property Value

This is a boolean value. It returns TRUE\$ if the System Menu should be allowed, or FALSE\$ otherwise.

### Property Traits

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

### Remarks

A form must have a System Menu if it wants to display an icon on the caption bar.

At runtime the System Menu can be activated by clicking the form's icon on the left side of the caption bar. If the form does not have a caption bar or an icon the System Menu may still be activated by using the "Alt+Space" hot-key combination.

This property implements the WS\_SYSTEMMENU window style. Please see the Microsoft website for more details.

### Example

```
// Example - Ensure that the current form has a System Menu  
Call Set_Property_Only( @Window, "SYSTEMMENU", TRUE$ )
```

### **See Also**

WINDOW FORMBORDERSTYLE property, WINDOW ICON property, WINDOW SHOWCAPTION property.

## TABLE property

### Description

Returns the name of the primary table that specified the form is bound to.

### Property Value

This property value returns the name of the primary table if any controls on the form are data-bound, otherwise it returns null.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
N/a	Get	No	No	Yes

### Remarks

N/a.

### Example

```
// Example - The primary table that the form is bound to  
TableList = Get_Property( CtrlEntID, "TABLE" )
```

### See Also

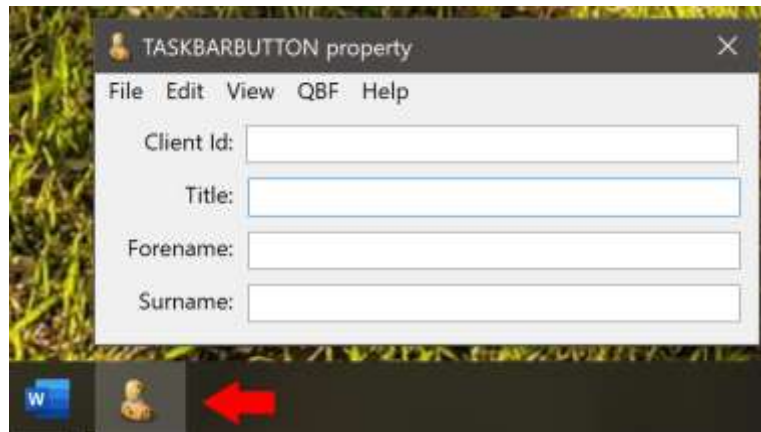
WINDOW ATRECORD property, WINDOW ID property, WINDOW ROW property, WINDOW RECORD property, WINDOW READROW method, WINDOW READ event.



## TASKBARBUTTON property

### Description

Returns a flag denoting if Windows has created a taskbar button for the specified form.



### Property Value

This is a boolean value. It returns TRUE\$ if Windows has created a taskbar button for the form, or FALSE\$ otherwise.

### Property Traits

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

### Remarks

There is usually a short delay between a form being created and Windows adding a button for it on the taskbar, and it sends a notification to the form once the button has been added. At this point it is then possible to set an OVERLAYICON if desired.

### Example

```
// Example - set an overlay icon, but check if the taskbar button has been created

Loop
  HasButton = Get_Property( @Window, "TASKBARBUTTON" )
Until HasButton
  Call Yield( TRUE$ )
Repeat

Call Set_Property_Only( @Window, "OVERLAYCON", IconFile )
```

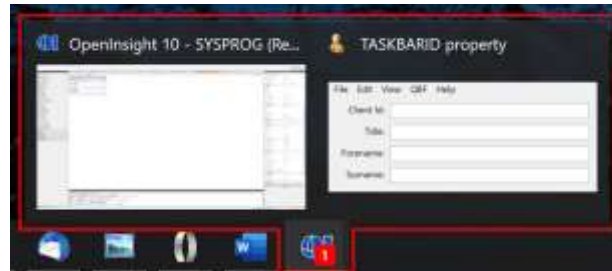
### **See Also**

PROGRESSBAR SYNCTASKBAR property, WINDOW ICON property, WINDOW OVERLAYICON property, WINDOW TASKBARID property.

## TASKBARID property

### Description

Specifies a text string used to group or ungroup forms on the Windows taskbar. By default, Windows groups all forms belonging to the same process together under same taskbar button like so (shown here grouped under the main OpenInsight process):



This behavior can be changed for a form by setting the TASKBARID property to a new text value - this forces Windows to create a new button on the taskbar instead. (All forms that share this value will be grouped together):



### Property Value

This is a string value – when it is not null any forms sharing the same value will be grouped under the same taskbar button. When it is null the forms will be grouped under a default taskbar button.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	No	No

## Remarks

This property uses the Windows SHGetPropertyStoreForWindow function to set a form-specific AppUserModelID internally. More for information on this and programming the Windows taskbar please see the relevant documentation on the Microsoft website.

## Example

```
// Example - set the TASKBARID for the current form so it appears  
// under its own button on the TaskBar. An easy way to do this is  
// to use its actual name, as this will always be unique within  
// a single instance of OpenInsight
```

```
Call Set_Property_Only( @Window, "TASKBARID", @Window )
```

## See Also

WINDOW ICON property, WINDOW OVERLAYICON property, WINDOW TASKBARBUTTON property.

## TRACKINGSIZE property

### Description

Specifies the minimum and maximum sizes that a user may resize a form to.

### Property Value

This property value is an @Fm-delimited array of integer values:

- <1> Minimum Tracking Width
- <2> Minimum Tracking Height
- <3> Maximum Tracking Width
- <4> Maximum Tracking Height

Each value must be greater than zero, or have a value of "-1", which means use the default Windows value for that attribute instead.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	Yes	No

### Remarks

Setting all four fields to the same non-negative value will prevent the user resizing the form with the mouse or with its System Menu (see example below).

- This is useful for preventing the resizing of MDI child forms because Windows always creates them with a resizable border style, overriding any selected design settings such as a thin, non-sizable border.

Equated constants for use with the TRACKINGSIZE property can be found in the PS\_EQUATES insert record.

The TRACKINGSIZE property is used during WM\_GETMINMAXINFO message processing to determine form resizing limits. Please refer to the documentation on the Microsoft website for further information regarding this process.

## Example

```
// Example - fix the size of the form so that the user cannot change it
$insert PS_Equates

FormSize = Get_Property( @Window, "SIZE" )

TrackSize = ""
TrackSize<PS_TRACKSIZE_MINWIDTH$> = FormSize<3>
TrackSize<PS_TRACKSIZE_MINHEIGHT$> = FormSize<4>
TrackSize<PS_TRACKSIZE_MAXWIDTH$> = FormSize<3>
TrackSize<PS_TRACKSIZE_MAXHEIGHT$> = FormSize<4>

Call Set_Property_Only( @Window, "TRACKINGSIZE", TrackSize )

// Example - set the TRACKINGSIZE back to it's default values

TrackSize = str( PS_TRACKSIZE_VAL_NOTSET$ : @fm, 4 )
TrackSize[-1,1] = ""

Call Set_Property_Only( @Window, "TRACKINGSIZE", TrackSize )
```

## See also

Common GUI SIZE property, WINDOW FORMBORDERSTYLE property, WINDOW SYSTEMMENU property, WINDOW SIZE event.

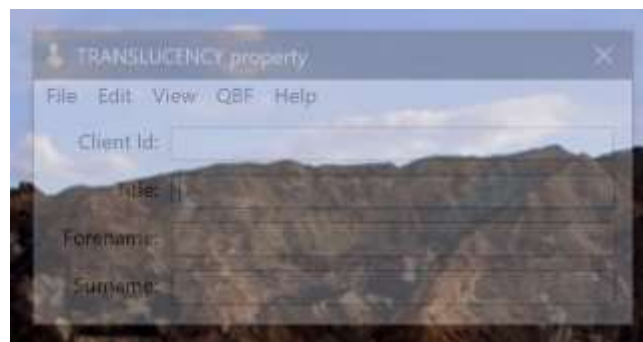
## TRANSLUCENCY property

### Description

Specifies the degree of transparency applied to a form when it is painted. Note that unlike the common GUI TRANSLUCENCY property, this effect applies to the entire form, not just its client area.



Form with 30% TRANSLUCENCY applied



Form with 70% TRANSLUCENCY applied

### Property Value

This property is an integer value between 1 and 100, which represents the percentage of transparency applied to the form. A value of 0 means fully opaque, while a value of 100 means fully transparent (i.e. the form will not be drawn).

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	No	No

### Remarks

Setting the TRANSLUCENCY to 100 will hide the form, but it is still considered to be visible, i.e. the VISIBLE property will not return an WS\_HIDE\$ value.

### Example

```
// Set the TRANSLUCENCY of the current form to 30%  
PrevVal = Set_Property( @Window, "TRANSLUCENCY", 30 )  
  
// Remove the TRANSLUCENCY from the current form  
PrevVal = Set_Property( @Window, "TRANSLUCENCY", 0 )  
  
// Hide the current form (Note - the form is still considered to be visible!)  
PrevVal = Set_Property( @Window, "TRANSLUCENCY", 100 )
```

### See also

Common GUI TRANSLUCENCY property, WINDOW HIDEFFECT property, WINDOW SHOWN property, WINDOW VISIBLE property, WINDOW HIDE method, WINDOW SHOW method property.



## TOPMOST property

### Description

Specifies if a form appears above all other non-TOPMOST forms in the system z-order, even when the form is not active.

### Property Value

This is a boolean value. It returns TRUE\$ if the form is flagged as a topmost form, or FALSE\$ otherwise.

### Property Traits

<i>Development</i>	<i>Runtime</i>	<i>Indexed</i>	<i>Scaled</i>	<i>Synthetic</i>
Get/Set	Get/Set	No	No	No

### Remarks

The TOPMOST property 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

```
// Example - Determine if the current form is marked as a topmost form.  
IsTopMost = Get_Property( @Window, "TOPMOST" )
```

### See Also

Common GUI SETZORDER method.

## VISIBLE property

### Description

Specifies if a form is visible, hidden, maximized, or minimized.

### Property Value

The VISIBLE property is an integer value that specifies how the form is displayed. It may be one of the following when used with Set\_Property:

Value	Name	Description
0	SW_HIDE	Hides the form and activates another form.
1	SW_SHOWNORMAL	Displays the form and activates it. If the form is minimized or maximized, the system restores it to its original size and position.
2	SW_SHOWMINIMIZED	Minimizes the form and activates it.
3	SW_SHOWMAXIMIZED	Maximizes the form and activates it.
4	SW_SHOWNOACTIVATE	Displays the form in its most recent size and position. This value is similar to SW_SHOWNORMAL, except that the form is not activated.
5	SW_SHOW	Activates the form and displays it in its current size and position.
6	SW_MINIMIZE	Minimizes the specified form and activates the next top-level form in the Z order.
7	SW_SHOWMINNOACTIVE	Displays the form as a minimized form. This value is similar to SW_SHOWMINIMIZED, except the form is not activated.
8	SW_SHOWNA	Displays the form in its current size and position. This value is similar to SW_SHOW, except that the form is not activated.
9	SW_RESTORE	Activates and displays the form. If the form is minimized or maximized, the system restores it to its original size and position. An application should specify this flag when restoring a minimized form.

When used with Get\_Property only the following values are returned:

- (0) SW\_HIDE (form is hidden).
- (1) SW\_SHOWNORMAL (form is visible).
- (2) SW\_SHOWMINIMIZED (form is visible and minimized).
- (3) SW\_SHOWMAXIMIZED (form is visible and maximized).

### 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 corresponds to the function's nCmdShow parameter value. Please see the Microsoft website for more details on controlling form visibility.

Constants for these values are defined in the MSWIN\_SHOWWINDOW\_EQUATES insert record.

### Example

```
$Insert MsWin_ShowWindow_Equates

// Example - Hide the current form

Call Set_Property_Only( @Window, "VISIBLE", SW_HIDE$ )

// Example - Maximize the current form

IsMaximized = Get_Property( @Window, "VISIBLE" )
If IsMaximized Else

    Call Set_Property_Only( @Window, "VISIBLE", SW_SHOWMAXIMIZED$ )

End
```

### See also

Common GUI VISIBLE property.

## WRITEATRECORD property

### Description

Specifies how data set with the ATRECORD property is saved during a write operation.

### Property Value

This is a boolean value. When set to TRUE\$, data from columns that are not bound to a control on the form (and that were set via the ATRECORD property) is written to the table in addition to the data in the controls. When set to FALSE\$ (the default) only data from the controls is written back during a write operation.

### Property Traits

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

### Remarks

See the ATRECORD property for a full explanation of how WRITEATRECORD works in conjunction with ATRECORD.

This property is considered deprecated in favor of the WRITEMODE property. It is effectively a backwards-compatible synonym for WRITEMODE.

### Example

```
// Example - set the WRITEATRECORD so that any data not bound to a control in the
// current form is still updated in the WRITE event.

Call Set_Property_Only ( @Window, "WRITEATRECORD", TRUE$ )
Call Set_Property_Only( @Window, "ATRECORD", MyDataRow )
```

### See Also

WINDOW ATRECORD property, WINDOW RECORD property, WINDOW ROW property, WINDOW WRITEMODE property.

## WRITEMODE property

### Description

Specifies how data set with the ROW property is saved during a write operation.

### Property Value

This is an integer value that can be one of the following:

Value	Name	Description
0	WriteControlsOnly	Only data from data-bound controls on the form is written back to the table during a write operation. This is the default behavior.
1	WriteEntireRow	Data from columns that are not bound to a control on the form (and that were set via the ROW property) is written to the table in addition to the data in the controls.

### Property Traits

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

### Remarks

See the ROW property for a full explanation of how WRITEMODE works in conjunction with ROW.

Equated constants for use with the WRITEMODE property may be found in the PS\_WINDOW\_EQUATES insert record.

### Example

```
// Example - set the WRITEMODE so that any data not bound to a control in the current.  
// form is still updated in the WRITE event.  
$Insert PS_Window_Equates  
  
Call Set_Property_Only ( @Window, "WRITEMODE", PS_WRMD_ALL$ )  
Call Set_Property_Only( @Window, "ROW", MyDataRow )
```

### See Also

WINDOW ROW property.

## WINDOW Methods

The WINDOW object supports the following methods in addition to the Common GUI Object methods, except where noted below:

Name	Description
<b>CENTER</b>	Centers a form on the desktop or over its parent.
<b>CLEARROW</b>	Clears the data row from a data-bound form.
<b>CLOSE</b>	Closes a form by triggering its CLOSE event.
<b>CLOSEDIALOG</b>	Closes a dialog box and returns a value to the owner.
<b>DELETEROW</b>	Deletes the currently loaded data row in a data-bound form.
<b>FLASH</b>	"Flashes" a form's caption and/or taskbar button a specified number of times.
<b>GETFOCUSEDCONTROL</b>	Returns the control with focus on the form.
<b>HIDE</b>	Hides a form using the specified effect.
<b>HIDEMENUBAR</b>	Hides the specified form's menu bar.
<b>MDICASCADE</b>	Arranges MDI child forms in a cascaded overlapping formation.
<b>MDIICONARRANGE</b>	Arranges minimized MDI child forms.
<b>MDITILE</b>	Arranges MDI child forms in a tiled formation.
<b>QBFASKQUERY</b>	Asks for and executes an RLIST query statement and populates the specified form's QBF result list.
<b>QBFCLOSESESSION</b>	Closes a QBF session for the specified form.
<b>QBFGOTO</b>	Loads a specific row from the form's QBF result list based on its position.
<b>QBFGOTOID</b>	Loads a specific row from the QBF result list using its ID.
<b>QBFINITSESSION</b>	Initializes and begins a QBF session for the specified form.
<b>QBFLOADSAVEDLIST</b>	Asks the user for the name of a saved list and loads the keys into the specified form's QBF result list.
<b>QBFRUNQUERY</b>	Builds and executes an RLIST query from the data in the controls and populates the specified form's QBF result list.
<b>QBFSHOWFIRST</b>	Shows the first row from the specified form's QBF result list.
<b>QBFSHOWLAST</b>	Shows the last row from the specified form's QBF result list.
<b>QBFSHOWNEXT</b>	Shows the next row from the specified form's QBF result list.
<b>QBFSHOWPREV</b>	Shows the previous row from the specified form's QBF result list.
<b>QBFSHOWTABLE</b>	Displays the specified form's QBF result list in a non-modal dialog box.
<b>READROW</b>	Reads the data into the specified data-bound form and populates the controls.
<b>READPREVROW</b>	Populates controls in the specified form with data from the previously loaded row.

<b>*SCROLL</b>	Scrolls the contents of the form by the specified amount.
<b>SHOW</b>	Displays a form using the specified effect.
<b>SHOWMENUBAR</b>	Displays the specified form's menu bar.
<b>SHOWDIALOG</b>	Displays a modal dialog using the specified form as the owner.
<b>SHOWINDEXLOOKUP</b>	Displays the index lookup dialog box using the specified form as the owner,
<b>SHOWMESSAGE</b>	Displays a message box, using the specified form as the parent.
<b>SHOWPOPUP</b>	Displays a popup box, using the specified form as the owner.
<b>STARTFORM</b>	Executes a new form, using the specified form as the owner.
<b>STARTMDICHILDFORM</b>	Executes a new MDI child form for the specified MDI frame parent.
<b>TRACKDROPDOWNMENU</b>	Displays a dropdown menu for a top-level menu bar item on the specified form.
<b>UPDATEROW</b>	Updates the data associated with the primary table of a data-bound form without updating data-bound controls.
<b>WRITEROW</b>	Writes the data contains in the specified data-bound form to the database.

## CENTER method

### Description

Centers a form on the desktop or over its parent object.

### Syntax

```
NewSize = Exec_Method( CtrlEntID, "CENTER", CenterParent, IdealSize, |  
                        CalculateOnly, ParentSize, Options )
```

### Parameters

Name	Required	Description
<b>CenterParent</b>	No	If TRUE\$ then the form is centered on its parent, otherwise it is centered on the desktop. Defaults to FALSE\$.
<b>IdealSize</b>	No	<p>This is an @Fm-delimited array specifying the desired coordinates and size to move the window to:</p> <ul style="list-style-type: none"><li>&lt;1&gt; Left-position (if -1 then the window is Centered on the X-axis)</li><li>&lt;2&gt; Top-position (if -1 then the window is centered On the Y-axis)</li><li>&lt;3&gt; Width (-1 means do not adjust the window width)</li><li>&lt;4&gt; Height (-1 means do not adjust the window height)</li></ul> <p>All these values default to -1.</p> <p><i>They must be in the same scale units as the form being centered.</i></p>
<b>CalculateOnly</b>	No	If TRUE\$ then the form is not moved or resized, but the resulting coordinates are returned instead. Defaults to FALSE\$.
<b>ParentSize</b>	No	<p>This is an @Fm-delimited array that can be used to override the size of the parent (if CenterParent is TRUE\$), or the Desktop (if CenterParent is FALSE\$).</p> <ul style="list-style-type: none"><li>&lt;1&gt; Left-position</li><li>&lt;2&gt; Top-position</li><li>&lt;3&gt; Width</li><li>&lt;4&gt; Height</li></ul> <p><i>They must be in the same scale units as the form being centered.</i></p>



**Options**      No      This is an @Fm-delimited array of options structured like so:

<1> Force boundary check. If this is TRUE\$ then the form is kept within the boundary of the desktop even if the IdealSize Top and Left positions have been explicitly specified.

<2> Desktop "anchor" form. Contains the name of a form to use when deciding which monitor to center the form on. The form is centered on the same desktop as the anchor form.

### Returns

An @Fm-delimited dynamic array containing the new size of the form in the same format as the standard SIZE property.

### Remarks

N/a.

### Example

```
// Center a form on the desktop
Call Exec_Method( @Window, "CENTER" )

// Center a form on its parent window
Call Exec_Method( @Window, "CENTER", TRUE$ )

// Center a form on the desktop with a specific size of 800x600
FormSize = -1 : @Fm : -1 : @Fm : 800 : @Fm : 600
Call Exec_Method( @Window, "CENTER", FALSE$, FormSize )

// Center a form on the desktop with a specific size of 800x600
// but only return the coordinates - do not update the form.
FormSize = -1 : @Fm : -1 : @Fm : 800 : @Fm : 600
NewSize = Exec_Method( @Window, "CENTER", FALSE$, FormSize, TRUE$ )

// Center a form on the same desktop as the RTI_IDE form and
// ensure it stays within the desktop boundary.
FormSize = -20 : @Fm : 10 : @Fm : 800 : @Fm : 600
Options = TRUE$
Options<2> = "RTI_IDE"
Call Exec_Method( @Window, "CENTER", FALSE$, FormSize, FALSE$, "", Options )
```

### See Also

Common GUI MONITOR property, Common GUI SCREENSIZE property, Common GUI SIZE property, SYSTEM MONITORLIST property.

## CLEARROW method

### Description

Clears the data from the controls in a data-bound form by triggering the form's CLEAR event.

### Syntax

```
Status = Exec_Method( CtrlEntID, "CLEARROW", SaveKey, SuppressWarning, |  
                    MaintainFocus )
```

### Parameters

Name	Required	Description
<b>SaveKey</b>	No	This is a boolean value. If TRUE\$ then the controls bound to key columns will not be cleared. The default is FALSE\$.
<b>SuppressWarning</b>	No	This is a boolean value. If TRUE\$ then the user will not be warned if they have unsaved changes in the form before it is cleared. The default is FALSE\$.
<b>MaintainFocus</b>	No	This is a boolean value. If TRUE\$ then the focus is not moved. By default it is moved to the first control in the tab-order.

### Returns

The CLEAR event status. If this is not null then an error has occurred or the user stopped the clear request).

### Remarks

N/a.

### Example

```
// Example - Clear the contents of the current form without any warning  
// and change the focus back to the first in the tab-order  
  
Call Exec_Method( @Window, "CLEARROW", FALSE$, TRUE$, FALSE$ )
```

### See also

WINDOW CLEAR event.

## CLOSE method

### Description

Closes a form by triggering its CLOSE event.

### Syntax

```
Status = Exec_Method( CtrlEntID, "CLOSE", CloseFlags, CloseAsync )
```

### Parameters

Name	Required	Description
<b>CloseFlags</b>	No	This is an @Fm delimited array with the following structure:  <1> If TRUE\$ then suppress any warnings with due to changed data if the form is data-bound.
<b>CloseAsync</b>	No	This is a boolean value. If TRUE\$ then the form is closed in an asynchronous manner. The default is FALSE\$.  (This is analogous to using Send_Event or Post_Event to trigger a CLOSE event in previous versions of OpenInsight)

### Returns

If the form is closed asynchronously a boolean value is returned; TRUE\$ if the CLOSE event request has made successfully, or FALSE\$ otherwise.

If the form is *not* closed asynchronously the event status from the underlying CLOSE event is returned - if this is not null an error has occurred (or the has user stopped the close request).

### Remarks

Care should be taken when closing a form from an event that is raised from one of the form's own child controls. In this case it is always better to set the CloseAsync flag to TRUE\$ so that the current event has chance to finish executing before the form is destroyed.

### Example

```
// Example - Close the current form asynchronously allowing for any data warnings  
Call Exec_Method( @Window, "CLOSE", FALSE$, TRUE$ )
```

***See also***

WINDOW CLOSE event, End\_Dialog stored procedure, End\_Window stored procedure.

## CLOSEDIALOG method

### Description

Closes a modal dialog box and returns a value back to the caller.

### Syntax

```
ClosedFlag = Exec_Method( CtrlEntID, "CLOSEDIALOG",RetVal )
```

### Parameters

Name	Required	Description
<b>RetVal</b>	No	Value to return to the caller.

### Returns

Returns TRUE\$ if the dialog box was closed, or FALSE\$ otherwise.

### Remarks

This method is a wrapper around the End\_Dialog stored procedure. Please see the End\_Dialog stored procedure description for more details.

### Example

```
// Example - Close the current dialog box returning the contents of the EDL_NAME  
// control to the caller  
  
RetName = Get_Property( @ Window : ".EDL_NAME", "TEXT" )  
bClosed = Exec_Method( @Window, "CLOSEDIALOG", RetName )
```

### See Also

WINDOW SHOWDIALOG method, Dialog\_Box stored procedure, End\_Dialog stored procedure.

## DELETEROW method

### Description

Deletes the currently loaded data row in a data-bound form by triggering the form's DELETE event.

### Syntax

```
Status = Exec_Method( CtrlEntID, "DELETEROW", SuppressWarning )
```

### Parameters

Name	Required	Description
<b>SuppressWarning</b>	No	This is a boolean value. If TRUE\$ then the user will not be asked to confirm if they wish to delete the row. The default is FALSE\$.

### Returns

The DELETE event status. If this is not null an error has occurred (or the user has stopped the delete request).

### Remarks

N/a.

### Example

```
// Example - Delete the contents of the current form unconditionally.  
Call Exec_Method( @Window, "DELETEROW", TRUE$ )
```

### See also

WINDOW DELETE event.

## FLASH method

### Description

"Flashes" a form's caption and/or taskbar button the specified number of times to draw the user's attention to it. The active state of the window is not changed.

### Syntax

```
ActiveFlag = Exec_Method( CtrlEntID, "FLASH", FlashCount, FlashCaption, |  
FlashTaskBar , FlashRate )
```

### Parameters

Name	Required	Description
<b>FlashCount</b>	No	An integer value specifying the number of times to flash the form, or flash it continuously (defaults to 1) <ul style="list-style-type: none"><li>• A value greater than 0 flashes it that many times.</li><li>• A value of -1 flashes the form until it is stopped.</li><li>• A value of -2 flashes the form until it is brought to the foreground.</li><li>• A value of 0 stops the form flashing.</li></ul>
<b>FlashCaption</b>	No	A boolean value – if TRUE\$ (the default) the form's caption bar is flashed.
<b>FlashTaskBar</b>	No	A boolean value – if TRUE\$ (the default) the form's taskbar button is flashed (if it has one).
<b>FlashRate</b>	No	A integer value specifying the flash rate in milliseconds. This defaults to the system cursor blink rate.

### Returns

Returns FALSE\$ if the form was inactive before the flash or TRUE\$ if it was active.

### Remarks

The FLASH method is implemented internally using the FlashWindowEx Windows API function, so please refer to the documentation on the Microsoft website for further information.

## Example

```
// Example - flash the current form continuously until it is activated.
```

```
Call Exec_Method( @Window, "FLASH", -2 )
```

## See also

WINDOW ACTIVE property.



## GETFOCUSEDCONTROL method

### Description

Returns the name of the control that has the focus if the form is active, or, if the form is not active, the name of the control that will receive the focus when it is.

### Syntax

```
FocusCtrlName = Exec_Method( CtrlEntID, "GETFOCUSEDCONTROL", NoInternal )
```

### Parameters

Name	Required	Description
<b>NoInternal</b>	No	If TRUE\$ then only controls <i>not</i> marked as "Internal" will be returned. Defaults to FALSE\$.

### Returns

The name of the control that is currently focused or will receive the focus when the form is activated.

### Remarks

Executing this method is essentially the same as using the "get" operation in the WINDOW FOCUS property to return the current focus control for the form. However, some objects, like the cell editor in an EDITTABLE control, are marked as "internal" and it may not be appropriate to use them when returned via the FOCUS property. In this case it is better to use the GETFOCUSEDCONTROL method and the NoInternal parameter to return the parent "non-internal" control instead.

### Example

```
$Insert Logical  
  
// Return the focused control for the current form, resolving it to a non-internal ID.  
  
FocusCtrlID = Exec_Method( @Window, "GETFOCUSEDCONTROL", TRUE$ )
```

### See Also

Common GUI object FOCUS property, SYSTEM FOCUS property, WINDOW FIRSTFOCUS property, WINDOW FOCUS property, WINDOW ACTIVATED event, WINDOW INACTIVATED event.

## HIDE method

### Description

Hides a form using the specified effect.

### Syntax

```
Call Exec_Method( CtrlEntID, "HIDE", HideEffect )
```

### Parameters

Name	Required	Description
<b>HideEffect</b>	No	If specified this should be a numeric value corresponding to a hide effect as defined in the HIDEEFFECT property.  If null or "-1" then the effect specified in the HIDEEFFECT property is used.

### Returns

N/a.

### Remarks

This method does not apply when used with an MDI Child form.

Equated constants for the HIDEEFFECT property value can be found in the PS\_WINDOW\_EQUATES insert record.

### Example

```
// Example - hide the current form using a "Slide Up" effect
$insert PS_Window_Equates

Call Exec_Method( @Window, "HIDE", PS_SHE_SLIDE_UP$ )
```

### See Also

WINDOW HIDEEFFECT property, WINDOW SHOWEFFECT property, WINDOW TRANSLUCENCY property, WINDOW VISIBLE property, WINDOW SHOW method.

## HIDEMENUBAR method

### Description

Hides the menubar for the specified form.



### Syntax

```
Call Exec_Method( CtrlEntID, "HIDEMENUBAR" )
```

### Parameters

N/a.

### Returns

N/a.

### Remarks

N/a.

### Example

```
// Example - hide the menubar for the current form  
Call Exec_Method( @Window, "HIDEMENUBAR" )
```

### See Also

MENUBAR object, WINDOW SHOWMENUBAR method

## MDICASCADE method

### Description

Arranges MDI child forms in a stacked cascading formation, ensuring the title bar of each is visible.

### Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "MDICASCADE", SkipDisabled, UseZOrder )
```

### Parameters

Name	Required	Description
<b>SkipDisabled</b>	No	This is a boolean value - if TRUE\$ then any disabled MDI child forms are excluded from the cascade operation.
<b>UseZOrder</b>	No	This is a boolean value - if TRUE\$ then the MDI Child forms are arranged in z-order.

### Returns

TRUE\$ if the cascade operation was successful, FALSE\$ otherwise.

### Remarks

This method only applies to MDI frame forms.

### Example

```
// Cascade the MDI child forms for the current MDI frame form, ignoring disabled forms.  
Call Exec_Method( @Window, "MDICASCADE", TRUE$ )
```

### See Also

WINDOW MDIFRAME property, WINDOW MDIICONARRANGE method, WINDOW MDITILE method.

## MDIICONARRANGE method

### Description

Arranges all minimized MDI child forms for the specified MDI frame form. Non-minimized MDI child forms are not affected.

### Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "MDIICONARRANGE" )
```

### Parameters

N/a.

### Returns

TRUE\$ if the operation was successful, FALSE\$ otherwise.

### Remarks

This method only applies to MDI frame forms.

This method is called by the system-level ARRANGEICONS event handler.

For more details on this method please see the documentation for the WM\_MDIICONARRANGE message on the Microsoft website.

### Example

```
// Arrange the minimized MDI child icons for the current MDI frame form
```

```
Call Exec_Method( @Window, "MDIICONARRANGE", TRUE$ )
```

### See Also

WINDOW MDIFRAME property, WINDOW MDICASCADE method , WINDOW MDITILE method, WINDOW ARRANGEICONS event.

## MDITILE method

### Description

Arranges MDI child forms in a tiled format. Each child is displayed in its entirety, overlapping none of the other child forms. All of the child forms are sized, as necessary, to fit within the MDI client area.

### Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "MDITILE", TileHorizontal, SkipDisabled )
```

### Parameters

Name	Required	Description
<b>TileHorizontal</b>	No	This is a boolean value - if TRUE\$ then the MDI Child forms are tiled horizontally, otherwise they are tiled vertically.
<b>SkipDisabled</b>	No	This is a boolean value - if TRUE\$ then any disabled MDI child forms are excluded from the tiling operation.

### Returns

TRUE\$ if the tiling operation was successful, FALSE\$ otherwise.

### Remarks

This method only applies to MDI frame forms.

The arguments passed to this method have been changed from previous versions of OpenInsight to support horizontal tiling and the "SkipDisabled" option simultaneously.

### Example

```
// Tile the MDI child forms vertically for the current MDI frame form,  
// ignoring disabled child forms.
```

```
Call Exec_Method( @Window, "MDITILE", FALSE$, TRUE$ )
```

### See Also

WINDOW MDIFRAME property, WINDOW MDICASCADE method, WINDOW MDIICONARRANGE method.

## QBFASKQUERY method

### Description

Asks the user for an RLIST query statement and loads the results into the specified form's QBF result list (QBFLIST property).

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFASKQUERY" )
```

### Parameters

N/a.

### Returns

The QBFQUERY event status. If this is not null then an error has occurred or the user has cancelled the operation.

### Remarks

This method triggers the form's QBFQUERY event.

The query statement entered must be a valid RLIST SELECT statement.

### Example

```
// Example - ask the user for an RLIST query  
  
Status = Exec_Method( @Window, "QBFASKQUERY" )  
If BLen( Status ) Then  
    // Error or cancelled ....  
End
```

### See also

WINDOW QBFLIST property, WINDOW QBFQUERY event, RLIST stored procedure.

## QBFCLOSESESSION method

### Description

Closes the QBF session for the specified form, returning it to normal operation.

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFCLOSESESSION" )
```

### Parameters

N/a.

### Returns

The QBFCLOSE event status. If this is not null then an error has occurred.

### Remarks

This method triggers the form's QBFCLOSE event. If successful, the form's QBFSTATUS property is set to "QBFINactive".

### Example

```
// Example - end the current form's QBF session.  
Status = Exec_Method( @Window, "QBFCLOSESESSION" )
```

### See also

WINDOW QBFSTATUS property, WINDOW QBFINITSESSION method, WINDOW QBFCLOSE event.



## QBFGOTO method

### Description

Loads a specified row from the form's QBF result list (QBFLIST property) using an index position.

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFGOTO", Position )
```

### Parameters

Name	Required	Description
<b>Position</b>	No	This is a numeric value which should be between 1 and the number of rows in the QBF result list.  If not specified, the user is asked to enter the position via a message box.

### Returns

The QBFABS event status. If this is not null then an error has occurred, or the user has cancelled the process.

### Remarks

This method triggers the form's QBFABS event.

The form must have a valid QBF result list active, i.e. the QBFSTATUS property must have a value of "QBFActive".

### Example

```
// Example - display the 10th row in the QBF result list  
  
Status = Exec_Method( @Window, "QBFGOTO", 10 )  
If BLen( Status ) Then  
    // Error ...  
End
```

**See also**

WINDOW QBFLIST property, WINDOW QBFPOS property, WINDOW QBFSTATUS property, WINDOW QBFABS event.

## QBFGOTOID method

### Description

Loads a specified row from the form's QBF result list (QBFLIST property) using a row ID.

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFGOTOID", RowID )
```

### Parameters

Name	Required	Description
<b>RowID</b>	No	This is key value that should be in the QBF result list.  If not specified, the user is asked to enter it directly via a message box.

### Returns

The QBFLOADID event status. If this is not null then an error has occurred, or the user has cancelled the process.

### Remarks

This method triggers the form's QBFLOADID event.

The form must have a valid QBF result list active, i.e. the QBFSTATUS property must have a value of "QBFActive".

### Example

```
// Example - display the row with the key of "A123" in the QBF result List  
  
Status = Exec_Method( @Window, "QBFGOTOID", "A123" )  
If BLen( Status ) Then  
    // Error ...  
End
```

### See also

WINDOW QBFLIST property, WINDOW QBFSTATUS property, WINDOW QBFLOADID event.

## QBFINITSESSION method

### Description

Initializes and begins a QBF session for the specified form. Any existing data is cleared, and normal data validation processing is temporarily removed from controls to allow easy data entry for the query.

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFINITSESSION" )
```

### Parameters

N/a.

### Returns

The QBFINIT event status. If this is not null then an error has occurred.

### Remarks

This method triggers the form's QBFINIT event. If successful, the form's QBSTATUS property is set to "QBFINitalize " to signify that the form is in "query entry" mode.

### Example

```
// Example - begin a QBF session for the current form.  
Status = Exec_Method( @Window, "QBFINITSESSION" )
```

### See also

WINDOW QBSTATUS property, WINDOW QBFCLOSESESSION method, WINDOW QBFINIT event.

## QBFLOADSAVEDLIST method

### Description

Asks the user for the name of a saved list and loads the keys into the specified form's QBF result list (QBFLIST property).

The source of the list may be chosen from the TCL Query Table or from the SYSLISTS table.

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFLOADSAVEDLIST" )
```

### Parameters

N/a.

### Returns

The QBFLOADLIST event status. If this is not null then an error has occurred or the user has cancelled the operation.

### Remarks

This method triggers the form's QBFLOADLIST event.

### Example

```
// Example - ask the user for the List of keys to Load.  
Status = Exec_Method( @Window, "QBFLOADSAVEDLIST" )
```

### See also

WINDOW QBFLIST property, WINDOW QBFLOADLIST event.

## QBFRUNQUERY method

### Description

Builds and executes an RLIST query from the data in the controls and populates the specified form's QBF result list (QBFLIST property).

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFRUNQUERY" )
```

### Parameters

N/a.

### Returns

The QBFRUN event status. If this is not null then an error has occurred.

### Remarks

This method triggers the form's QBFRUN event.

The form's QBSTATUS property must have been set to "QBFINitalize" by executing the QBFINITSESSION method first.

### Example

```
// Example - execute the QBF query process to find and load the results into the  
// current form:  
  
Status = Exec_Method( @Window, "QBFRUNQUERY" )
```

### See also

WINDOW QBFLIST property, WINDOW QBSTATUS property, WINDOW QBFINITSESSION method, WINDOW QBFRUN event.

## QBFSHOWFIRST method

### Description

Loads the first row from the specified form's QBF result list (QBFLIST property).

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFSHOWFIRST" )
```

### Parameters

N/a.

### Returns

The QBFFIRST event status. If this is not null then an error has occurred.

### Remarks

This method triggers the form's QBFFIRST event.

The form must have a valid QBF result list active, i.e. the QBFSTATUS property must have a value of "QBFActive".

### Example

```
// Example - display the first row in the QBF result list  
Status = Exec_Method( @Window, "QBFSHOWFIRST" )  
If BLen( Status ) Then  
    // Error ...  
End
```

### See also

WINDOW QBFLIST property, WINDOW QBFPOS property, WINDOW QBFSTATUS property, WINDOW QBFSHOWLAST method, WINDOW QBFSHOWNEXT method, WINDOW QBFSHOWPREV method, WINDOW QBFSHOWTABLE method, WINDOW QBFFIRST event.

## QBFSHOWLAST method

### Description

Loads the last row from the specified form's QBF result list (QBFLIST property).

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFSHOWLAST" )
```

### Parameters

N/a.

### Returns

The QBFLAST event status. If this is not null then an error has occurred.

### Remarks

This method triggers the form's QBFLAST event.

The form must have a valid QBF result list active, i.e. the QBFSTATUS property must have a value of "QBFActive".

### Example

```
// Example - display the last row in the QBF result list  
  
Status = Exec_Method( @Window, "QBFSHOWLAST" )  
If BLen( Status ) Then  
    // Error ...  
End
```

### See also

WINDOW QBFLIST property, WINDOW QBFPOS property, WINDOW QBFSTATUS property, WINDOW QBFSHOWFIRST method, WINDOW QBFSHOWNEXT method, WINDOW QBFSHOWPREV method, WINDOW QBFSHOWTABLE method, WINDOW QBFLAST event.



## QBFSHOWNEXT method

### Description

Loads the next row from the specified form's QBF result list (QBFLIST property).

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFSHOWNEXT" )
```

### Parameters

N/a.

### Returns

The QBFNEXT event status. If this is not null then an error has occurred.

### Remarks

If the current position is at the end of the QBF result list then it is reset to point at the first row instead.

This method triggers the form's QBFNEXT event.

The form must have a valid QBF result list active, i.e. the QBFSTATUS property must have a value of "QBFActive".

### Example

```
// Example - display the next row in the QBF result list  
  
Status = Exec_Method( @Window, "QBFSHOWNEXT" )  
If BLen( Status ) Then  
    // Error ...  
End
```

### See also

WINDOW QBFLIST property, WINDOW QBFPOS property, WINDOW QBFSTATUS property, WINDOW QBFSHOWFIRST method, WINDOW QBFSHOWLAST method, WINDOW QBFSHOWPREV method, WINDOW QBFSHOWTABLE method, WINDOW QBFNEXT event.

## QBFSHOWPREV method

### Description

Loads the previous row from the specified form's QBF result list (QBFLIST property).

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFSHOWPREV" )
```

### Parameters

N/a.

### Returns

The QBFPREV event status. If this is not null then an error has occurred.

### Remarks

If the current position is at the start of the QBF result list then it is reset to point at the last row instead.

This method triggers the form's QBFPREV event.

The form must have a valid QBF result list active, i.e. the QBFSTATUS property must have a value of "QBFActive".

### Example

```
// Example - display the previous row in the QBF result list  
  
Status = Exec_Method( @Window, "QBFSHOWPREV" )  
If BLen( Status ) Then  
    // Error ...  
End
```

### See also

WINDOW QBFLIST property, WINDOW QBFPOS property, WINDOW QBFSTATUS property, WINDOW QBFSHOWFIRST method, WINDOW QBFSHOWLAST method, WINDOW QBFSHOWNEXT method, WINDOW QBFSHOWTABLE method, WINDOW QBFPREV event.

## QBFSHOWTABLE method

### Description

Displays the QBF result list in a non-modal dialog box. As the user selects a value in the dialog the corresponding row is loaded in the owning form.

### Syntax

```
Status = Exec_Method( CtrlEntID, "QBFSHOWTABLE" )
```

### Parameters

N/a.

### Returns

The QBFTABLE event status. If this is not null then an error has occurred.

### Remarks

This method triggers the form's QBFTABLE event.

The form must have a valid QBF result list active, i.e. the QBFSTATUS property must have a value of "QBFActive".

### Example

```
// Example - display the QBF result List  
  
Status = Exec_Method( @Window, "QBFSHOWTABLE" )  
If BLen( Status ) Then  
    // Error ...  
End
```

### See also

WINDOW QBFLIST property, WINDOW QBFPOS property, WINDOW QBFSTATUS property, WINDOW QBFSHOWFIRST method, WINDOW QBFSHOWLAST method, WINDOW QBFSHOWNEXT method, WINDOW QBFPREV event.

## READPREVROW method

### Description

Populates controls in the specified form with data from the previously loaded row.

### Syntax

```
Status = Exec_Method( CtrlEntID, "READPREVROW", ControlList )
```

### Parameters

Name	Required	Description
<b>ControlList</b>	No	If specified this an @Fm-delimited array of data-bound control names that should be loaded from the cached data.  If this parameter is null then all data-bound controls will be loaded from the cached data.

### Returns

Error status results. If this is not null then an error has occurred.

### Remarks

A data-bound form caches row data at the following points:

- When data is written to the database – a copy of the data in the controls is saved.
- When data is loaded into the controls during a read operation. This is optional and only happens if the LOADPREVALWAYS property is TRUE\$ : if so a copy of the data loaded into the controls during a READ event is cached (this is always overwritten by data from a write operation of course).

The READPREVROW method uses this cached data to load controls when requested.

Controls bound to key columns are ignored.

This method is intended to mimic the "Alt-C" functionality of found in Advanced Revelation applications.

## Example

```
// Example - Load data from a previous row into the control that currently  
// has the focus. This sort of functionality would typically be found on a  
// form's Edit menu.
```

```
CtrlID = Get_Property( @Window, "FOCUS" )  
If BLen( CtrlID ) Then  
  ColPos = Get_Property( CtrlID, "POS" )  
  If ColPos Then  
    // CtrlID is a non-key databound control  
    Status = Exec_Method( @Window, "READPREVROW", CtrlID )  
  End  
End
```

## See Also

WINDOW LOADPREVALWAYS property, WINDOW ROW property, WINDOW READ event, WINDOW WRITE event.

## READROW method

### Description

Reads the data into the specified data-bound form and populates the controls.

### Syntax

```
Status = Exec_Method( CtrlEntID, "READROW", RowID, SuppressWarning )
```

### Parameters

Name	Required	Description
<b>RowID</b>	No	<p>If specified this is the ID of the row to be loaded into the form.</p> <p>For a multi-table form RowID should be an @fm-delimited array of keys that matches the order of the tables in the form's join specification:</p> <p>i.e.</p> <pre>rowID&lt;1&gt; = Key for primary table rowID&lt;2&gt; = Key for first subsidiary table rowID&lt;3&gt; = Key for second subsidiary table rowID&lt;n&gt; = Key for nth subsidiary table</pre> <p>etc.</p> <p>If null (the default) then the data entered in the key-controls is extracted to build the RowID used to load the row.</p>
<b>SuppressWarning</b>	No	<p>If TRUE\$ then the SAVEWARN property is not checked before a new row is loaded when rowID is specified.</p> <p>The default is FALSE\$, which means the SAVEWARN property will be processed as normal.</p> <p>This parameter is ignored is RowID is null.</p>

### Returns

The READ event status. If this is not null then an error has occurred or the user has cancelled the process.

### Remarks

This method triggers the form's READ event.

## Example

```
// Example - Read a row into the current form unconditionally  
  
RowID = "AS123*EF"  
  
ReadStatus = Exec_Method( @Window, "READROW", RowID, TRUE$ )  
If BLen( ReadStatus ) Then  
    // Error or cancelled...  
End
```

## See Also

WINDOW ID property, WINDOW ROW property, WINDOW SAVEWARN property,  
WINDOW WRITEROW method, WINDOW READ event.

## SCROLL method

### Description

Scrolls the contents of the specified form's client area.

### Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "SCROLL", XShift, YShift )
```

### Parameters

Name	Required	Description
XShift	Yes	Integer value specifying the amount of horizontal scrolling.
YShift	Yes	Integer value specifying the amount of vertical scrolling.

### Returns

TRUE\$ if the client area was scrolled successfully, or FALSE\$ otherwise.

### Remarks

The XShift and YShift values are interpreted as DIPs or PX based on the form's SCALEUNITS property.

The SCROLL method is implemented internally using the ScrollWindow Windows API function. More information on this function can be found on the Microsoft website.

### Example

```
// Example - scroll the form's contents by 100 DIPs vertically  
Call Exec_Method( @Window, "SCROLL", 0, 100 )
```

### See Also

N/a.



## SHOW method

### Description

Displays a form using the specified effect.

### Syntax

```
Call Exec_Method( CtrlEntID, "SHOW", ShowEffect )
```

### Parameters

Name	Required	Description
<b>ShowEffect</b>	No	If specified this should be a numeric value corresponding to a show effect as described in the SHOWEFFECT property.  If null or "-1" then the effect specified in the SHOWEFFECT property is used.

### Returns

N/a.

### Remarks

This method does not apply when used with an MDI Child form.

Equated constants for the SHOWEFFECT property value can be found in the PS\_WINDOW\_EQUATES insert record.

### Example

```
// Example - shown the current form using a "Slide Down" effect
$insert PS_Window_Equates

Call Exec_Method( @Window, "SHOW", PS_SHE_SLIDE_DOWN$ )
```

### See Also

WINDOW HIDEFFECT property, WINDOW SHOWEFFECT property, WINDOW TRANSLUCENCY property, WINDOW VISIBLE property, WINDOW HIDE method.

## SHOWDIALOG method

### Description

Displays a dialog box using the specified form as the owner.

### Syntax

```
RetVal = Exec_Method( CtrlEntID, "SHOWDIALOG", DialogName, CreateParam, |  
Options, AsyncParams )
```

### Parameters

Name	Required	Description
<b>DialogName</b>	Yes	Name of the form to execute as a dialog box. Must be in upper-case.
<b>CreateParam</b>	No	Data to pass to the form's CREATE event. This data is passed as the "CreateParam" argument when the CREATE event is triggered. It must <i>not</i> contain any @Rm system delimiter characters.
<b>Options</b>	No	A dynamic array of extra options for launching the form:  <1> If TRUE\$ then disable ALL other forms owned by the owner form, not just the owner form itself.  <2> GETPARENTFORM override. The Dialog_Box stored procedure uses the WINDOW object GETPARENTFORM method internally. This option allows it to be tweaked as desired.
<b>AsyncParams</b>	No	A dynamic array of options that control Asynchronous mode. When executed in this mode the returned data is not passed directly back to the calling stored procedure – rather it is passed back via the owner's ENDDIALOG event.  <1> If TRUE\$ then the dialog will be executed in Asynchronous mode.  <2> Contains a string argument passed back to the owner's ENDDIALOG event. Useful for identifying the returned data.

### Returns

If the dialog box is executed in synchronous mode the return value is the data passed back from the CLOSEDIALOG method (or an End\_Dialog function call).

If the dialog is executed in asynchronous mode the return value will be the instance ID of the created dialog box. In this case the data passed back from an End\_Dialog call will be passed as an argument to the owner form's ENDDIALOG event.

If an error occurs the return value will be null.

### Remarks

This method is a wrapper around the Dialog\_Box stored procedure. Please see the Dialog\_Box stored procedure description for more details.

### Example

```
// Example - Display a dialog box in synchronous mode using the  
// current form as the owner and passing it the contents of a  
// variable called CurrName as the CreateParam.  
  
NewName = Exec_Method( @Window, "SHOWDIALOG", "MY_DIALOG_BOX", CurrName )  
  
If BLen( NewName ) Then  
    // The user entered a new name so process it  
    Call Do_Something_With_This_Name( NewName )  
End
```

### See Also

WINDOW CLOSEDIALOG method, WINDOW STARTFORM method, Dialog\_Box stored procedure.

## SHOWINDEXLOOKUP method

### Description

Displays an Index Lookup dialog box using the specified form as the owner.

### Syntax

```
RetVal = Exec_Method( CtrlEntID, "SHOWINDEXLOOKUP", IndexedTable, |
                    SearchColumns, DisplayColumns, SelMode )
```

### Parameters

Name	Required	Description
<b>IndexedTable</b>	Yes	The indexed table to search. This table must have at least one Btree Index on it.
<b>SearchColumns</b>	Yes	Contains an @Fm-delimited list of column names to display in the dialog that the user may search.
<b>DisplayColumns</b>	Yes	Contains an @Fm-delimited list of columns names to show in the results popup when a user chooses the row(s) to load into the form.
<b>SelMode</b>	No	Contains an @fm delimited list of selection options:  <1> Mode - this can be either "MULTI" or "SINGLE" (the default). If "MULTI" then the user may select more than one row to return.  <2> Contains the name of the control to return the selected row IDs to. If null then the row IDs are loaded into the form as a QBF result list.  This can be a virtual name like "@WINDOW" or "@SELF" that are used in normal quick event programming.  <3> If a control name is specified in field <2> this field contains the name of the property to set such as "TEXT"

### Returns

The IXLOOKUP event status. If this is not null then an error has occurred or the user has cancelled the process.

## Remarks

This method triggers the form's IXLOOKUP event.

## Example

```
// Example - Launch the index lookup dialog using the current window as the owner,  
// and put the returned key into the EDL_CUST_ID control  
  
TableName   = "CUSTOMERS"  
SearchCols  = "SURNAME" : @Fm : "STREET" : @Fm : "CITY"  
DisplayCols = "FULL_NAME" : @Fm : "FULL_ADDRESS"  
  
SelMode     = "SINGLE"  
SelMode<2>  = @Window : ".EDL_CUST_ID"  
SelMode<3>  = "TEXT"  
  
Status      = Exec_Method( @Window, "SHOWINDEXLOOKUP", |  
                          TableName,                |  
                          SearchCols,              |  
                          DisplayCols,             |  
                          SelMode )
```

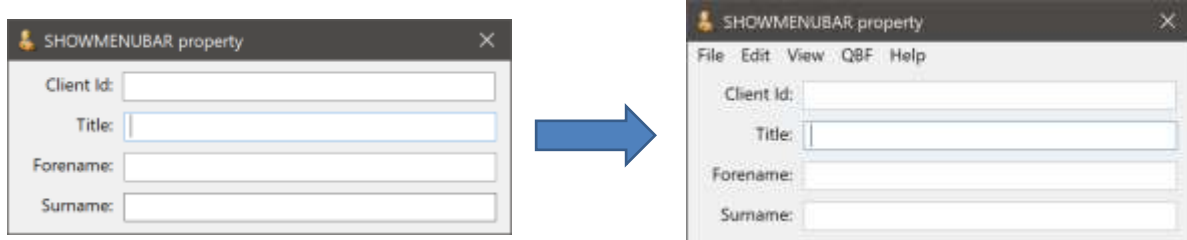
## See Also

WINDOW STARTDIALOG method, WINDOW SHOWMESSAGE method, WINDOW SHOWPOPUP method, WINDOW IXLOOKUP event.

## SHOWMENUBAR method

### Description

Shows the menubar for the specified form.



### Syntax

```
Call Exec_Method( CtrlEntID, "SHOWMENUBAR" )
```

### Parameters

N/a.

### Returns

N/a.

### Remarks

N/a.

### Example

```
// Example - show the menubar for the current form  
Call Exec_Method( @Window, "SHOWMENUBAR" )
```

### See Also

MENUBAR object, WINDOW HIDEMENUBAR method

## SHOWMESSAGE method

### Description

Displays a message box, using the specified form as the parent.

### Syntax

```
MessageValue = Exec_Method( CtrlEntID, "SHOWMESSAGE", MessageStruct, |  
                             MessageName )
```

### Parameters

Name	Required	Description
<b>MessageStruct</b>	Maybe	This is an @Fm-delimited array that contains a message structure as per the Msg() stored procedure.  Required if MessageName is null.
<b>MessageName</b>	Maybe	Contains the name of an OpenInsight repository MSG entity to display. The fields in the MessageStruct parameter override the fields from the stored entity.  Required if MessageStruct is null.

### Returns

The value as specified by the message definition.

### Remarks

This method is basically a wrapper around the Msg stored procedure. Please see the documentation on Msg for more information.

### See also

SYSTEM SHOWMESSAGE method, Msg stored procedure.

## Example

```
// Example - display a simple message

$insert Msg_Equates
MsgStruct      = ""
MsgStruct<MTEXT$> = "WINDOW SHOWMESSAGE method example "
MsgStruct<MICON$> = "*"
MsgStruct<MJUST$> = "C"
MsgStruct<MCAPTION$> = "Owned Message Box"

MsgVal = Exec_Method( CtrlEntID, "SHOWMESSAGE", MsgStruct )

// Example display a message with an entity name
MsgVal = Exec_Method( CtrlEntID, "SHOWMESSAGE", "", "OI_ABOUT" )
```

## See Also

SYSTEM SHOWMESSAGE method, WINDOW STARTDIALOG method, WINDOW SHOWINDEXLOOKUP method, WINDOW SHOWPOPOP method, Msg stored procedure.



## SHOWPOPUP method

### Description

Displays a popup box using the specified form as the owner.

### Syntax

```
RetVal = Exec_Method( CtrlEntID, "SHOWPOPUP", PopupStruct, PopupName )
```

### Parameters

Name	Required	Description
<b>PopupStruct</b>	No	Contains the popup definition structure. This is an @Fm-delimited array of values that define the popup as per the Popup stored procedure.  This parameter is required if PopupName is null.
<b>MessageName</b>	No	Contains the name of a valid POPUP entity as stored in the OpenInsight repository. This can be the fully qualified entity ID, or just the name of the popup, e.g.  MYAPP*POPUP**MYPOPUP or MYPOPUP  This parameter is required if PopupStruct is null.

### Returns

The value as defined by the popup structure. See the Popup stored procedure for more details.

### Remarks

This method is a wrapper around the Popup stored procedure. Please see the Popup stored procedure description and the POPUP\_EQUATES insert record for more details.

### Example

```
// Example - Display the ASCII_CHART popup box.  
PopupVal = Exec_Method( @Window, "SHOWPOPUP", "", "ASCII_CHART" )
```

### **See Also**

SYSTEM SHOWPOPUP method, WINDOW STARTDIALOG method, WINDOW SHOWINDEXLOOKUP method, WINDOW SHOWMESSAGE method, Popup stored procedure.

## STARTFORM method

### Description

Executes a new form, using the specified form as the owner.

### Syntax

```
RetVal = Exec_Method( CtrlEntID, "STARTFORM", FormName, CreateParam )
```

### Parameters

Name	Required	Description
<b>FormName</b>	Yes	Name of the form to execute. Must be in upper-case.
<b>CreateParam</b>	No	Data to pass to the form's CREATE event. This data is passed as the "CreateParam" argument when the CREATE event is triggered.

### Returns

The instance name of the newly created form, or null if the form cannot be started. The instance ID is usually the same as the passed FormName, but if the form is flagged as multi-instance then the PS can append a unique number (delimited with an "\*" character) to the returned ID to ensure that there are no conflicts with existing forms.

### Remarks

This method is a wrapper around the Start\_Window stored procedure. Please see the Start\_Window stored procedure description for more details.

### Example

```
// Example - Display a form using the current form as the owner and passing  
// it the contents of a variable called SearchVar as the CreateParam.  
  
FormID = Exec_Method( @Window, "STARTFORM", "QUICK_SEARCH", SearchVar )  
  
If BLen( FormID ) Else  
    // Error!....  
End
```

### **See Also**

SYSTEM STARTFORM method, WINDOW SHOWDIALOG method, WINDOW STARTMDICHILDFORM method, Start\_Window stored procedure.

## STARTMDICHILDFORM method

### Description

Executes a new MDI child form for the specified MDI Frame form.

### Syntax

```
RetVal = Exec_Method( CtrlEntID, "STARTMDICHILDFORM", FormName, CreateParam | AppearanceMode, InitX, InitY )
```

### Parameters

Name	Required	Description
<b>FormName</b>	Yes	Name of the child form to execute. Must be in upper-case.
<b>CreateParam</b>	No	Data to pass to the child form's CREATE event. This data is passed as the "CreateParam" argument when the CREATE event is triggered.
<b>AppearanceMode</b>	No	Specifies how the child form should be displayed. Can be one of the following values:  0 : Displays in the same way as the currently active child (this is the default) 1 : Normal 2 : Minimized 3 : Maximized
<b>InitX</b>	No	The initial X position of the child in the parent form's MDI Client area.
<b>InitY</b>	No	The initial Y position of the child in the parent form's MDI Client area.

### Returns

The Instance ID of the newly created form is returned if successful. Null is returned if the form fails to start. The instance ID is usually the same as the passed FormName, but if the form is flagged as multi-instance then the PS can append a unique number (delimited with an "\*" character) to the returned ID to ensure that there are no conflicts with existing forms.

### Remarks

This method is only supported for MDI Frame forms.

This method is a wrapper around the Start\_MDICChild stored procedure. Please see the Start\_MDICChild stored procedure description for more details.

### Example

```
// Example - Start a maximized MDI Child form in an MDI Frame form, passing
//           an ID to load in the child's CREATE event.

$insert MSWin_ShowWindow_Equates

CustID = "A12345"
ChildID = Exec_Method( @Window, "STARTMDICHILDFORM",
                      "CUSTOMER_ENTRY",
                      CustID,
                      SW_SHOWMAXIMIZED$, "", "" )
```

### See Also

WINDOW MDIFRAME property, SYSTEM STARTFORM method, WINDOW SHOWDIALOG method, WINDOW STARTMDICHILDFORM method, Start\_MDICChild stored procedure.

## TRACKDROPDOWNMENU method

### Description

Displays a dropdown menu for a top-level menu bar item on the specified form. The menu is created and displayed from the passed menu item structure.

### Syntax

```
Status = Exec_Method( CtrlEntID, "TRACKDROPDOWNMENU", MenuItemID, |  
MenuStruct )
```

### Parameters

Name	Required	Description
<b>MenuItemID</b>	Yes	Contains the fully qualified name of the top-level menu item to "drop-down".
<b>MenuStruct</b>	Yes	A dynamic array containing the executable structure of the menu.  Note that this structure does <i>not</i> include the usual menu header fields.

### Returns

TRUE\$ if the menu is created and displayed successfully, or FALSE\$ otherwise.

### Remarks

This method is called by the DROPDOWNMENU event to display the dropdown menu associated with a top-level menu item.

This is considered a low-level method. It is better to make any adjustments to the menu structure in the DROPDOWN menu event handler instead.

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.

## Example

```
// Display a droppdown menu for the current form's "EXAMPLE" top-level menu item  
// with two items and a separator
```

```
$Insert PS_Menu_Equates  
$insert MSWin_Menu_Equates
```

```
MenuStruct = ""  
MenuID     = @Window : ".MENU.EXAMPLE"
```

```
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( @Window, "TRACKDROPDOWNMENU", MenuID, MenuStruct )
```

## See also

Common GUI MENU event, WINDOW DROPDOWNMENU event.



## UPDATEROW method

### Description

Updates the data associated with the primary table of a data-bound form *without* updating data-bound controls.

### Syntax

```
SuccessFlag = Exec_Method( CtrlEntID, "UPDATEROW", NewRow )
```

### Parameters

Name	Required	Description
<b>NewRow</b>	Yes	New data for the row.

### Returns

The TRUE\$ if the row was updated successfully, or FALSE\$ otherwise.

### Remarks

This method is essentially the same as setting the ROW property except that the controls are *not* updated. This can be useful when updating data columns that are not bound to a control without unnecessarily re-populating those that are, such as just before a WRITE operation for example. SAVEWARN will be set to TRUE\$.

### Example

```
// Example - Update column <20> of the current row without reloading  
//           any data-bound controls  
  
$Insert Logical  
  
RowData      = Get_Property( @Window, "ROW" )  
RowData<20> = TRUE$  
  
Call Exec_Method( @Window, "UPDATEROW", RowData )
```

### See Also

WINDOW ROW property, WINDOW ID property, WINDOW WRITE event.

## WRITEROW method

### Description

Writes the data contains in the specified data-bound form to the database.

### Syntax

```
Status = Exec_Method( CtrlEntID, "READROW", RowID )
```

### Parameters

Name	Required	Description
<b>RowID</b>	No	<p>If specified this is the ID to use when writing the data to the database. If this is different to the ID of the currently loaded row a "Save As" operation is performed, and the data in the key controls updated to reflect this.</p> <p>For a multi-table form RowID should be an @fm-delimited array of keys that matches the order of the tables in the form's join specification:</p> <p>i.e.</p> <pre>rowID&lt;1&gt; = Key for primary table rowID&lt;2&gt; = Key for first subsidiary table rowID&lt;3&gt; = Key for second subsidiary table rowID&lt;n&gt; = Key for nth subsidiary table</pre> <p>etc.</p> <p>If null (the default) then the data entered in the key-controls is extracted to build the RowID used to write the row.</p>

### Returns

The WRITE event status. If this is not null then an error has occurred or the user has cancelled the process.

### Remarks

This method triggers the form's WRITE event.

## Example

```
// Example - Write the data in the current form using a new ID  
  
RowID = "AS123*EF"  
  
WriteStatus = Exec_Method( @Window, "WRTEROW", RowID )  
If BLen( WriteStatus ) Then  
    // Error or cancelled...  
End
```

## See Also

WINDOW CLEARONWRITE property, WINDOW ID property, WINDOW ROW property, WINDOW READROW method, WINDOW, CLEAR event, WINDOW WRITE event.

## WINDOW Events

The WINDOW object supports the following events:

Name	Description
<b>ACTIVATED</b>	Occurs when a form is activated.
<b>ARRANGEICONS</b>	Arranges all minimized MDI Child forms for an MDI frame form.
<b>CASCADE</b>	Arranges all MDI Child forms into a cascading layout for an MDI frame form.
<b>CLEAR</b>	Occurs when data is cleared from a data-bound form.
<b>CLOSE</b>	Occurs when a form is being closed.
<b>CREATE</b>	Occurs when a form is created.
<b>DBLCLK</b>	Occurs when a user double-clicks the mouse on a form.
<b>DELETE</b>	Occurs when data is cleared from a data-bound form.
<b>DROPDOWNMENU</b>	Occurs when a top-level item on a form's menu bar is selected.
<b>ENDDIALOG</b>	Occurs when an asynchronous dialog box returns a value to its owner form.
<b>FORMSTATECHANGED</b>	Occurs when the "form state" of a form changes.
<b>INACTIVATED</b>	Occurs when a form becomes inactive.
<b>IXLOOKUP</b>	Occurs when a form shows an Index Lookup dialog box.
<b>MDICHILDSTATECHANGED</b>	Occurs when the "form state" of an MDI Child form changes.
<b>MDISELECT</b>	Occurs when a user selects the "More Windows..." item from the "Window" menu on an MDI Frame form.
<b>PAGE</b>	Pseudo-method used to change the current page of a form.
<b>QBFABS</b>	Occurs when the form loads a row in a QBF result list using a position index.
<b>QBFCLOSE</b>	Occurs when a form's QBF session is closed.
<b>QBFFIRST</b>	Occurs when the form loads the first row in a QBF result list.
<b>QBFINIT</b>	Occurs when a QBF session is started for a form.
<b>QBFLAST</b>	Occurs when the form loads the last row in a QBF result list.
<b>QBFLOADID</b>	Occurs when the form loads a row from the QBF result list using a specified key.
<b>QBFLOADLIST</b>	Occurs when the form needs to obtain the name of a saved list of keys to load into its QBF result list.
<b>QBFNEXT</b>	Occurs when the form loads the next row in a QBF result list.
<b>QBFPREV</b>	Occurs when the form loads the previous row in a QBF result list.
<b>QBFQUERY</b>	Occurs when the user wished to execute a "raw" QBF SELECT statement

<b>QBFRUN</b>	Occurs when the form uses the entered query data to search the database and load the QBF result list.
<b>QBFTABLE</b>	Occurs when a form's QBF result list is about to be displayed.
<b>READ</b>	Occurs when data is read from the database into a data-bound form.
<b>SCALED</b>	Occurs when the scaling factor of a form changes.
<b>TILE</b>	Arranges all MDI Child forms into a tiled layout for an MDI frame form.
<b>VISUALSTYLECHANGED</b>	Occurs when Window's visual styling changes.
<b>WRITE</b>	Occurs when data is written from a data-bound form to the database.

The following Common GUI Object events are also supported:

- BUTTONDOWN
- BUTTONUP
- CONTEXTMENU
- DROPFILES
- HELP
- HSCROLL
- INITCONTEXTMENU
- LOSTCAPTURE
- MOUSEMOVE
- NOTES
- OMNIEVENT
- OPTIONS
- SYSMSG
- TIMER
- VSCROLL
- WINMSG

The following Container Object API events are also supported:

- PAGECHANGED
- SIZE

## ACTIVATED event

### Description

Occurs when a form is activated.

### Syntax

```
bForward = ACTIVATED( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form object receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 event handler for ACTIVATED raises a FORMSTATECHANGED event.

For more information on this event please refer to the Windows documentation regarding WM\_ACTIVATE window message on the Microsoft website.

### Example

```
Function ACTIVATED( CtrlEntID, CtrlClassID )

    // Example - ACTIVATED event for an MDI child form to let it parent frame
    // know that it has been activated so it may set its menu items correctly

    MDIFrame = Get_Property( CtrlEntID, "MDIFRAME" )

    // Assume that the frame uses an OMNIEVENT handler to respond to the
    // child being activated with a message of "CHILD_ACTIVATED" and the
    // child ID as the first parameter

    EvStatus = Exec_Method( MDIFrame, "SENDEVENT", "OMNIEVENT", |
        "CHILD_ACTIVATED", CtrlEntID )

Return TRUE$
```

### **See Also**

Common GUI FOCUS property, SYSTEM FOCUS property, WINDOW ACTIVE property, WINDOW FOCUS property, WINDOW FORMSTATECHANGED event, WINDOW INACTIVATED event.

## ARRANGEICONS event

### Description

Arranges all minimized MDI Child forms for an MDI frame form.

### Syntax

```
bForward = ARRANGEICONS( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW")

### 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 this event calls the MDIICONARRANGE method to perform the arrange operation, and because of this it has been deprecated in favor of that method. It is implemented only for backwards compatibility with earlier versions of OpenInsight.

(Note that this event is not a "true" event as such as it is never triggered by the PS, it can only be "manually" triggered by the developer in an application (typically from an MDI Window menu item) – it is actually a method masquerading as an event).

Please see the documentation for the WM\_MDIICONARRANGE message on the Microsoft website for more details.

### Example

N/a.

### See Also

WINDOW MDIICONARRANGE method.



## CASCADE event

### Description

Arranges all MDI Child forms into a cascading layout for an MDI frame form.

### Syntax

```
bForward = CASCADE( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 this event calls the MDICASCADE method to perform the cascade operation, and because of this it has been deprecated in favor of that method. It is implemented only for backwards compatibility with earlier versions of OpenInsight.

(Note that this event is not a "true" event as such as it is never triggered by the PS, it can only be "manually" triggered by the developer in an application (typically from an MDI Window menu item) – it is actually a method masquerading as an event).

Please see the documentation for the WM\_MDICASCADE message on the Microsoft website for more details.

### Example

N/a.

### See Also

WINDOW MDICASCADE method.

## CLEAR event

### Description

Occurs when data is cleared from a data-bound form.

### Syntax

```
bForward = CLEAR( CtrlEntID, CtrlClassID, bSaveKey, bSuppressWarning |  
                  bMaintainFocus )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>bSaveKey</b>	This is a boolean value. If TRUE\$ then the controls bound to key columns will not be cleared.
<b>bSuppressWarning</b>	This is a boolean value. If TRUE\$ then the user will not be warned if they have unsaved changes in the form before it is cleared.
<b>bMaintainFocus</b>	This is a boolean value. If TRUE\$ then the focus is not moved. By default it is moved to the first control in the tab-order.

### 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 CLEAR event has a system-level handler that performs the following tasks:

- If the form is cleared while browsing a QBF result list, then a QBFNEXT event is raised to move to the next record instead.
- If the form is cleared while the QBFSTATUS property is QBFInitialize then a QBFCLOSE event is raised to clear the form instead.

If neither of these conditions apply the normal CLEAR process is followed:

- If the form's SUPPRESSSAVEWARN property is FALSE\$ *and* the bSuppressWarning parameter is FALSE\$ then:
  - Update the form's SAVEWARN property if data in the currently focused control has changed.

- Check the SAVEWARN property and warns the user of any unsaved changes if the bSuppressWarning flag is FALSE allowing the operation to be cancelled. If so, then the clear operation is stopped here.
- Reset the form's SAVEWARN property to FALSE\$
- Unlock any data-rows locked by the form.
- If bSaveKey is FALSE\$ then clear data in the data-bound key controls
- Clear the contents of the data-bound controls.
- Reset the form's NEWROW property to FALSE\$
- If bMaintainFocus is FALSE\$ move the focus to the first control in the form's tab order.

A CLEAR event will be triggered by the system in the following circumstances:

- From a READ event, if the read operation is cancelled or fails.
- From a WRITE event, if the CLEARONWRITE property is TRUE\$.
- From a DELETE event.

Applications needing to execute a clear operation programmatically should use the WINDOW CLEARROW method rather than using the Send\_Event stored procedure to invoke a CLEAR event directly (as was the case in previous versions of OpenInsight).

### Example

```
Function CLEAR( CtrlEntID, CtrlClassID, bSaveKey, bSuppressWarning, bMaintainFocus )

// Example - CLEAR event script that removes some information from a STATIC
// control called TXT_INFO that is not data-bound.
$insert RTI_SSP_Equates

// First let the system event handler perform the clear in case the user cancels it
Call Set_EventStatus( SETSTAT_OK$ )
Call Forward_Event( bSaveKey, bSuppressWarning, bMaintainFocus )
If Get_EventStatus() Then
    // Assume cancelled or error
    Null
End Else
    Call Set_Property_Only( @Window : ".TXT_INFO", "TEXT", "" )
End

// Return FALSE$ to stop the event chain as we've already forwarded to the
// system CLEAR event handler above.

Return FALSE$
```

### See Also

WINDOW CLEARROW method, WINDOW READ event, WINDOW SYSMMSG event, WINDOW WRITE event.

## CLOSE event

### Description

Occurs when a form is being closed.

### Syntax

```
bForward = CLOSE( CtrlEntID, CtrlClassID, CancelFlag, CloseFlags )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>CancelFlag</b>	A boolean value used to check the response from SYMSG calls. This is always set to FALSE\$ when the CLOSE event is executed, it can be set to TRUE\$ if the user cancels the CLOSE attempt.
<b>CloseFlags</b>	This is an @Fm-delimited array with this following structure (currently there is only one field):  <1> If TRUE\$ then suppress SAVEWARN processing

### 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 CLOSE event handler performs the following tasks:

- If the suppress SAVEWARN flag is not set (or the form's SUPPRESSSAVEWARN property is FALSE\$) then:
  - Update the form's SAVEWARN property if data in the currently focused control has changed.
  - Check the SAVEWARN property and warn the user of any unsaved changes, giving them the opportunity to save, ignore or cancel:
    - Choosing save performs a write operation and proceeds to close the form if successful, otherwise the close operation is stopped.
    - Choosing cancel stops the form from closing.
- The CLOSE quick event handler is executed (if defined) and the event status checked – if it returns anything other than FALSE\$ the operation is cancelled.

- Any forms owned by the current form are closed. If any owned forms cannot be closed the entire close operation is cancelled.
- If the current form is an MDI Frame form, then any MDI Child forms are closed. If any child forms cannot be closed the entire close operation is cancelled.
- All database locks held by the form are released.
- The form is removed from screen via the HIDE method.
- The form is terminated via the End\_Window stored procedure.

Care should be taken when closing a form from an event that is raised from one of the form's own child controls. In this case it is always better to set the CloseAsync flag in the CLOSE method so that the current event has chance to finish executing before the form is destroyed.

Applications needing to execute a close operation programmatically should use the WINDOW\_CLOSE method rather than using the Send\_Event or Post\_Event stored procedures to invoke a CLOSE event directly (as was the case in previous versions of OpenInsight).

### Example

```
Function CLOSE( CtrlEntID, CtrlClassID, CancelFlag, CloseFlags )

  // Example - CLOSE event script - Let the system handler execute via a call
  // to forward_event, and cleanup resources if the form was closed.
  $Insert RTI_SSP_Equates

  Call Set_EventStatus( STESTAT_OK$ )
  Call Forward_Event( CancelFlag, CloseFlags )
  If Get_EventStatus() Then
    // Form wasn't closed
    Return FALSE$
  End

  // Final check - just in case someone forgot to set the EventStatus
  If Get_Property( CtrlEntID, "HANDLE" ) Else
    // Form wasn't closed
    Return FALSE$
  End

  // Now be very careful, as the form no longer exists, so try not to
  // reference it from this point onwards (especially any synthetic
  // properties)

  GoSub CleanAllTheThings

  // Return FALSE$ to stop the event chain as we've already forwarded to the
  // system CLOSE event handler above.

Return FALSE$
```

**See Also**

WINDOW SAVEWARN property, WINDOW CLOSE method, WINDOW HIDE method, End\_Window stored procedure.

## CREATE event

### Description

Occurs when a form is created.

### Syntax

```
bForward = CREATE( CtrlEntID, CtrlClassID, CreateParam )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>CreateParam</b>	Data passed from the method that created the form.

### 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 CREATE.

### Example

```
Function CREATE( CtrlEntID, CtrlClassID, CreateParam )  
  
    // Example CREATE event. Assume this is a non-visible data bound form and we have  
    // been passed the key to Load in the CreateParam argument  
  
    CloseForm = FALSE$  
    If BLen( CreateParam ) Then  
        EvStatus = Exec_Method( @Window, "READROW", CreateParam )  
        If BLen( EvStatus ) Then  
            CloseForm = TRUE$ ; // Read Failed - close the form  
        End  
    End  
  
    If CloseForm Then  
        Call Exec_Method( @Window, "CLOSE", TRUE$, TRUE$ ) ; // Async  
    End Else  
        Call Exec_Method( @Window, "SHOW" ); // Loaded OK - show the form  
    End  
  
    Return Not( CloseForm )
```

### **See Also**

SYSTEM SHOWDIALOG method, SYSTEM STARTFORM method, WINDOW SHOWDIALOG method, WINDOW STARTFORM method, WINDOW STARTMDICHILDFORM method, Dialog\_Box stored procedure, Start\_MDChild stored procedure, Start\_Window stored procedure.



## DBLCLK event

### Description

Occurs when the user double-clicks the mouse button on a form.

### Syntax

```
bForward = DBLCLK( CtrlEntID, CtrlClassID, CtrlKey, ShiftKey, MouseButton )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW" ).
<b>CtrlKey</b>	TRUE\$ if the Ctrl key was pressed down when the double-click was triggered, FALSE\$ otherwise.
<b>ShiftKey</b>	TRUE\$ if the Shift key was pressed down when the double-click was triggered, FALSE\$ otherwise.
<b>MouseButton</b>	Integer specifying which mouse button was double-clicked:  "0" - Left button or middle button "1" - Right 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 DBLCLK.

## Example

```
Function DBLCLK( CtrlEntID, CtrlClassID, CtrlKey, ShiftKey, MouseButton )  
  
    // Example DBLCLK event. If the Ctrl key is down then Load a non-modal, owned  
    // child form  
  
    If CtrlKey Then  
        Call Exec_Method( @Window, "STARTFORM", "MY_INFO_FORM" )  
    End  
  
Return TRUE$
```

## See Also

Common GUI BUTTONDOWN event, Common GUI BUTTONUP Event, Common GUI LOSTCAPTURE event, Common GUI MOUSEMOVE event.

## DELETE event

### Description

Occurs when the data row loaded into a data-bound form is deleted from the database.

### Syntax

```
bForward = DELETE( CtrlEntID, CtrlClassID, bSuppressWarning )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>bSuppressWarning</b>	This is a boolean value. If TRUE\$ then the delete operation should be unconditional, otherwise, the user will be warned about the delete and given the opportunity to cancel.

### 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 DELETE event has a system-level handler that performs the following tasks:

- Checks to see if all loaded data rows are locked. If not, the user is warned and the delete operation is aborted.
- If the bSuppressWarning parameter is FALSE\$ the form displays a message warning the user that data is about to be deleted, giving them an opportunity to cancel the process.
- The data row is deleted from the database.
- The SAVEWARN property is reset to FALSE\$.
- If the row was deleted while browsing a QBF result list, the list is updated and the next item in the list loaded, otherwise a CLEAR event is raised to reset the form's data-bound controls to their default state.

A DELETE event is never triggered by the system – it is only triggered by user action or developer code.

Applications needing to execute a delete operation programmatically should use the WINDOW DELETEROW method rather than using the Send\_Event stored procedure to invoke a DELETE event directly (as was the case in previous versions of OpenInsight).

### Example

```
Function DELETE( CtrlEntID, CtrlClassID, bSuppressWarning )

    // Example - DELETE event script that checks to see if the user has rights to
    // perform the delete
    $Insert RTI_SSP_Equates
    $Insert EvErrors

    IsOK = TRUE$
    GoSub IsTheUserOKToDeleteRows ; // whatever
    If IsOK Else
        // Failed the check - warn the user and stop here
        Call Exec_Method( @Window, "SHOWMESSAGE", "What do you think you are doing Dave?" )
        Call Set_EventStatus( SETSTAT_ERR$, EV_VALIDERR$ )
        Return FALSE$
    End

    // Assume single table form.
    TableID = Get_Property( @Window, "TABLE" )
    RowID = Get_Property( @Window, "ID" )

    // Now let the system event handler perform the DELETE in case we have a problem
    Call Set_EventStatus( SETSTAT_OK$ )
    Call Forward_Event( bSuppressWarning )
    If Get_EventStatus() Then
        // Assume error or cancelled
        Null
    End Else
        // Update the audit log...
        Call Update_Some_Audit Log( @UserName : " deleted" : TableID : " " : RowID )
    End

    // Return FALSE$ to stop the event chain as we've already forwarded to the
    // system DELETE event handler above.

Return FALSE$
```

### See Also

WINDOW SAVEWARN property, WINDOW CLEARROW method, WINDOW DELETEROW method, WINDOW CLEAR event, WINDOW SYSMMSG event.

## DROPDOWNMENU event

### Description

Occurs when a top-level item on a form's menu bar is selected and gives a chance for the application to modify the menu before it is displayed.

### Syntax

```
bForward = DROPDOWNMENU( CtrlEntID, CtrlClassID, menuID, menuStruct )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form object receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>MenuID</b>	Contains the fully qualified name of the top-level menu item to "drop-down".
<b>MenuStruct</b>	A dynamic array containing the executable structure of the menu.  Note that this structure does <i>not</i> include the usual menu header fields.

### 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 DROPDOWNMENU performs the following tasks:

- The DROPDOWNMENU quick event handler is executed (if defined) and the event status checked – if it returns anything other than FALSE\$ the operation is cancelled.
- If the menu item being selected is the "Window" item on an MDI Frame form then the details of the first ten MDI child forms are appended to the MenuStruct array – this is so they may be activated from the dropdown menu. If there are more than ten then a "More..." item is also appended which will trigger an "MDISELECT" event when selected.
- The dropdown menu is displayed via the TRACKDROPDOWNMENU method.

## Example

```
Function DROPDOWNMENU( CtrlEntID, CtrlClassID, menuID, menuStruct )

    // Example DROPDOWNMENU event code - check to see which clipboard items should
    // be enabled if this is the "Edit" menu item
    $Insert PS_Menu_Equates
    $insert PS_Equates

    Begin Case
        Case ( MenuID == @Window : ".MENU.EDIT" )
            // Get the current focus and it's EditState
            FocusID = Get_Property( "SYSTEM", "FOCUS" )
            EditState = Get_Property( FocusID, "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

        End Case

    Return TRUE$
```

## See Also

WINDOW TRACKDROPDOWNMENU method, Common GUI CONTEXTMENU event, WINDOW MDISELECT event.

## ENDDIALOG event

### Description

Occurs when an asynchronous dialog box returns a value to its owner form.

### Syntax

```
bForward = ENDDIALOG( CtrlEntID, CtrlClassID, DialogID, DialogValue, AsyncID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form object receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>DialogID</b>	Name of the dialog box returning the value to its owner.
<b>DialogValue</b>	Value returned from the dialog box to its owner.
<b>AsyncID</b>	ID passed to the dialog box by its owner when it was created.

### 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 ENDDIALOG.

### Example

```
Function ENDDIALOG( CtrlEntID, CtrlClassID, DialogID, DialogValue, AsyncID )  
  
    // Example ENDDIALOG event script - Look for a return value from the  
    // GET_CLIENT_DATA dialog box and assume it was passed the name of  
    // the control to return the data to:  
  
    Begin Case  
        Case ( DialogID == "GET_CLIENT_DATA" )  
            Call Set_Property_Only( AsyncID, "DEFPROP", DialogValue )  
        End Case  
  
    Return TRUE$
```

### **See Also**

WINDOW SHOWDIALOG method, WINDOW SHOWPOPUP method, Dialog\_Box stored procedure, End\_Dialog stored procedure.



## FORMSTATECHANGED event

### Description

Occurs when the "state" of a form changes. The term state refers to attributes that can affect the visual state of the form and the enabled state of its controls, such as being activated, reading data, clearing data etc. These actions usually require common UI menu items and buttons to be updated, for example Save, Clear and Delete menu items and buttons that perform the same actions.

This event is raised from the following system event handlers:

- ACTIVATED (for MDI Child forms)
- CLEAR, READ and WRITE events
- CLOSE (for MDI Child forms)
- QBF events

### Syntax

```
bForward = FORMSTATECHANGED( CtrlEntID, CtrlClassID, EventSource, FormState )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>EventSource</b>	Name of the event that triggered the form state change.
<b>FormState</b>	An @Fm-delimited array of boolean flags that denote the proposed enabled state of various common menu and toolbuttons:  <1> Save <2> SaveAs <3> Clear <4> Delete <5> LoadPrevRow <6> Print <7> PrintPreview <8> Close <20> QBFFinit <21> QBFRun <22> QBFFirst <23> QBFPprev <24> QBFFNext <25> QBFLast <26> QBFFPosition <27> QBFFTable <28> QBFFLoadList <29> QBFFGetQuery <30> QBFFClose

## 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 is only raised if the form's ALLOWFORMSTATEEVENTS property is TRUE\$.

The system-level event handler for FORMSTATECHANGED performs the following tasks:

- Builds the default FormState array based on the state of the row data loaded into the form.
- Executes the FORMSTATECHANGED quick event handler (if defined) and checks the event status – if it returns anything other than FALSE\$ the operation is cancelled.
- Sets the ENABLED state for a default set of items based on the flags in the FormState array parameter.
- If the form is an MDI Child then an MDICHILDSTATECHANGED event is raised for the parent MDI frame.

The default list of items affected by the form state is:

- MENU.FILE.SAVE
- MENU.FILE.SAVE\_F9
- MENU.FILE.SAVE\_ROW
- MENU.FILE.SAVE\_AS
- MENU.FILE.CLEAR
- MENU.FILE.CLEAR\_ROW
- MENU.FILE.DELETE
- MENU.FILE.DELETE\_ROW
- MENU.FILE.PRINT
- MENU.FILE.PRINT\_PREVIEW
- MENU.FILE.CLOSE
- MENU.EDIT.LOAD\_PREVIOUS\_DATA
- MENU.EDIT.LOAD\_PREVIOUS\_ROW
- MENU.QBF.INITIALIZE
- MENU.QBF.EXECUTE
- MENU.QBF.LOAD\_FROM\_EXTERNAL\_LIST
- MENU.QBF.LOADLIST
- MENU.QBF.GETQUERY
- MENU.QBF.NEXT
- MENU.QBF.PREVIOUS
- MENU.QBF.FIRST
- MENU.QBF.LAST
- MENU.QBF.ABSOLUTE
- MENU.QBF.GOTO\_ID
- MENU.QBF.TABLE
- MENU.QBF.CLOSE

Equates for use with the FORMSTATECHANGED event can be found in the RTI\_FORMSTATE\_EQUATES insert record.

## Example

```
Function FORMSTATECHANGED( CtrlEntID, CtrlClassID, EventSource, FormState )

    // Example - Set the state of some buttons based on the state of the form, and
    //             set the Delete flag to FALSE$ if the user is not an administrator
    //             (This assumes that this code runs _before_ the system event
    //             handler!)

    $Insert RTI_FormState_Equates
    $Insert Logical

    // Check the user
    If ( @Admin == 0 ) Then
        // Not an Admin - no deleting
        FormState<FormState<FSTATE_POS_DELETE$> = FALSE$
    End

    ObjxArray = @Window : ".BTN_SAVE"
    PropArray = "ENABLED"
    dataArray = FormState<FSTATE_POS_SAVE$>

    ObjxArray := @Rm : @Window : ".BTN_CLEAR"
    PropArray := @Rm : "ENABLED"
    dataArray := @Rm : FormState<FSTATE_POS_CLEAR$>

    ObjxArray := @Rm : @Window : ".BTN_DELETE"
    PropArray := @Rm : "ENABLED"
    dataArray := @Rm : FormState<FSTATE_POS_DELETE$>

    Call Set_Property_Only( ObjxArray, PropArray, dataArray )

Return TRUE$
```

## See Also

WINDOW ALLOWFORMSTATEEVENTS property, WINDOW ACTIVATED event, WINDOW CLOSE event, WINDOW CLEAR event, WINDOW MDICHILDSTATECHANGED event, WINDOW QBFABS event, WINDOW QBFCLOSE event, WINDOW QBFINIT event, WINDOW QBFFIRST event, WINDOW QBFLAST event, WINDOW QBFLLOADID event, WINDOW QBFNEXT event, WINDOW QBFPREV event, WINDOW QBFQUERY event, WINDOW READ event, WINDOW WRITE event.

## INACTIVATED event

### Description

Occurs when a form becomes inactive.

### Syntax

```
bForward = INACTIVATED( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 INACTIVATED.

For more information on this event please refer to the Windows documentation regarding WM\_ACTIVATE window message on the Microsoft website.

### Example

```
Function INACTIVATED( CtrlEntID, CtrlClassID )  
  
    // Example - INACTIVATED event for an MDI child form to let it parent frame  
    // know that is has lost the active status so it may set its menu items correctly  
  
    MDIFrame = Get_Property( @Window, "MDIFRAME" )  
  
    // Assume that the frame uses an OMNIEVENT handler to respond to the  
    // child being deactivated with a message of "CHILD_INACTIVATED" and the  
    // child ID as the first parameter  
  
    EvStatus = Exec_Method( MDIFrame, "SENDEVENT", "OMNIEVENT", |  
                           "CHILD_INACTIVATED", @Window )  
  
Return TRUE$
```

### **See Also**

Common GUI FOCUS property, SYSTEM FOCUS property, WINDOW ACTIVE property, WINDOW FOCUS property, WINDOW ACTIVATED event.

## IXLOOKUP event

### Description

Occurs when a form shows an Index Lookup dialog box.

### Syntax

```
bForward = INDEXLOOKUP( CtrlEntID, CtrlClassID, IndexedTable, SearchColumns,  
                        DisplayColumns, SelMode )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>IndexedTable</b>	The indexed table to search. This table must have at least one Btree index on it.
<b>SearchColumns</b>	Contains an @Fm-delimited list of column names to display in the dialog that the user may search.
<b>DisplayColumns</b>	Contains an @Fm-delimited list of columns names to show in the results popup when a user chooses the row(s) to load into the form.
<b>SelMode</b>	Contains an @fm delimited list of selection options:  <1> Mode - this can be either "MULTI" or "SINGLE" (the default). If "MULTI" then the user may select more than one row to return.  <2> Contains the name of the control to return the selected row IDs to. If null then the row IDs are loaded into the form as a QBF result list.  This can be virtual name like "@WINDOW" or "@SELF" that are used in normal quick event programming.  <3> If a control name is specified in field <2> this field contains the name of the property to set such as "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 IXLOOKUP event handler performs the following tasks:

- Parses and verifies the parameters passed to the event.
- Constructs a dialog box to display the search columns.
- Constructs a results popup to allow the user to select one or more rows from the filtered search data.
- Returns the selected rows to the specified control or the form's QBF result list.

### Example

N/a.

### See Also

WINDOW QBFLIST property, WINDOW SHOWINDEXLOOKUP method.

## MDICHILDSTATECHANGED event

### Description

Occurs for an MDI Frame form when the "state" of an MDI Child form changes. The term state refers to attributes that can affect the visual state of the form and the enabled state of its controls, such as being activated, reading data, clearing data etc. These actions usually require common UI menu items and buttons to be updated, for example Save, Clear and Delete menu items and buttons that perform the same actions.

This event is raised from an MDI Child form's FORMSTATECHANGED event.

### Syntax

```
bForward = MDICHILDSTATECHANGED( CtrlEntID, CtrlClassID, EventSource, |  
                                   FormState )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the MDI Frame form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>EventSource</b>	Name of the event that triggered the form state change.
<b>FormState</b>	An @Fm-delimited array of boolean flags that denote the proposed enabled state of various common menu and toolbuttons – this is set by the originating FORMSTATECHANGED event:  <1> Save <2> SaveAs <3> Clear <4> Delete <5> LoadPrevRow <6> Print <7> PrintPreview <8> Close <20> QBFFinit <21> QBFRun <22> QBFFirst <23> QBFPprev <24> QBFFNext <25> QBFFLast <26> QBFFPosition <27> QBFFTable <28> QBFFLoadList <29> QBFFGetQuery <30> QBFFClose



## 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 is only raised if the MDI Child form's ALLOWFORMSTATEEVENTS property is TRUE\$.

The system-level event handler for MDICHILDSTATECHANGED performs the following tasks:

- If the EventSource is a CLOSE event and the last MDI Child form is being closed the flags in the FormState array are all set to FALSE\$
- Executes the MDICHILDSTATECHANGED quick event handler (if defined) and checks the event status – if it returns anything other than FALSE\$ the operation is cancelled.
- Sets the ENABLED state for a default set of items based on the flags in the FormState array parameter.

The list of items affected by the default form state handler is:

- MENU.FILE.SAVE
- MENU.FILE.SAVE\_F9
- MENU.FILE.SAVE\_ROW
- MENU.FILE.SAVE\_AS
- MENU.FILE.CLEAR
- MENU.FILE.CLEAR\_ROW
- MENU.FILE.DELETE
- MENU.FILE.DELETE\_ROW
- MENU.FILE.PRINT
- MENU.FILE.PRINT\_PREVIEW
- MENU.FILE.CLOSE
- MENU.EDIT.LOAD\_PREVIOUS\_DATA
- MENU.EDIT.LOAD\_PREVIOUS\_ROW
- MENU.QBF.INITIALIZE
- MENU.QBF.EXECUTE
- MENU.QBF.LOAD\_FROM\_EXTERNAL\_LIST
- MENU.QBF.LOADLIST
- MENU.QBF.GETQUERY
- MENU.QBF.NEXT
- MENU.QBF.PREVIOUS
- MENU.QBF.FIRST
- MENU.QBF.LAST
- MENU.QBF.ABSOLUTE
- MENU.QBF.GOTO\_ID
- MENU.QBF.TABLE
- MENU.QBF.CLOSE

Equates for use with the MDICHILDSTATECHANGED event can be found in the RTI\_FORMSTATE\_EQUATES insert record.

## Example

```
Function MDICHILDSTATECHANGED( CtrlEntID, CtrlClassID, EventSource, FormState )

    // Example - Set the state of some buttons based on the state of the child form

    $Insert RTI_FormState_Equates
    $Insert Logical

    ObjxArray =         @Window : ".BTN_SAVE"
    PropArray =         "ENABLED"
    dataArray =         FormState<FSTATE_POS_SAVE$>

    ObjxArray := @Rm : @Window : ".BTN_CLEAR"
    PropArray := @Rm : "ENABLED"
    dataArray := @Rm : FormState<FSTATE_POS_CLEAR$>

    ObjxArray := @Rm : @Window : ".BTN_DELETE"
    PropArray := @Rm : "ENABLED"
    dataArray := @Rm : FormState<FSTATE_POS_DELETE$>

    Call Set_Property_Only( ObjxArray, PropArray, dataArray )

    Return TRUE$
```

## See Also

Common GUI ENABLED property, WINDOW ALLOWFORMSTATEEVENTS property, WINDOW MDIACTIVE property, WINDOW MDIFRAME property, WINDOW ACTIVATED event, WINDOW CLOSE event, WINDOW FORMSTATECHANGED event.

## MDISELECT event

### Description

Occurs when the user selects the "More Windows..." item from an MDI Frame form's "Windows" menu.

### Syntax

```
bForward = MDISELECT( CtrlEntID, CtrlClassID, activeID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>ActiveID</b>	Contains either: <ul style="list-style-type: none"><li>• The name of the MDI Child form to activate, or</li><li>• "-1" (the default) to show the RTI_MDISELECT dialog box – this allows the user to select an MDI Child form to activate.</li></ul>

### 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 for MDISELECT performs the following tasks:

- The MDISELECT quick event handler is executed (if defined) and the event status checked – if it returns anything other than FALSE\$ the event is cancelled.
- If ActiveID contains the name of a valid MDI child form it is activated, otherwise the the RTI\_MDISELECT dialog box is displayed allow a user to activate an MDI child form.
- Returns FALSE\$ because the event has been forwarded when checking the quick event handler as above.

### Example

N/a.

**See Also**

WINDOW MDIACTIVE property, WINDOW MDIFRAME property, WINDOW ACTIVATED event, WINDOW INACTIVATED event.

## PAGE event

### Description

Pseudo-method used to change the current page of a form.

### Syntax

```
bForward = PAGE( CtrlEntID, CtrlClassID, PageAction )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>PageAction</b>	May be one of the following values: <ul style="list-style-type: none"><li>• An integer that specifies the page to display. If this is greater than the page count then the last page is displayed, of less than 1 then the first page is displayed.</li><li>• "+" means go to the next page, and do not wrap around to the first page if on the last page.</li><li>• "++" means go to the next page and wrap around to the first page if on the last page.</li><li>• "-" means go to the previous page and do not wrap around to the last page if on the first page.</li><li>• "--" means go to the previous page and wrap around to the last page if on the first page.</li><li>• "L" means go to the last page.</li></ul>

### 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 is not really an event as such as it is never triggered in response to some other action – rather it is used as a method to change the current page on a form.

The system-level PAGE event handler uses the PageAction parameter to set the form's CURRENTPAGE property.

This event is considered deprecated in favor of the SETPAGE method.

**Example**

N/a.

**See Also**

Container API CURRENT page property, Container API PAGECOUNT property, Container API SETPAGE method.

## QBFABS event

### Description

Occurs when the form loads a row in a QBF result list using a position index.

### Syntax

```
bForward = QBFABS( CtrlEntID, CtrlClassID, AbsPos )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>AbsPos</b>	If specified this contains the index of the row to display (i.e. the "absolute" position).  If may also be null to allow the user to enter the index via a message box.

### 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 is triggered by setting the QBFPOS property.

The system level event for QBFABS performs the following tasks:

- If there is only one row in the QBF result list a message is displayed and the QBFABS operation is cancelled.
- If AbsPos is null then a message is displayed to allow the user to enter the index of the data row to display. If an invalid index is entered the user is warned and the QBFABS operation is cancelled.
- The form's SAVEWARN property is checked and the user warned of any unsaved changes with the option to cancel the QBFABS operation, ignore the changes, or save the data before continuing.
- If the QBFREADMODE property is OnlyRead then the READROW method is used to load the data row into the form, otherwise the custom QBF form load process is used instead. If QBFREADMODE is OBFThenRead a READ event is then triggered.
- The index into the QBF result list (QBFPOS property) is updated.

- The form's caption is updated to show the current index in the QBF result list.
- If displayed, the QBFTABLE result list dialog box is synchronized with the new index.
- The form's SAVEWARN property is set to FALSE\$.

### **Example**

N/a.

### **See Also**

WINDOW QBFPOS property, WINDOW QBFREADMODE property, WINDOW QBFSHOWFIRST method, QBFSHOWLAST method, WINDOW QBFSHOWNEXT method, WINDOW QBFSHOWPREV method.



## QBFCLOSE event

### Description

Occurs when a form's QBF session is closed.

### Syntax

```
bForward = QBFCLOSE( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 is triggered from the QBFCLOSESESSION method.

The system level event for QBFCLOSE performs the following tasks:

- Checks the form's SAVEWARN property and warns the user of any unsaved changes. If so, then the user is warned and given the option to cancel the QBFCLOSE operation, ignore the changes, or save the data before continuing.
- Clears the data-bound controls in the form and unlocks the row if needed (Note – this is via an internal method and not via a CLEARROW method, i.e. the CLEAR event is not triggered).
- The QBFBSTATUS is set to QBFINACTIVE, and the QBFPOS and QBFLIST properties are cleared,
- Enabled/read-only controls that were enabled when the QBF session was started are disabled again if necessary.
- The form's caption is reset.
- A FORMSTATECHANGED event is raised (with a "QBFCLOSE" EventSource parameter).

### **Example**

N/a.

### **See Also**

WINDOW QBFSTATUS property, WINDOW QBFINITSESSION method, WINDOW QBFCLSESESSION method, WINDOW FORMSTATECHANGED event, WINDOW QBFINIT event.

## QBFFIRST event

### Description

Occurs when the form loads the first row in a QBF result list.

### Syntax

```
bForward = QBFFIRST( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 is triggered from the QBFSHOWFIRST method.

The system level event for QBFFIRST performs the following tasks:

- The form's SAVEWARN property is checked and the user warned of any unsaved changes with the option to cancel the QBFABS operation, ignore the changes, or save the data before continuing.
- If the QBFREADMODE property is OnlyRead then the READROW method is used to load the first data row from the QBF results list into the form, otherwise the custom QBF form load process is used instead. If QBFREADMODE is OBFThenRead a READ event is then triggered.
- The index into the QBF result list (QBFPOS property) is updated.
- The form's caption is updated to show the current index in the QBF result list.
- If displayed, the QBFTABLE result list dialog box is synchronized with the new index.
- The form's SAVEWARN property is set to FALSE\$.

### Example

N/a.

**See Also**

WINDOW QBFPOS property, WINDOW QBFREADMODE property, WINDOW QBFSHOWLAST method, WINDOW QBFSHOWNEXT method, WINDOW QBFSHOWPREV method.

## QBFINIT event

### Description

Occurs when QBF session is initialized for a form.

### Syntax

```
bForward = QBFINIT( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 is triggered from the QBFINITSESSION method.

The system level event for QBFINIT performs the following tasks:

- Checks the form's SAVEWARN property and warns the user of any unsaved changes. If so, then the user is warned and given the option to cancel the QBFINIT operation, ignore the changes, or save the data before continuing.
- Updates the form's caption to show that the QBF session is active.
- The QBSTATUS is set to QBFINitalize (Note that LOSTFOCUS data validation and conversion on controls will not be performed while the QBSTATUS is QBFINitalize).
- Clears the data-bound controls in the form (Note: this is via an internal method and not via a CLEARROW method, i.e. the CLEAR event is not triggered).
- Enables all non-primary disabled/read-only data-bound controls and read-only/protected data-bound Edit Table columns

## Example

```
Function QBFINIT( CtrlEntID, CtrlClassID )

    // Example: allow the system QBFINIT to execute, and if successful raise an
    // OMNIEVENT event on the parent MDI frame form to let it know the state of
    // the child.

    Call Set_EventStatus( SETSTAT_OK$ )
    Call Forward_Event()
    If Get_EventStatus() Then
        // Assume cancelled or error
        Return FALSE$
    End

    MDIFrame = Get_Property( @Window, "MDIFRAME" )

    // Assume that the frame uses an OMNIEVENT handler to respond to the
    // child being initialized with a message of "CHILD_QBFINIT" and the
    // child ID as the first parameter

    EvStatus = Exec_Method( MDIFrame, "SENDEVENT", "OMNIEVENT", |
        "CHILD_QBFINIT", CtrlEntID )

    // Return FALSE$ to stop the event chain as we've already forwarded to the
    // system QBFINIT event handler above.

Return FALSE$
```

## See Also

Common GUI CONV property, Common GUI VALID property, WINDOW QBFSTATUS property, WINDOW QBFCLOSESESSION method, WINDOW QBFINITSESSION method, WINDOW QBFRUNQUERY method, Common GUI LOSTFOCUS event, WINDOW QBFCLOSE event, WINDOW QBFRUN event.

## QBFLAST event

### Description

Occurs when the form loads the last row in a QBF result list.

### Syntax

```
bForward = QBFLAST( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 is triggered from the QBFSHOWLAST method.

The system level event for QBFLAST performs the following tasks:

- The form's SAVEWARN property is checked and the user warned of any unsaved changes with the option to cancel the QBFABS operation, ignore the changes, or save the data before continuing.
- If the QBFREADMODE property is OnlyRead then the READROW method is used to load the last data row from the QBF results list into the form, otherwise the custom QBF form load process is used instead. If QBFREADMODE is OBFThenRead a READ event is then triggered.
- The index into the QBF result list (QBFPOS property) is updated.
- The form's caption is updated to show the current index in the QBF result list.
- If displayed, the QBFTABLE result list dialog box is synchronized with the new index.
- The form's SAVEWARN property is set to FALSE\$.

### Example

N/a.

### **See Also**

WINDOW QBFPOS property, WINDOW QBFREADMODE property, WINDOW QBFSHOWFIRST method, WINDOW QBFSHOWNEXT method, WINDOW QBFSHOWPREV method.



## QBFLOADID event

### Description

Occurs when the form loads a row in a QBF result list using a row ID.

### Syntax

```
bForward = QBFLOADID( CtrlEntID, CtrlClassID, RowID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>RowID</b>	If specified this contains the ID of the row to display. It must be in the QBF result list.  It may also be null to allow the user to enter the ID via a message box.

### 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 is triggered by the QBFGOTOID method.

The system level event for QBFLOADID performs the following tasks:

- If RowID is null then a message is displayed to allow the user to enter the ID of the data row to display. If an invalid ID is entered the user is warned and the QBFLOADID operation is cancelled.
- The form's SAVEWARN property is checked and the user warned of any unsaved changes with the option to cancel the QBFLOADID operation, ignore the changes, or save the data before continuing.
- If the QBFREADMODE property is OnlyRead then the READROW method is used to load the data row into the form, otherwise the custom QBF form load process is used instead. If QBFREADMODE is OBFTThenRead a READ event is then triggered.
- The index into the QBF result list (QBFPOS property) is updated.
- The form's caption is updated to show the current index in the QBF result list.

- If displayed, the QBFTABLE result list dialog box is synchronized with the new index.
- The form's SAVEWARN property is set to FALSE\$.

### **Example**

N/a.

### **See Also**

WINDOW QBFLIST property, WINDOW QBFPOS property, WINDOW QBFREADMODE property, WINDOW QNFGOTOID method, WINDOW QBFSHOWFIRST method, QBFSHOWLAST method, WINDOW QBFSHOWNEXT method, WINDOW QBFSHOWPREV method.

## QBFLOADLIST event

### Description

Occurs when the form needs to obtain the name of a saved list of keys to load into its QBF result list.

### Syntax

```
bForward = QBFLOADLIST( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 is triggered from the QBFLOADSAVEDLIST method.

The system level event for QBFLOADLIST performs the following tasks:

- Displays a dialog allowing the user to choose the source of the list. This may be an entry in the TCL Query Table, or the name of a saved list in the SYSLISTS table.
- Once the source has been determined the keys are extracted and loaded into the form's QBFLIST property.

### Example

N/a.

### See Also

WINDOW QBFLIST property, WINDOW QBFLOADSAVEDLIST method.

## QBFNEXT event

### Description

Occurs when the form loads the next row in a QBF result list.

### Syntax

```
bForward = QBFNEXT( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 is triggered from the QBFSHOWNEXT method.

The system level event for QBFNEXT performs the following tasks:

- The form's SAVEWARN property is checked and the user warned of any unsaved changes with the option to cancel the QBFABS operation, ignore the changes, or save the data before continuing.
- The index into the QBF result list is calculated for the next item (If it is already set to the last item it is set to show the first item instead).
- If the QBFREADMODE property is OnlyRead then the READROW method is used to load the specified data row from the QBF results list into the form, otherwise the custom QBF form load process is used instead. If QBFREADMODE is OBFTThenRead a READ event is then triggered.
- The index into the QBF result list (QBFPOS property) is updated.
- The form's caption is updated to show the current index in the QBF result list. If displayed, the QBFTABLE result list dialog box is synchronized with the new index.
- The form's SAVEWARN property is set to FALSE\$.

### **Example**

N/a.

### **See Also**

WINDOW QBFPOS property, WINDOW QBFREADMODE property, WINDOW QBFSHOWFIRST method, WINDOW QBFSHOWLAST method, WINDOW QBFSHOWPREV method.

## QBFPREV event

### Description

Occurs when the form loads the previous row in a QBF result list.

### Syntax

```
bForward = QBFPREV( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 is triggered from the QBFSHOWPREV method.

The system level event for QBFPREV performs the following tasks:

- The form's SAVEWARN property is checked and the user warned of any unsaved changes with the option to cancel the QBFABS operation, ignore the changes, or save the data before continuing.
- The index into the QBF result list is calculated for the previous item (If it is already set to the first item it is set to show the last item instead).
- If the QBFREADMODE property is OnlyRead then the READROW method is used to load the specified data row from the QBF results list into the form, otherwise the custom QBF form load process is used instead. If QBFREADMODE is OBFTThenRead a READ event is then triggered.
- The index into the QBF result list (QBFPOS property) is updated.
- The form's caption is updated to show the current index in the QBF result list.
- If displayed, the QBFTABLE result list dialog box is synchronized with the new index.
- The form's SAVEWARN property is set to FALSE\$.

### **Example**

N/a.

### **See Also**

WINDOW QBFPOS property, WINDOW QBFREADMODE property, WINDOW QBFSHOWFIRST method, WINDOW QBFSHOWLAST method, WINDOW QBFSHOWNEXT method.

## QBFQUERY event

### Description

Occurs when the user enters a "raw" RLIST SELECT statement to load a QBF result list.

### Syntax

```
bForward = QBFQUERY( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 is triggered from the QBFASKQUERY method.

The system level event for QBFQUERY performs the following tasks:

- Checks to see if the QBFSTATUS is QBFInitialize, or QBFActive – if so a QBFCLOSESESSION method is executed to return the form to a clean state, otherwise a CLEARROW method is executed instead.
- This user is prompted to enter the RLIST SELECT statement they wish to execute.
- If they do so the form's caption is updated to indicate that a search is being processed.
- If the search returns one or more rows a QBFSHOWFIRST method is executed to display the first result, otherwise the user is informed that no matching rows were found.

### Example

N/a.



### **See Also**

WINDOW QBFSTATUS property, WINDOW CLEARROW method, WINDOW QBFCLOSESESSION method, WINDOW QBFSHOWFIRST method,

## QBFRUN event

### Description

Occurs when the form uses the query data entered into the data-bound controls to search the database and load a QBF result list.

### Syntax

```
bForward = QBFRUN( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 is triggered from the QBFRUNQUERY method.

The system level event for QBFRUN performs the following tasks:

- Checks to see if the QBFSTATUS is QBFInitialize. If not, the query operation is cancelled.
- Updates the form's caption to indicate a query is being processed.
- Checks each data-bound control on the form to see if contains data – if so, this is used to build an RLIST query string.
- If no query data has been entered the user is asked if they wish to select all rows for the QBF result list instead. If not the form's caption is reset to indicate that the QBFSTATUS is still QBFInitialize and the event is stopped.
- Otherwise, the query string is parsed and executed to search the database for matching rows.
- If the search was unsuccessful the form returns to a QBFSTATUS of QBFInitialize and the user can enter different search terms or close the QBF session.
- If the search is successful the resulting keys are loaded into a QBF result list, and the enabled/read-only controls are reset. The QBFSTATUS is set to QBFInactive and a QBFFIRST event is triggered to load the first row in the result list.

### **Example**

N/a.

### **See Also**

WINDOW QBFSTATUS property, WINDOW QBFINITSESSION method, WINDOW QBFCLOSESESSION method, WINDOW QBFRUNQUERY method, WINDOW QBFFIRST event.

## QBFTABLE event

### Description

Occurs when a form's QBF result list is about to be displayed in a non-modal dialog box, allowing the user an easy visual way of navigating it.

### Syntax

```
bForward = QBFTABLE( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 is triggered from the QBFSHOWTABLE method.

The system level event for QBFTABLE executes the non-modal OIWIN\_QBFLISTRESULTS dialog box that displays the rows in the QBF result list along with the data from the data-bound controls in the form.

As the user selects a row in the dialog the QBFPOS property of the form is updated to load the row into it.

If a row is deleted from the form, it is also deleted from the dialog. Likewise, if data is updated in the form it is updated in the dialog too.

### Example

N/a.

### See Also

WINDOW QBFLIST property, WINDOW QBFPOS property, WINDOW QBFSHOWTABLE method.

## READ event

### Description

Occurs when data is read from the database into a data-bound form.

### Syntax

```
bForward = READ( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 READ event has a system-level handler that performs the following tasks:

- Checks the form's SAVEWARN property and warns the user of any unsaved changes. If so, then the user is warned and given the option to cancel the read operation (This step normally happens if the READ event is invoked programmatically via the READROW method).
- Checks to see that all controls bound to a key part are filled. If not, the user is warned and the read operation is aborted.
- Locks and reads the primary data row (and secondary ones if this is a multi-table form), constructing a "row-map" of data and bound controls.
  - If any locks fail the user is given the option to cancel the read operation or continue in a "view-only mode". If cancelled the CLEARROW method is executed to reset the form's contents.
  - The form's caption is updated to reflect the view-only mode if appropriate.
- The data in the row-map is loaded into the data-bound controls.
- The NEWROW property is set appropriately.
- The row-map is cached for use with the READPREVROW method if the LOADPREVALWAYS property is TRUE\$.

A READ event will be triggered by the system in the following circumstances:

- From the LOSTFOCUS event of controls bound to key columns when they all contain data, and the key has been changed.
- From the READROW method.

Applications needing to execute a read operation programmatically should use the WINDOW READROW method rather than using the Send\_Event stored procedure to invoke a READ event directly (as was the case in previous versions of OpenInsight).

### Example

```
Function READ( CtrlEntID, CtrlClassID )

    // Example - READ event script that adds some information to STATIC control
    // called TXT_INFO that is not data-bound after the read operation
    $Insert RTI_SSP_Equates

    // First let the system event handler perform the READ in case we have a problem
    Call Set_EventStatus( SETSTAT_OK$ )
    Call Forward_Event()
    If Get_EventStatus() Then
        // Assume cancelled or error
        Null
    End Else

    KeyCode = Get_Property( @Window : ".EDL_KEYCODE, "TEXT" )
    ISNCode = Get_Property( @Window : ".EDL_ISNCODE, "TEXT" )

    InfoText = KeyCode : ":" : ISNCode

    Call Set_Property_Only( @Window : ".TXT_INFO", "TEXT", InfoText )
End

// Return FALSE$ to stop the event chain as we've already forwarded to the
// system READ event handler above.

Return FALSE$
```

### See Also

WINDOW LOADPREVALWAYS property, WINDOW SAVEWARN property, WINDOW CLEARROW method, WINDOW READPREVROW method, WINDOW READROW method, Common GUI LOSTFOCUS event, WINDOW SYSMMSG event.

## TILE event

### Description

Arranges all MDI Child forms into a tiled layout for an MDI frame form.

### Syntax

```
bForward = TILE( CtrlEntID, CtrlClassID, Orientation )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>Orientation</b>	Boolean value – TRUE\$ to tile vertically, of FALSE\$ to tile horizontally.

### 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 this event calls the MDITILE method to perform the tiling operation, and because of this it has been deprecated in favor of that method. It is implemented only for backwards compatibility with earlier versions of OpenInsight.

(Note that this event is not a "true" event as such as it is never triggered by the PS, it can only be "manually" triggered by the developer in an application (typically from an MDI Window menu item) – it is actually a method masquerading as an event).

Please see the documentation for the WM\_MDITILE message on the Microsoft website for more details.

### Example

N/a.

### See Also

WINDOW MDITILE method.

## SCALED event

### Description

This event is raised when the form's DPI or SCALEFACTOR property is changed. OpenInsight will automatically adjust the following properties for the form's child controls:

- Position and size coordinates.
- Fonts.
- Images.

The SCALED event provides the opportunity for a developer to apply any further modifications needed to handle the new scale.

### Syntax

```
bForward = SCALED( CtrlEntID, CtrlClassID, OrigDpiX, OrigDpiY, OrigScaleFactor, |  
                  NewDpiX, NewDpiY, NewScaleFactor )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").
<b>OrigDpiX</b>	X DPI value before the scale change took place.
<b>OrigDpiY</b>	Y DPI value before the scale change took place.
<b>OrigScaleFactor</b>	Custom scaling factor before the scale change took place.
<b>NewDpiX</b>	X DPI value after the scale change took place.
<b>NewDpiY</b>	Y DPI value after the scale change took place.
<b>NewScaleFactor</b>	Custom scaling factor after the scale change took place.

### 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 this event.



The Windows WM\_DPICHANGED message will trigger the SCALED event when the form's DPI is changed by moving it to a monitor with a different DPI. More information on WM\_DPICHANGED can be found on the Microsoft website.

### Example

```
Function SCALED( CtrlEntID, CtrlClassID, OrigDpiX, OrigDpiY, OrigScaleFactor, |
                NewDpiX, NewDpiY, NewScaleFactor )

    // Example - SCALED event to set the Zoom property of an imaginary OLE preview
    //           control based on the passed DPI and custom scalefactor.

    $Insert Logical

    // We are assuming that this imaginary "Zoom" property is an integer
    // value which expresses the zoom factor as a percentage, e.g.
    //
    // E.g.
    //
    // Zoom Factor   Property Value
    // =====
    //      50%      ->      50
    //     100%      ->     100
    //     125%      ->     125
    //
    // etc.
    //
    // We are going to combine the scaling factor of the DPI along with
    // the custom scale factor (if any).

    DpiFactor   = ( NewDpiX / 96 )
    TotalFactor = DpiFactor * NewScaleFactor

    NewZoom = Int( 100 * TotalFactor )

    Call Set_Property_Only( @Window : ".OLE_PREVIEW", "OLE.Zoom", NewZoom )

Return TRUE$
```

### See Also

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

## VISUALSTYLECHANGED event

### Description

Occurs when Window's visual styling changes, usually in response to the user changing colors in the Personalization Settings.

### Syntax

```
bForward = VISUALSTYLECHANGED( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form object receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 default event handler for the VISUALSTYLECHANGED event.

This event is raised for the following Windows messages:

- WM\_DWMCOLORIZATIONCOLORCHANGED
- WM\_SYSCOLORCHANGE

Further information on these messages may be found on the Microsoft web site.

### Example

N/a.

### See Also

SYSTEM DWMCOLORS property, SYSTEM SYSTEMFONTS property, SYSTEM THEMED property, SYSTEM ALPHACOLOR method, SYSTEM DARKENCOLOR method, SYSTEM LIGHTENCOLOR method, SYSTEM MIXCOLORS method.

## WRITE event

### Description

Occurs when data is written from a data-bound form to the database.

### Syntax

```
bForward = WRITE( CtrlEntID, CtrlClassID )
```

### Parameters

Name	Description
<b>CtrlEntID</b>	Fully qualified name of the form receiving the event.
<b>CtrlClassID</b>	Type of object receiving the event (always "WINDOW").

### 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 WRITE event has a system-level handler that performs the following tasks:

- Triggers a LOSTFOCUS event on the control with focus to ensure that any data it contains is properly validated. If there is a problem here the write operation is aborted.
- Checks to see that all controls bound to a key part are filled. If not, the user is warned and the write operation is aborted.
- Checks to see if all loaded data rows are locked. If not, the user is warned and the write operation is aborted.
- Data is extracted from the data-bound controls into a "row-map" structure. This is then used to update the data row columns as necessary, and each row is then written back to the database after all its columns have been updated.
  - Note that using the ATRECORD or ROW properties, or the UPDATEROW method may cause data not bound to controls to be written back here as well – see the individual entries for those members for more details.
- The SAVEWARN property is reset to FALSE\$.
- The NEWROW property is reset to FALSE\$
- The row-map is cached for use with the READPREVROW method and the ORIGROWVALUE property.
- The form's caption is reset to its default state.

- If the CLEARONWRITE property is TRUE\$ a CLEARROW method is executed to reset the form's data-bound controls to their default state.

A WRITE event will be triggered by the system in the following circumstances:

- From the LOSTFOCUS event when a new key has been entered and changes have also been made in the data-bound controls. The user is presented with a message which allows them to save the changes, which can then execute a write operation.

Applications needing to execute a write operation programmatically should use the WINDOW WRITEROW method rather than using the Send\_Event stored procedure to invoke a WRITE event directly (as was the case in previous versions of OpenInsight).

### Example

```
Function WRITE( CtrlEntID, CtrlClassID )

    // Example - WRITE event script that performs some preliminary validation checks
    // before allowing the WRITE event to proceed. If that's OK then update an audit
    // Log after the WRITE
    $Insert RTI_SSP_Equates
    $Insert EvErrors

    IsOK = TRUE$
    GoSub DoTheChecks
    If ISOK Else
        // Failed the checks - warn the user and stop here
        Call Exec_Method( @Window, "SHOWMESSAGE", "Err ... Computer says no" )
        Call Set_EventStatus( SETSTAT_ERR$, EV_VALIDERR$ )
        Return FALSE$
    End

    // Assume single-table form.
    TableID = Get_Property( @Window, "TABLE" )
    RowID   = Get_Property( @Window, "ID" )

    // Now let the system event handler perform the WRITE in case we have a problem
    Call Set_EventStatus( SETSTAT_OK$ )
    Call Forward_Event()
    If Get_EventStatus() Then
        // Assume error
        Null
    End Else
        // Update the audit Log...
        Call Update_Some_Audit Log( @UserName : " updated " : TableID : " " : RowID )
    End

    // Return FALSE$ to stop the event chain as we've already forwarded to the
    // system WRITE event handler above.

Return FALSE$
```

### See Also

Common GUI ORIGROWVAL property, WINDOW ATRECORD property, WINDOW ROW property, WINDOW SAVEWARN property, WINDOW CLEARROW method, WINDOW READPREVROW method, WINDOW WRITEROW method, WINDOW UPDATEROW method, Common GUI LOSTFOCUS event, WINDOW CLEAR event, WINDOW SYSMSG event.