SYSTEM Object

The SYSTEM object is an intrinsic object that represents the Presentation Server and OS environment and provides an interface to many operating system functions. It is automatically created at startup and is always available.

Developer Notes

1. In contrast to other intrinsic objects the SYSTEM object DESTROY method *may* be used on the SYSTEM object itself to programmatically close an application.
2. Many of the methods from the deprecated Utility stored procedure are now exposed as methods of the SYSTEM object.

SYSTEM Properties

The SYSTEM object supports the following properties:

Name	Description
ASYNCCURSORPOS	Retrieves the current cursor position in screen coordinates.
ASYNCKEYSTATE	Retrieves the state of a key at the time the property is accessed.
AUTOEXEC	Specifies if the application is running in "AutoExec" mode.
BLOCKEVENTS	Enabled or disables the Presentation Server from generating and dispatching events.
CHARMAP	Sets a character conversion map for ANSI applications.
CMDLINE	Returns the command line used to start the Presentation Server.
COMCTLVERSION	Returns the version of the Windows COMCTL32 DLL loaded by the system.
CONFIGFILE	Returns the name and path of the RXI file used at startup.
CONFIGINFO	Returns the settings extracted from the RXI file used at startup.
COUNTER	Returns and increments the system counter value.
CURRENTEVENT	Returns the name of the currently executing event.
CURSOR	Specifies the default cursor for the application.
CURSORPOS	Returns the position of the mouse cursor with respect to the last message processed.
CUSTOMCOLORS	Specifies the array of custom colors used with the Windows Color dialog box.
DELIMCOUNT	Specifies the "delimiter count" used by the application when converting ANSI strings to Unicode and vice-versa.
DEVMODE	Specifies if the Presentation Server is operating in development mode.
DEVSYSTEM	Specifies if the application is licensed for development capabilities.
DPI	Returns the System DPI setting.
DPIAWARE	Specifies if the Presentation Server is running in DPI-Aware mode.
DPIPERMONITOR	Specifies if the operating system supports monitors with individual DPI settings.
DRAGSOURCE	Returns the name of the control being dragged from during a Drag And Drop operation.
DROPTARGET	Returns the name of the control that is currently the target of a Drag and Drop operation.
DWMCOLORS	Returns the current color set used for the Windows DWM (Desktop Window Manager).
ENGINEHANDLE	Returns the handle of the RevEngine form.
ENVVARLIST	Returns a list containing of all the Windows environment variables and their values.

EVENTSTACK	Returns the Presentation Server EventStack.
EXITCODE	Returns the exit code of the last process executed via the RUNWIN method.
FOCUS	Returns the ID of the control with focus, or sets the focus to specific control.
FOCUSSTYLES	Specifies the global style information for edit-type controls.
FONTLIST	Returns a list of available fonts.
GUITHREADINFO	Returns GUI information for the active thread.
IDLEPROC	Gets the first item in the "idle procedure" queue, or replaces the queue contents.
IDLEPROCQUEUE	Returns the list of "idle procedures" waiting to be executed.
INTERACTIVE	Specifies if the Presentation Server is running on an interactive window station.
KEYSTATE	Retrieves the state of a key or all keys.
LOCALE	Returns the default User and System locale names.
LOGININFO	Returns the credentials used to log into the application.
MESSAGEFONT	Returns the font used by Windows for displaying text in message boxes.
METRICS	Returns a specified Windows System Metric value.
MODAL	Disables or enables all Presentation Server forms.
MODULEFILENAME	Returns the path and file name of the current Presentation Server instance.
MONITORLIST	Returns a dynamic array of monitor information.
MOUSECAPTURED	Returns the ID and handle of the GUI object that is capturing the mouse.
PREVFOCUS	Returns the ID of the GUI object that previously had the focus.
PROCESSID	Returns the process identifier of the current Presentation Server instance.
QUERYEND	Specifies if the Presentation Server is processing a WM_QUERYENDSESSION message.
QUEUEDEVENTS	Returns a list of events queued for execution by the Presentation Server.
RECEIVER	Specifies the name of a GUI object that receives and displays data sent from a Basic+ Send_Dyn() call.
RUNMODE	Specifies if the Presentation Server is operating in runtime mode.
SERVERNAME	Returns the name of the Pipe or the TCPIP Address and Port used to communicate with the RevEngine.
SHOWACCELERATORS	Specifies if controls should display accelerator keys at all times or only when a window is launched via a keyboard action.
SHUTDOWN	Specifies if the Presentation Server is shutting down.
SIZE	Returns the dimension of primary monitor and its workspace.
STATUSFONT	Returns the font used by Windows for displaying text in status lines.

SUPPRESSAUTODESTROY	Specifies if the "auto-shutdown" process is enabled.
SYSTEMFONTS	Returns the fonts used by Windows for displaying text in various parts of the system.
TASKBARID	Returns the TaskBar ID for the current system process.
THEMED	Specifies if the system is running with OS visual styles enabled.
TIMEZONE	Returns current time zone information.
TYPES	Returns a list of all object types supported by the Presentation Server.
UTF8	Specifies if the Presentation Server is running in UTF8 or ANSI mode.
VERSION	Returns OS and Presentation Server version information.
VISIBLE	Specifies if the IDE toolset (RTI_IDE) is visible.
WIN64	Returns the PS and OS 64-bit flags.
WINCOUNT	Returns the number of currently running forms.
WINDOWGHOSTING	Specifies if "window-ghosting" is active for the current Presentation Server instance.

ASYNCCURSORPOS property

Description

Retrieves the *current* cursor position in screen coordinates.

If the name of a window or control is passed in the index argument the returned screen coordinates will be scaled to DIPs based on the DPI and scale factor of the passed object.

Property Value

This property is an @Fm-delimited array containing the cursor coordinates.

```
<1> X-Position
<2> Y-Position
```

Index Value

If set this should contain the name of a window or control to scale the coordinates too. The coordinates are then returned in DIPs.

Property Traits

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

Remarks

This property is essentially a wrapper around the GetCursorPos Windows API function, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Get the current screen position of the cursor in pixels

CursorPos = Get_Property( "SYSTEM", "ASYNCCURSORPOS" )

// Get the current screen position of the cursor in DIPs with
// respect to the current window

CursorPos = Get_Property( "SYSTEM", "ASYNCCURSORPOS", @Window )
```

See also

SYSTEM CURSORPOS property.

ASYNCKEYSTATE property

Description

Returns the state of a key at the time the property is accessed. The key to check is passed as a virtual-key code in the index parameter.

Property Value

This property is an integer value that represents a set of bit flags. As a general rule:

- If the key is pressed down the property returns a value < 0.
- If the key is not pressed 0 is returned.
- If the key is not pressed *but has been since the last ASYNCKEYSTATE call*, then 1 is returned.

See remarks for more information.

Property Traits

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

Remarks

This property is essentially a wrapper around the GetAsyncKeyState Windows API function, so please refer to the documentation on the Microsoft website for further information, especially regarding some caveats regarding the value returned.

Virtual-key code constants are defined in the MSWin_VirtualKey_Equates insert record.

Example

```
 // Get the state of the Shift key

$Insert MSWin_VirtualKey_Equates

ShiftDown = ( Get_Property( "SYSTEM", "ASYNCKEYSTATE", VK_SHIFT$ ) < 0 )
```

See also

SYSTEM KEYSTATE property.

AUTOEXEC property

Description

Specifies if the "/AE" Autoexec switch was set when the application was started.

Property Value

This property is a boolean value. If the PS was started with the "/AE" command line switch set to 1, or the "autoExec" setting in the RXI file set to 1, this property returns TRUE$, otherwise it returns FALSE$.

Property Traits

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

Remarks

N/a.

Example

```
IsAutoExec = Get_Property( "SYSTEM", "AUTOEXEC" )
```

See also

Starting the Presentation Server chapter.

BLOCKEVENTS property

Description

Enabled or disables the Presentation Server from generating and dispatching events (except for CLOSE events).

Property Value

This property is a boolean value. When set to TRUE$ no events will be raised by the Presentation Server *except* for any CLOSE events. When set to FALSE$ all events are generated as normal and dispatched to OpenEngine for processing.

Property Traits

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

Remarks

This property was originally called BLOCK_EVENTS in previous versions of OpenInsight. BLOCK_EVENTS is now an alias for this property to preserve backwards compatibility.

Example

```
$Insert Logical

// Prevent any events (like GOTFOCUS and LOSTFOCUS) from firing
// while we move the focus.

ObjxArray =         "SYSTEM"
PropArray =         "BLOCKEVENTS"
DataArray =         TRUE$

ObjxArray := @Rm : "SYSTEM"
PropArray := @Rm : "FOCUS"
DataArray := @Rm : @Window : ".EDL_SURNAME"

ObjxArray := @Rm : "SYSTEM"
PropArray := @Rm : "BLOCKEVENTS"
DataArray := @Rm : FALSE$

Call Set_Property_Only( ObjxArray, PropArray, DataArray )
```

See also

SYSTEM MODAL property.

CHARMAP property

Description

This property contains a conversion map to use when processing string values (not object and property names) passed to and from the Presentation Server. It swaps specified characters with other characters before storing them in the database, and converts them back before displaying them.

In ANSI applications this can be useful when using a language that supports characters that conflict with the system delimiters, such as the " ý" character (code 253). Under normal circumstances this would be treated as an @Vm delimiter (code 253) which might corrupt a structure or a multivalued array. Using CHARMAP it is possible to map this to an "unused" character like "§" (code 167) instead and preserve the integrity of the program when the value is fetched from the PS.

Property Value

This property is a 510-character string.

The first 255 characters are the "get-map". When set, the value from each character in a property value is compared to its position in the get-map. It is then swapped with the get-map character at that position during "Get_Property" type operations.

The latter 255 characters are the "set-map". When set, the value from each character in a property value is compared to its position in the set-map. It is then swapped with the set-map character at that position during "Set_Property" type operations.

Property Traits

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

Remarks

This property is considered to be deprecated and is supported for backwards compatibility only. New applications that require internationalization support should be designed as UTF8-aware applications instead.

Note that CHARMAP-converted strings stored in the database must be converted back if they are to be used with other tools outside of the Presentation Server. This includes data that is exported to files, or is accessed directly via methods that use RevCAPI, like web-applications and the EngineServer.

CHARMAP cannot be set for the SYSPROG application.

Example

```
Equ FIRST_UNUSED_CHAR$ To 129

Map    = Get_Property( "SYSTEM", "CHARMAP" )
MapGet = Map[1,255]
MapSet = Map[256,255]

// Let's map six system delimiters (@Stm, @Tm, @Svm, @Vm, @Fm, @Rm)
// to the unused area starting at FIRST_UNUSED_CHAR$

For Ctr = 0 To 5
  MapGet[Seq( @Stm ) + Ctr, 1] = Char( FIRST_UNUSED_CHAR$ + Ctr )
Next

For Ctr = 0 to 5
  MapSet[FIRST_UNUSED_CHAR$ + Ctr, 1] = Char(Seq( @Stm ) + Ctr )
Next

Call Set_Property( "SYSTEM", "CHARMAP", mapGet : mapSet )
```

See also

SYSTEM DELIMCOUNT property, SYSTEM UTF8 property, Appendix I – UTF8 Processing.

CMDLINE property

Description

Returns the command line used to start the Presentation Server.

Property Value

The command line string for the current process.

Property Traits

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

Remarks

This property uses the GetCommandW Windows API function internally. The name of the executable in the command line this property returns is not necessarily identical to that in the command line that the calling process gives to the CreateProcess function. The operating system may prepend a fully qualified path to an executable name that is provided without a fully qualified path.

Example

```
CommandLine = Get_Property( "SYSTEM", "CMDLINE" )
```

See also

SYSTEM CONFIGFILE property, SYSTEM CONFIGINFO property, SYSTEM MODULEFILENAME property, SYSTEM RESTART method, Starting the Presentation Server chapter.

COMCTLVERSION property

Description

Returns the version of ComCtl32.dll that has been loaded by the Presentation Server.

Property Value

The version number of ComCtl32.dll that is loaded into the current process. Example versions are:

Version	Description
400	Windows 95/Windows NT 4.0 (v4.00)
470	Internet Explorer 3.x (v4.70)
471	Internet Explorer 4.0 (v4.71)
472	Internet Explorer 4.01 and Windows 98 (v4.72)
580	Internet Explorer 5 (v5.80)
581	Windows 2000 and Windows ME (v5.81)
582	Windows XP, Vista and Windows 7 without XP themes (v5.82)
600	Windows XP with XP themes (v6.00)
610	Windows Vista/7 with XP themes (v6.10)

Property Traits

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

Remarks

There are usually two versions of ComCtl32.dll present on Windows systems since Windows XP/2003:

- Version 6.x is loaded when an application is running in an environment where Windows Visual Styles are enabled.
- Versions earlier than 6.0 are loaded when Visual Styles are not enabled, such as running with the "Windows Classic" theme set for the OS (which is not possible since Windows 8.x).

As well as visual differences, some controls actually behave differently or have reduced functionality dependent on the version loaded, so it is useful to be able to detect this and program accordingly.

Example

```
VersionNo = Get_Property( "SYSTEM", "COMCTLVERSION" )
```

See also

N/a.

CONFIGFILE property

Description

Returns the name and path of the RXI file used to start the application.

Property Value

A string containing the name and path of the RXI file used to initialize the application at startup.

Property Traits

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

Remarks

N/a.

Example

```
RXIFile = Get_Property( "SYSTEM", "CONFIGFILE" )
```

See also

SYSTEM CMDLINE property, SYSTEM CONFIGINFO property, SYSTEM DEVMODE property, SYSTEM LOGININFO property, SYSTEM RUNMODE property, Starting the Presentation Server chapter.

CONFIGINFO property

Description

Returns either:

1. The settings calculated from the combination of the RXI file and command-line switches at startup, or
2. The "raw" settings contained in the RXI file before any command-line switches are applied.

Property Value

This property is an @Fm-delimited array containing the raw or calculated values as extracted from the RXI file. It is structured like so:

```
<1>  AutoExec
<2>  BannerFile
<3>  Caption
<4>  DevMode
<5>  Elevate
<6>  EnginePath
<7>  HideEngine
<8>  HidePS
<9>  MaxInstances
<10> MinDisplaySecs
<11> NoSpy
<12> QueueName
<13> ServerName
<14> ShowBanner
<15> ShowSystemMonitor
<16> SuppressDpiAware
<17> TaskBarID
<18> RunMode
<19> AutoForm
<20> CleanBoot
<21> LoginTemplate
```

Index Value

This is a boolean value. Setting it to FALSE$ (the default) returns the configuration information calculated from the combination of the RXI file and command-line switches at startup. Setting it to TRUE$ returns the "raw" settings contained in the RXI file before any command-line switches are applied.

Property Traits

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

Remarks

Equates for this property can be found in the PS_SYSTEM_EQUATES insert record.

Example

```
$Insert PS_System_Equates
$Insert Logical

// Get the raw data from the RXI file

CfgInfo       = Get_Property( "SYSTEM", "CONFIGINFO", TRUE$ )
RawEnginePath = CfgInfo<PS_SYSCI_POS_ENGINEPATH$>

// Get the calculated data from the RXI file and command-line

CfgInfo    = Get_Property( "SYSTEM", "CONFIGINFO" )
EnginePath = CfgInfo<PS_SYSCI_POS_ENGINEPATH$>
```

See also

SYSTEM CMDLINE property, SYSTEM CONFIGFILE property, SYSTEM DEVMODE property, SYSTEM RUNMODE property, Starting the Presentation Server chapter.

COUNTER property

Description

Returns the current value of the Presentation Server counter, a number which is incremented after each access, thereby providing a unique numeric value for the current process.

Property Value

This property is a positive integer value.

Property Traits

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

Remarks

This property is useful for returning a simple unique number when used with Get_Property. The system itself uses COUNTER to provide unique suffix values for form and control IDs when creating multi-instance windows and tool-panels.

Example

```
CounterVal = Get_Property( "SYSTEM", "COUNTER" )
```

See also

N/a.

CURRENTEVENT property

Description

Returns the name of the currently executing event.

Property Value

This property is a string containing the name of the currently executing event.

Property Traits

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

Remarks

N/a.

Example

```
EventName = Get_Property( "SYSTEM", "CURRENTEVENT" )
```

See also

SYSTEM EVENTSTACK property, SYSTEM QUEUEDEVENTS property, Common POSTEVENT method, Common SENDEVENT method, FORWARD_EVENT stored procedure, GET_CURRENT_EVENT stored procedure, GET_EVENTSTATUS stored procedure, POST_EVENT stored procedure, SEND_EVENT stored procedure, SET_EVENTSTATUS stored procedure.

CURSOR property

Description

Specifies the default cursor image to be used for the application.

Property Value

This property can be one of the following formats:

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

  E.g.

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

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

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

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

Symbol	Description
A	Arrow
H	Wait
I	I-Beam (for text entry)
C	Cross
V	Vertical (Up) Arrow
&	Hand
S	App Starting
?	Help
N	No
+	Size All
****	Size NWSE
/	Size NESW
-	Size WE
|	Size NS
D	DragMove
DC	DragCopy

Property Traits

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

Remarks

This property is used in response to the windows WM_SETCURSOR message to set the cursor shape when the cursor is over an application control or window. It can be overridden by setting the CURSOR property of an individual control or window.

Example

```
// Set the SYSTEM cursor property to an hourglass...
Call Set_Property_Only( "SYSTEM", "CURSOR", "H" )
```

See also

Common GUI CURSOR property, SYSTEM CURSORPOS property, SYSTEM SETCURSOR method.

CURSORPOS property

Description

Retrieves the cursor position for the last window message retrieved by the PS in screen coordinates.

If the name of a form or control is passed in the index argument the returned coordinates will be scaled to DIPs based on the DPI and scale factor of the passed object.

Property Value

This property is an @Fm-delimited array containing the cursor coordinates.

```
<1> X-Position
<2> Y-Position
```

Index Value

If set this should contain the name of a form or control to scale the coordinates too. The coordinates are then returned in DIPs.

Property Traits

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

Remarks

This property is essentially a wrapper around the GetMessagePos Windows API function, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Get the current screen position of the cursor in pixels

CursorPos = Get_Property( "SYSTEM", "CURSORPOS" )

// Get the screen position of the cursor in DIPs with respect
// to the current window

CursorPos = Get_Property( "SYSTEM", "CURSORPOS", @Window )
```

See also

SYSTEM ASYNCCURSORPOS property.

CUSTOMCOLORS property

Description

This property gets or sets the array of custom colors used with the Windows "ChooseColor" common dialog box. This applies to dialogs presented by the CHOOSECOLOR method, and by the COLORDROPDOWN PROPERTYGRID controls.

Property Value

This property is an @Fm-delimited array of RGB color values. The maximum number of colors in this array is 16.

Property Traits

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

Remarks

This property is loaded from the primary instance of the IDE when the system is in development mode and saved when it is closed.

Example

```
// Save the custom colors when the IDE closes
Colors = Get_Property( "SYSTEM", "CUSTOMCOLORS" )

Convert @Fm To @Vm In Colors
Call RTI_IDE_Cfg( "SETVALUE", IDE_CFG_POS_CUSTOM_COLORS$, Colors )

////////////////////////////////////////////////////////////////////////////////

// Reload the saved custom colors for the IDE when it boots
Colors = RTI_IDE_Cfg( "GETVALUE", IDE_CFG_POS_CUSTOM_COLORS$ )

If BLen( Colors ) Then
   Convert @Vm To @Fm In Colors
   Call Set_Property_Only( "SYSTEM", "CUSTOMCOLORS", Colors )
End
```

See also

SYSTEM CHOOSECOLOR method, COLORDROPDOWN control, PROPERTYGRID control

DELIMCOUNT property

Description

Specifies the "delimiter count" used by the application when converting ANSI strings to Unicode strings.

Property Value

This property should be an integer between 0 and 8 (inclusive). The value specifies which single byte characters (counting backwards from 256) should be treated as system delimiters when performing the ANSI to UTF8 or UTF16 translation.

Property Traits

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

Remarks

This property can be set at design time by using the Application Settings dialog from within the IDE Settings menu.

This property was originally called NO_OF_DELIMITERS in previous versions of OpenInsight. NO_OF_DELIMITERS is now an alias for this property to preserve backwards compatibility.

For further information please see Appendix I – UTF8 Processing at the end of this manual.

Example

```
// Get the current delimiter count

DelimCount = Get_Property( "SYSTEM", "DELIMCOUNT" )

// Set the current delimiter count so that only @Rm, @Fm and
// @Vm are preserved as system delimiters when translated into
// Unicode.

Call Set_Property( "SYSTEM", "DELIMCOUNT", 3 )
```

See also

UTF8 property, Appendix I – UTF8 processing.

DEVMODE property

Description

Specifies if the Presentation Server was started in development mode or not.

Property Value

This property is a boolean value. If the PS was started with the "/DV" command line switch set to 1, or the "devMode" setting in the RXI file set to 1, this property returns TRUE$, otherwise it returns FALSE$.

Property Traits

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

Remarks

N/a.

Example

```
IsDevMode = Get_Property( "SYSTEM", "DEVMODE" )
```

See also

Starting the Presentation Server chapter, SYSTEM RUNMODE property.

DEVSYSTEM property

Description

Specifies if the application is licensed for development capabilities.

Property Value

This property is a boolean value. Returns TRUE$ if the application is licensed for development capabilities, or FALSE$ otherwise.

Property Traits

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

Remarks

N/a.

Example

```
IsDevSystem = Get_Property( "SYSTEM", "DEVSYSTEM" )
```

See also

N/a.

DPI property

Description

Returns the System DPI Setting, i.e. the DPI of the primary monitor at logon.

Property Value

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

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

Property Traits

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

Remarks

This property is essentially a wrapper around the GetDeviceCaps Windows API function for the LOGPIXELSX and LOGPIXELSY attributes, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Get the System DPI

SystemDPI = Get_Property( "SYSTEM", "DPI" )
```

See also

Common GUI object DPI Property, Appendix K –High-DPI Programming.

DPIAWARE property

Description

Specifies if the Presentation Server is running in "DPI-Aware" mode. When in "DPI-Aware" mode the Presentation Server forms and controls will respond to changes made to the system or monitor DPI. If this is not the case, then all windows and controls assume they are running with a fixed DPI of 96 and are scaled by the operating system accordingly.

Property Value

This property is a boolean value. It returns FALSE$ if the Presentation Server was loaded with "suppressDPIAware" flag set to "1" in the RXI file, or if the "SD" command line switch was set to "1". If neither of these conditions are met the property returns TRUE$.

Property Traits

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

Remarks

The DPIAWARE property is controlled by the "SD" command line switch or the "suppressDPIAware" RXI configuration file option. See the "Starting the Presentation Server" chapter for more details.

Example

```
IsDpiAware = Get_Property( "SYSTEM", "DPIAWARE" )
```

See also

Starting the Presentation Server chapter, Appendix K – High-DPI Programming.

DPIPERMONITOR property

Description

Specifies if the operating system supports monitors with individual DPI settings. Windows 7 and 8 do not, while Windows 8.1 and later do.

Property Value

This property is a boolean value. It returns TRUE$ if the operating system supports monitors with individual DPI settings, or FALSE$ otherwise.

Property Traits

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

Remarks

N/a.

Example

```
IsDpiPerMonitor = Get_Property( "SYSTEM", "DPIPERMONITOR" )
```

See also

Appendix K – High-DPI Programming.

DRAGSOURCE property

Description

Returns the name of the control whose contents are being dragged during a Drag and Drop operation.

Property Value

This property is a string containing the name of a control. It is only valid during a Drag and Drop operation that was started from a Presentation Server control.

Property Traits

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

Remarks

N/a.

Example

```
DragSourceID = Get_Property( "SYSTEM", "DRAGSOURCE" )
```

See also

SYSTEM DROPTARGET property, Appendix L – Drag and Drop.

DROPTARGET property

Description

Returns the name of the control that is currently the target of a Drag and Drop operation.

Property Value

This property is a string containing the name of a control. It is only valid during a Drag and Drop operation and only if the control under the cursor can accept the data being dragged.

Property Traits

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

Remarks

N/a.

Example

```
DropTargetID = Get_Property( "SYSTEM", "DROPTARGET" )
```

See also

SYSTEM DRAGSOURCE property, Appendix L – Drag and Drop.

DWMCOLORS property

Description

Retrieves the current color-set used for Desktop Window Manager (DWM) non-client frame composition rendering.

Property Value

This property is an @Fm-delimited array containing DWM color values. It is structured like so:

```
<1> ColorizationColor
<2> ColorizationAfterglow
<3> ColorizationColorBalance
<4> ColorizationAfterglowBalance
<5> ColorizationBlurBalance
<6> AccentColor
```

Property Traits

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

Remarks

Note these colors are in "ARGB" format, so the most significant byte is the "Alpha" value.

Equates for this property can be found in the PS_SYSTEM_EQUATES insert record.

Example

```
$Insert PS_System_Equates
$Insert Logical

// Get the DSM Colors and extract the ColorizationColor value (i.e. the frame color)
DWMColorSet       = Get_Property( "SYSTEM", "DWMCOLORS" )
ColorizationColor = DWMColorSet<PS_DWM_POS_COLORIZATIONCOLOR$>


// Extract the RGB from the ARGB value
RGBVal = BitAnd( ColorizationColor, 0x00FFFFFF )
```

See also

N/a.

ENGINEHANDLE property

Description

Returns the window handle (HWND) of the RevEngine form.

Property Value

This property is an integer value containing the Windows handle (HWND) of the RevEngine form.

Property Traits

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

Remarks

The MSWin_ShowWindow stored procedure may be used to show or hide the RevEngine form at runtime (see example below).

MSWin_ShowWindow is a prototyped DLL function exported from USER32.DLL and supplied as part of OpenInsight. Equated constants for use with MSWin_ShowWindow can be found in the MSWIN_SHOWWINDOW_EQUATES insert record.

Example

```
Subroutine ShowEngineForm( ShowEngine )

  // Example : Use the Windows API ShowWindow function to show or hide the
  // RevEngine form

  $Insert MSWin_ShowWindow_Equates

  EngineHWND = Get_Property( "SYSTEM", "ENGINEHANDLE" )
  If ShowEngine Then
     CmdShow = SW_SHOWNORMAL$
  End Else
     CmdShow = SW_HIDE$
  End

  Call MSWin_ShowWindow( EngineHWND, CmdShow )

Return
```

See also

WINDOW VISIBLE property, GetEngineWindow stored procedure.

ENVVARLIST property

Description

Returns a list of all the Windows environment variables and their values.

Property Value

This property is an @Fm-delimited list of Windows environment variable names and their values. Each item in the list has the format:

```
<var_name> "=" <var_value>
```

E.g.

```
HOMEDRIVE=C:
HOMEPATH=\Users\RevGuy
NUMBER_OF_PROCESSORS=8
```

And so on (this format is the same as that used with the "SET" OS command line function).

Property Traits

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

Remarks

This method is a simple wrapper around the Windows API GetEnvironmentStrings function - further information regarding Windows environment variables may be found on the Microsoft website.

Example

```
EnvVars = Get_Property( "SYSTEM", "ENVVARLIST" )
```

See also

SYSTEM GETENVVAR method, SYSTEM SETENVVAR method.

EVENTSTACK property

Description

Returns a list of currently executing events.

Property Value

This property contains an @Fm-delimited list of event information in the order that they were executed (i.e. the currently executing event is always the last item in the list). Each event in the list has the following @Vm-delimited structure:

```
<0,1>  Object ID
<0,2>  Application ID
<0,3>  Repos Type
<0,4>  Work Object ID
<0,5>  Object Type
<0,6>  Event Name
<0,7>  Generic Level
<0,8>  Event Status
<0,9>  Generic Application ID
<0,10> Quick Event Priority Flag
<0,11> Quick Event Handler
<0,12> Quick Event Return Value
```

Property Traits

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

Remarks

Equated constants for use with the EVENTSTACK property may be found in the RTI_EVENTSTACK_EQUATES insert record.

Example

```
// Exmaple - look in the event stack to see if we are being called from a READ event
$Insert RTI_EventStack_Equates

EvStack = Get_Property( "SYSTEM", "EVENTSTACK" )
IsRead  = FALSE$

EvCount = FieldCount( EventStack, @Fm )
For EvIdx = EvCount To 1 Step -1
  If ( EvStack<EvIdx,EVS_POS_EVENT_NAME$> == "READ" ) Then
     IsRead = TRUE$
     EvIdx  = 1 ; // break
  End
Next
```

See also

SYSTEM CURRENTEVENT property, SYSTEM QUEUEDEVENTS property, Common POSTEVENT method, Common SENDEVENT method, FORWARD_EVENT stored procedure, GET_CURRENT_EVENT stored procedure, GET_EVENTSTATUS stored procedure, POST_EVENT stored procedure, SEND_EVENT stored procedure, SET_EVENTSTATUS stored procedure.

EXITCODE property

Description

Returns the exit code of the last process that was executed by the RUNWIN method.

Property Value

This property is a numeric value. When a process exits it normally returns a value to indicate the status of the program when it closed. By convention a successful execution returns an exit code of 0, but this is not enforced.

Property Traits

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

Remarks

When executing a process with RUNWIN in an asynchronous fashion this property is only valid when accessed from the specified callback routine.

If several asynchronous tasks are executed in a simultaneous fashion then the value of the EXITCODE property may not be accurate when accessed in the callback – it is a global property and may be overwritten by a subsequent process's exit code before the callback can be executed.

Example

```
// Execute an asynchronous process with a callback stored procedure
// defined.
CmdLine = "c:\devtools\myproc.exe /D=34 /A=Wibble"

CmdArgs = ""
CmdArgs<1> = 1                    ; // Show Normal, Async
CmdArgs<2> = "MY_CALLBACK_PROC"   ; // Stored Proc to call on exit
CndArgs<3> = "myproc"             ; // Argument to pass to proc
CmdArgs<4> = Drive()              ; // Working Directory

Call Exec_Method( "SYSTEM", "RUNWIN", CmdLine, CmdArgs )

// ================================================================

// Example stored callback proc
Compile Subroutine MY_CALLBACK_PROC( CallBackArg )

   If ( CallBackArg == "myproc" ) Then
      // We got a callback fom "myproc.exe" - Get it's EXITCODE
      // and log it
      ExitCode = Get_Property( "SYSTEM", "EXITCODE" )
      Call UpdateLog( "MyProc.exe returned " : ExitCode )
   End

Return
```

See also

SYSTEM RUNWIN method.

FOCUS property

Description

Returns the name of the control that currently has focus or sets the focus to a specified control.

Property Value

A string containing the fully qualified name of the control with focus. When used with Set_Property this value can be the name of a form to activate, and it may also be null to remove the focus from all PS controls.

Property Traits

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

Remarks

Using this property to move the focus to a PS control will trigger the appropriate focus and activation events. When moving the FOCUS as a result of an error condition (e.g. due to invalid input data) then it is usually desirable to use the BLOCKEVENTS property to stop these events firing, thereby avoiding further validation checks that might send the program into a validation loop.

Example

```
// Example: Get the current focus
FocusCtrl = Get_Property( "SYSTEM", "FOCUS" )


// Example: LOSTFOCUS event to check a code and move the focus
// back to the current control if the data is wrong.
//
// The name of the current control is in the CtrlEntID variable.
$Insert Logical

CtrlData = Get_Property( CtrlEntID, "TEXT" )

Locate CtrlData In "A,B,C" Using "," Setting Pos Then
   // Fine - All Good
   Null
End Else
   Call Msg( @Window, "Bad Data" )

   // Move the focus back to this control without triggering
   // further validation
   Call Set_Property_Only( "SYSTEM", "BLOCKEVENTS", TRUE$ )
   Call Set_Property_Only( "SYSTEM", "FOCUS", CtrlEntID )
   Call Set_Property_Only( "SYSTEM", "BLOCKEVENTS", FALSE$ )

End
```

See also

Common GUI FOCUS property.

FOCUSSTYLES property

Description

Specifies the global "focus style" information for edit-type controls. When an edit-type control (EDITLINE, EDITBOX and EDITTABLE) receives focus then the styles contained in this property are applied. They are intended to help give a better visual indication to the user of where the focus is.

Property Value

This property is an @Fm-delimited array containing the style information:

```
<1> FocusBackcolor
<2> FocusTextColor
<3> CellFocusRectColor
<4> CellFocusRectStyle
<5> CellFocusRectWeight
```

Attribute	Description
FocusBackColor	This color is applied to the focus control's background, or the background of the current cell in an EditTable control.
FocusTextColor	This color is applied to the focus control's text, or the text of the current cell in an EditTable control.
CellFocusRectColor	This color is applied to the focus rectangle drawn around the current cell in an EditTable control.
CellFocusRectStyle	This attribute specifies the style of the focus rectangle drawn around the current cell in an EditTable control. It can be one of the following values: • 0 (Dotted) • 1 (Solid)
CellFocusRectWeight	Specifies the thickness of the style of the focus rectangle drawn around the current cell in an EditTable control. It is an integer value between 1 and 3 inclusive, or -1 to specify the system default value.

Property Traits

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

Remarks

In previous versions of OpenInnsight this information was stored in the Database Environment Settings. From version 10 onwards they have been moved into the Application Settings.

Example

```
// Example: get the current styles and update
$Insert PS_System_Equates
$Insert Colors

FocusStyles = Get_Property( "SYSTEM", "FOCUSSTYLES" )

// Set the background color to yellow, and the focus rect
// to Red, and ensure we are using a solid rect style so
// we see the color.

FocusStyles<PS_FS_POS_BKCOLOR$>        = YELLOW$
FocusStyles<PS_FS_POS_FGCOLOR$>        = CLR_USEDEFAULT$
FocusStyles<PS_FS_POS_CELLRECTCOLOR$>  = RED$
FocusStyles<PS_FS_POS_CELLRECTSTYLE$>  = PS_FS_CELLRECTSTYLE_SOLID$
FocusStyles<PS_FS_POS_CELLRECTWEIGHT$> = PS_FS_POS_CELLRECTWEIGHT_DFLT$

Call Set_Property_Only( "SYSTEM", "FOCUSSTYLES", FocusStyles )
```

See also

Common GUI BACKCOLOR property, Common GUI FOCUS property, Common GUI FORECOLOR property, SYSTEM FOCUS property, EDITBOX ALWAYSSHOWFOCUSCOLORS property, EDITLINE ALWAYSSHOWFOCUSCOLORS property, EDITTABLE FOCUSCELLCOLOR property, EDITTABLE FOCUSRECTCOLOR property, EDITTABLE FOCUSRECTSTYLE property, EDITTABLE FOCUSRECTWEIGHT property, EDITTABLE CellStyle properties.

FONTLIST property

Description

Returns a list containing the names of all available fonts.

Property Value

This property is an @Fm-delimited array of font names.

Property Traits

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

Remarks

N/a.

Example

```
// Example: Get a list of fonts and add them to a combo box called
// CBO_FONTS

FontList = Get_Property( "SYSTEM", "FONTLIST" )
Call Set_Property_Only( @window : ".CBO_FONTS", "LIST", FontList )
```

See also

Common GUI FONT property.

GUITHREADINFO property

Description

Returns information about the PS GUI thread.

Property Value

This property is an @Fm-delimited array containing the GUI thread information for the PS:

```
<1>  Flags
<2>  hwndActive
<3>  hwndFocus
<4>  hwndCapture
<5>  hwndMenuOwner
<6>  hwndMoveSize
<7>  hwndCaret
<8>  CaretSize
```

Attribute	Description
Flags	This bitmask containing the GUI thread flags. See the MSWIN_GUITHREADINFO_EQUATES insert record for more details.
hwndActive	Handle of the active window in the PS GUI thread.
hwndFocus	Handle of the object that has the keyboard focus in the PS GUI thread.
hwndCapture	Handle of the object that has captured the mouse in the PS GUI thread.
hwndMenuOwner	Handle of the object that owns any active menus in the PS GUI thread.
hwndMoveSize	Handle of the window in a move/size loop in the PS GUI thread.
hwndCaret	Handle of the object that is displaying the caret.

CaretSize	Size of the caret in pixels (not DIPs), relative to the client coordinates of the object that contains the caret. This is an @vm-delimited array with the standard size format of:
	
	`<1> X-coordinate`
	`<2> Y-coordinate`
	`<3> Width`
	`<4> Height`

Property Traits

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

Remarks

Equates for this property can be found in the PS_SYSTEM_EQUATES insert record. Equates for the bits used in the "Flags" attribute can be found in the MSWIN_GETGUITHREAD_EQUATES insert record.

This property is essentially a wrapper around the GetGUIThreadInfo Windows API function, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Example - get the PS GUI thread information and output it to
// the System Monitor, adding in the PS name for the returned
// handles where possible

$Insert PS_System_Equates
$Insert RTI_Text_Equates
$Insert Logical

GTI = Get_Property( "SYSTEM", "GUITHREADINFO" )

// Handles are in fields 2 to 7 of the info array
For X = PS_GTI_POS_HWNDACTIVE$ to PS_GTI_POS_HWNDCARET$

   hwnd = GTI<X>
   If hwnd Then
      ObjID = Exec_Method( "SYSTEM", "OBJECTID" )
      If BLen( ObjID ) Then
         GTI<X> = GTI<X> : " (" : ObjID : ")"
      End
   End

Next

Swap @fm with CRLF$ in GTI
Convert @vm to "," in GTI

Call Exec_Method( "SYSTEMMONITOR", "OUTPUT", GTI )
```

See also

Common GUI HANDLE property, SYSTEM FOCUS property, SYSTEM OBJECTID method.

IDLEPROC property

Description

The Presentation Server can maintain a list of stored procedures that can be executed when the system is "idle" (i.e. there are no other Events or Basic+ stored procedures to execute), or at a specific time and date. These are referred to as "idle procedures".

This property returns details of the first item waiting in the "idle procedure" queue, or replaces the entire queue with the details for a single procedure to execute instead.

Property Value

This property is an @Fm-delimited array containing the queued "idle procedure" name and associated data in the following format:

```
<1> Procedure Name
<2> Parameter Value – optional
<3> Execution Time (local time in the format hh:mm:ss) – optional
<4> Execution Date (in the format mm/dd/yy) – optional
```

If the time is null then then procedure will be executed as soon as the Presentation Server is idle. If the date is null then the current date is used as a default value.

Setting the Procedure Name to null will clear the IDLEPROC queue.

Property Traits

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

Remarks

This property is deprecated and implemented for backwards compatibility only. New applications should use the ADDIDLEPROC method instead, thereby preserving the queue contents when adding a new item.

Example

```
// Example - Clear the IDEPROC Queue

Call Set_Property_Only( "SYSTEM", "IDLEPROC", "" )

// Example - Replace the "idle procedure" queue with a new value

IDP    = ""
IDP<1> = "MYPROC"   ; // Procedure Name
IDP<2> = "DoStuff"  ; // Parameter to pass to "MYPROC"
IDP<3> = "12:00:00" ; // Execute at midday
IDP<4> = ""         ; // Execute today

Call Set_Property_Only( "SYSTEM", "IDLEPROC", IDP )
```

See also

SYSTEM IDLEPROC property, SYSTEM ADDIDLEPROC method, Common GUI TIMER event.

IDLEPROCQUEUE property

Description

Returns a list of "idle procedures" waiting to be executed.

Property Value

This property is an @Fm-delimited array of queued "idle procedure" names and associated data. Each item in the queue is an @Vm-delimited array with the following format:

```
<0,1> Procedure Name
<0,2> Parameter Value - can be null
<0,3> Execution Time (local time in the format hh:mm:ss) - optional
<0,4> Execution Date (in the format mm/dd/yy) - optional
```

Property Traits

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

Remarks

N/a.

Example

```
// Example - Check the IDLEPROC Queue to see if a program is waiting to
//           be executed
IPQ = Get_Property( "SYSTEM", "IDLEPROCQUEUE" )

// The queue is returned in "list" format - we want to do a "locate" search
// so we transpose the queue data to an "array" format first.
//
// i.e. We want to transpose something like this:
//
//   "PROC_1", "Param1", "Time1", "Date1"
//   "PROC_2", "Param2", "Time2", "Date2"
//   "PROC_3", "Param3", "Time3", "Date3"
//
// To this:
//
//   "PROC_1", "PROC_2", "PROC_3"
//   "Param1", "Param2", "Param3"
//   "Time1" , "Time2" , "Time3"
//   "Date1" , "Date2" , "Date3"

IPQ = Exec_Method( "SYSTEM", "LIST2ARRAY", IPQ, 0, 0, @Fm, @Vm )

Locate "MYPROC" in IPQ<1> Using @Vm Setting Pos Then
   // Found it
End Else
   // "MYPROC" is not waiting to be executed
End
```

See also

SYSTEM IDLEPROC property, SYSTEM ADDIDLEPROC method.

INTERACTIVE property

Description

Specifies if the Presentation Server is running on an interactive window station (aka. "WinSta0"). An interactive window station is one where a logged-in user can interact with the desktop. A non-interactive window station is one where this is not possible, such as the environment in which service applications run.

Property Value

This property is a boolean value. It returns TRUE$ if the Presentation Server is running on an interactive window station, or FALSE$ otherwise.

Property Traits

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

Remarks

Applications that are not running on an interactive window station should not create modal forms such as dialog boxes because it will not be possible for a user to dismiss them.

Further information on interactive window stations can be found on the Microsoft Website under the topic "Window Station and Desktop Creation".

Example

```
// Example - Check to see if we are running in a non-service
//           context before loading a modal dialog window

IsInteractive = Get_Property( "SYSTEM", "INTERACTIVE" )
If IsInteractive Then
   // It's OK To load a dialog
   RetVal = Dialog_Box( "GET_USER_ANSWER", @Window, "" )
End
```

See also

N/a.

KEYSTATE property

Description

This property returns:

1. The state of a key at the time the last message was read from the Presentation Server's message queue (the key to check is passed as a virtual-key code in the index parameter), or
2. An array representing the state of all virtual keys if no virtual key code is passed.

Property Value

If a virtual-key code was specified this property is an integer value that represents a set of bit flags. As a rule:

- If the key is pressed down the property returns a value < 0 (i.e. The most significant bit is set), otherwise the key is not pressed.
- If the key is toggled (such as the CAPS-LOCK key) then the least significant bit is set (this only applies to keys that can be toggled)

If a virtual-key code was not specified this property returns an @Fm-delimited array of flags for all virtual keys (this is a 256-element array) – the meaning of the flags is as described above.

See remarks for more information.

Property Traits

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

Remarks

This property is essentially a wrapper around the GetKeyboardState and GetKeyState Windows API functions, so please refer to the documentation on the Microsoft website for further information.

Virtual-key code constants are defined in the MSWin_VirtualKey_Equates insert record.

Example

```
$Insert msWin_VirtualKey_Equates

// Get the state of the Shift key
ShiftDown = ( Get_Property( "SYSTEM", "KEYSTATE", VK_SHIFT$ ) < 0 )

// Get the state of the CAPS-LOCK key
CapState   = Get_Property( "SYSTEM", "KEYSTATE", VK_CAPITAL$ )
CapPressed = ( CapState < 0 )
CapToggled = BitAnd( CapState, 0x01 )

// Get the state of all keys and extract the CAPS-LOCK state
KeyboardState = Get_Property( "SYSTEM", "KEYSTATE" )
CapState = KeyboardState<VK_CAPITAL$>
```

See also

SYSTEM ASYNCKEYSTATE property.

LOCALE property

Description

Returns the names of the default user and system locale names.

Property Value

This property is an @Fm-delimited array containing the following information.

```
<1> Default User Locale name
<2> Default System Locale name
```

Property Traits

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

Remarks

This method uses the Windows API GetUserDefaultLocaleName and GetSystemDefaultLocaleName functions internally, so please refer to the Microsoft website for more information on Windows Locales.

Example

```
// Example - Get the Locale names…

LocaleInfo   = Get_Property( "SYSTEM", "LOCALE" )

UserLocale   = LocaleInfo <1>
SystemLocale = LocaleInfo <2>
```

See also

N/a.

LOGININFO property

Description

Returns the credentials used to log into the application.

Property Value

This property is an @Fm-delimited array containing the following information.

```
<1> Application ID
<2> Username
<3> Password
```

Property Traits

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

Remarks

N/a.

Example

```
// Example - Get the user credentials

LoginInfo = Get_Property( "SYSTEM", "LOGININFO" )

AppID     = LoginInfo<1>
Username  = LoginInfo<2>
Password  = LoginInfo<3>
```

See also

SYSTEM CMDLINE property, SYSTEM RESTART method, Starting the Presentation Server chapter.

MESSAGEFONT property

Description

Returns the font used by Windows for displaying text in message boxes.

Property Value

This property is an @Svm-delimited array containing the following font information:

```
<0,0,1> Facename
<0,0,2> Height
<0,0,3> Weight
<0,0,4> Italic
<0,0,5> Underline
<0,0,6> Width
<0,0,7> Charset
<0,0,8> PitchAndFamily
<0,0,9> Strikeout
<0,0,10> OutPrecision
<0,0,11> ClipPrecision
<0,0,12> Quality
```

Property Traits

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

Remarks

The font returned is always scaled to 96 DPI.

Example

```
// Example - Get the Windows Message font, scale to the correct DPI, and set
//           it for a static control called TXT_INFO

MessageFont = Get_Property( "SYSTEM", "MESSAGEFONT" )
ScaledFont  = Exec_Method( @Window : ".TXT_INFO", "SCALEFONT", MessageFont )
Call Set_Property_Only( @Window : ".TXT_INFO", "FONT", ScaledFont )
```

See also

Common GUI FONT property, Common GUI SCALEFONT method, SYSTEM STATUSFONT property SYSTEM SYSTEMFONTS property.

METRICS property

Description

Returns a specified Windows SystemMetrics value.

Property Value

A numeric value containing the requested Windows SystemMetrics data .

Index Value

This should contain the numeric ID of the SystemMetrics value to retrieve. These IDs are defined in the MSWIN_SYSTEMMETRIC_EQUATES insert record.

Property Traits

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

Remarks

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

Please be aware that values returned via this function are always scaled to the System DPI (i.e. the DPI of the primary monitor at logon).

Example

```
// Example - Get the height of a window caption

$insert MsWin_SystemMetric_Equates

CaptionHeight = Get_Property( "SYSTEM", "METRICS", SM_CYCAPTION$ )

// Example - check if we're running in Remote Desktop

IsRDP = Get_Property( "SYSTEM", "METRICS", SM_REMOTESESSION$ )
```

See also

N/a.

MODAL property

Description

Enables or disables all existing top-level Presentation Server forms that belong to the application. This is sometimes referred to as "setting the application modality".

Property Value

This property is a simple boolean value when used with Get_Property. If TRUE$ then all Presentation Server forms are disabled (unless an exception ID was declared – see Set_Property below).

When used with Set_Property this property value is a @Fm-delimited array like so:

```
<1> Boolean disable flag – Set to TRUE$ to disable all Presentation
    Server forms.  Set to FALSE$ to re-enable them.
<2> Exception ID – optional, contains the name of a form to exclude
    when the disable flag is set to TRUE$
```

Property Traits

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

Remarks

N/a.

Example

```
// Example - disable all application forms except the FRM_APPMAIN form

$insert logical

ModalProp = ""
ModalProp<1> = TRUE$          ; // Set Application Modality
ModalProp<2> = "FRM_APPMAIN"  ; // Let FRM_APPMAIN remain enabled

Call Set_Property_Only( "SYSTEM", "MODAL", ModalProp )

// Check the application modality
IsAppModal = Get_Property( "SYSTEM", "MODAL" )
```

See also

Common GUI ENABLED property.

MODULEFILENAME property

Description

Returns the path and file name of the current Presentation Server instance.

Property Value

A string containing the path and module filename for the current process.

Property Traits

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

Remarks

This property uses the GetModuleFileNameW Windows API function internally.

Example

```
PSExeName = Get_Property( "SYSTEM", "MODULEFILENAME" )
```

See also

SYSTEM CMDLINE property, Starting the Presentation Server chapter.

MONITORLIST property

Description

Returns an array of monitor information for all monitors attached to the system. If a control or window is passed as the property index value all coordinates returned are scaled to DIPs with respect to that control or window.

Property Value

A dynamic array containing monitor information. Multiple monitors are delimited by @Fm. Each monitor has the following @Vm-delimited structure:

```
<0,1> Monitor Handle (HMONITOR)
<0,2> Display Rectangle (@Svm delimited)
      <0,2,1> Display Left
      <0,2,2> Display Top
      <0,2,3> Display Right
      <0,2,4> Display Bottom
<0,3> Work-area Rectangle (@Svm delimited)
      <0,3,1> Work-area Left
      <0,3,2> Work-area Top
      <0,3,3> Work-area Right
      <0,3,4> Work-area Bottom
<0,4> Flags
<0,5> Device name
```

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

Property Traits

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

Remarks

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

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

Example

```
// Get the monitor list in actual pixels
MonitorList = Get_Property( "SYSTEM", "MONITORLIST" )

// Get the monitor list in DPIs with respect to the current window
MonitorList = Get_Property( "SYSTEM", "MONITORLIST", @Window )
```

See also

Common GUI MONITOR property, SYSTEM SIZE property, Common GUI DPI property, Common GUI SCREENSIZE property

MOUSECAPTURED property

Description

Retrieves the ID of the Presentation Server GUI object that is capturing the mouse input (if any), along with the window handle (HWND) of the object.

Property Value

This property is an @fm-delimited array containing the mouse capture information:

```
<1> The fully qualified name of a PS object capturing the mouse, or
    null if the mouse is not being captured by a PS object.

<2> The window handle (HWND) of the object capturing the mouse. This
    could be a non-PS object, or it could be an "internal" control of a
    PS object like the editor in a ListBox.
```

Property Traits

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

Remarks

This property is essentially a wrapper around the GetCapture Windows API function, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Get the current screen position of the cursor in pixels

CaptureInfo = Get_Property( "SYSTEM", "MOUSECAPTURED" )
```

See also

Common GUI MOUSECAPTURED property, LOSTCAPTURE event.

PREVFOCUS property

Description

Returns the ID of the Presentation Server object that previously had the focus.

Property Value

This property contains the fully qualified name of a Presentation Server object, or null if the object that previously had the focus was not owned by the Presentation Server.

Property Traits

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

Remarks

This property is updated during the system-level GOTFOCUS processing.

Example

```
PrevFocus = Get_Property( "SYSTEM", "PREVFOCUS" )
```

See also

Common GUI GOTFOCUS event, Common GUI LOSTFOCUS event, Common GUI FOCUS property.

PROCESSID property

Description

Returns the Presentation Server process identifier, a unique number given by Windows that identifies the current process throughout the system.

Property Value

This is a numeric value that contains the process identifier of the current Presentation Server instance.

Property Traits

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

Remarks

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

Example

```
ProcessID = Get_Property( "SYSTEM", "PROCESSID" )
```

See also

N/a.

QUERYEND property

Description

Specifies if the Presentation Server is processing a WM_QUERYENDSESSION message. When a user decides to end their Windows session all running applications are sent a WM_QUERYENDSESSION message – this is to determine if any of them need to ask the user if it is appropriate to close down (i.e. it allows the chance to save data).

The Presentation Server responds to this message by sending each form a CLOSE event (which will trigger save-warning messages), but when this property is set the form is not actually closed - if the end session attempt is aborted for some reason this means that the state of the application is still preserved.

Property Value

This is a boolean value that returns TRUE$ if the system is processing a WM_QUERYENDSESSION message, or FALSE$ otherwise.

Property Traits

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

Remarks

This property is set based on the receipt of a WM_QUERYENDSESSION Windows message, so please refer to the documentation on the Microsoft website for further information on this.

Example

```
IsEndSession = Get_Property( "SYSTEM", "QUERYEND" )
```

See also

WINDOW DESTROYFLAG property, WINDOW CLOSE event.

QUEUEDEVENTS property

Description

Returns a list of events currently queue by the Presentation Server.

Property Value

This property is an @Fm-delimited array containing a list of event information. Each event has the following @Vm-delimited structure:

```
<0,1> Internal Event ID number
<0,2> Repository Type
<0,3> Object ID
<0,4> Object Type
<0,5> Event Qualifier
<0,6> First Event Parameter
<0,7> Second Event Parameter
      ...
<0,n> (n-5)'th Event Parameter
```

Property Traits

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

Remarks

Any system delimiters contained in event parameters present in this list will have been "stepped down" to ensure that they do not corrupt the returned array structure, i.e., @Vm will have been stepped down to @Svm, @Svm to @Tm and so on.

Example

```
// Get the list of events waiting to be executed by the Presentation Server

QueuedEvents = Get_Property( "SYSTEM", "QUEUEDEVENTS" )
```

See also

SYSTEM CURRENTEVENT property, System EVENTSTACK property, Common POSTEVENT method, Common QUALIFYEVENT method, Common SENDEVENT method, FORWARD_EVENT stored procedure, GET_CURRENT_EVENT stored procedure, GET_EVENTSTATUS stored procedure, POST_EVENT stored procedure, SEND_EVENT stored procedure, SET_EVENTSTATUS stored procedure.

RECEIVER property

Description

Specifies the name of a control to receive and display the contents of Basic+ Send_Dyn function calls.

Property Value

This property is a string containing the fully qualified name of a control. It must be one of the following types:

- EDITBOX
- LISTBOX
- EDITTABLE

The property value can also be set to null to ignore Send_Dyn data.

Property Traits

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

Remarks

N/a.

Example

```
// Example - set the current window's LST_DATA ListBox as the
// SYSTEM RECEIVER object

Call Set_Property_Only( "SYSTEM", "RECEIVER", @Window : ".LST_DATA" )
```

See also

Send_Dyn stored procedure, WINDOW STATUSLINE property.

RUNMODE property

Description

Specifies if the Presentation Server was started in runtime mode or not. This property is normally set by the "runMode" option in the RXI file, or the /RN command-line switch.

Property Value

This property is an integer value:

- 0 or null. Runtime mode is not specified – allows Development mode to be set.
- 1 – Runtime mode – allows any app to run.
- 2 – Runtime mode – allows only a specific app to run.

Property Traits

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

Remarks

Constants for use with the RUNMODE property can be found in the PS_SYSTEM_EQUATES insert record.

Example

```
RunMode = Get_Property( "SYSTEM", "RUNMODE" )
```

See also

Starting the Presentation Server chapter, SYSTEM DEVMODE property.

SERVERNAME property

Description

Returns the ID of the Named Pipe or the TCP/IP Address and Port used to communicate with the RevEngine VM.

Property Value

This property is a string containing one of the following:

- A string containing a Named Pipe.
- A string containing an IP address and a port delimited by the ":" character.

Property Traits

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

Remarks

The SERVERNAME property can be set via the "serverName" option in the RXI file, or by the /SN command line switch. By default the Presentation Server generates a random string to use as a Named Pipe name if neither is specified.

Example

```
ServerName = Get_Property( "SYSTEM", "SERVERNAME" )
```

See also

Starting the Presentation Server chapter.

SHOWACCELERATORS property

Description

When set to TRUE$ this property overrides the Windows default setting that hides the keyboard shortcuts when a form is loaded from a Mouse action rather than from a Keyboard action (this setting was first introduced along with Visual Styling in Windows XP).

Property Value

This property is a boolean value. When set to TRUE$ keyboard shortcut indicators are always displayed, otherwise they are only shown when a form is created from a keyboard action. This property defaults to TRUE$.

Property Traits

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

Remarks

Earlier versions of OpenInsight always displayed keyboard shortcuts, hence this property defaults to TRUE$ to maintain backwards compatibility.

This setting can be controlled at design time by using the Application Settings dialog box in the OpenInsight IDE.

More information on this topic can be found by reviewing the Microsoft documentation on the following messages:

- WM_QUERYUISTATE
- WM_CHANGEUISTATE
- WM_UPDATEUISTATE

Example

```
// Get the current SHOWACCELERATORS setting

IsShowAccelerators = Get_Property( "SYSTEM", "SHOWACCELERATORS" )

// Ensure they are always shown

Call Set_Property( "SYSTEM", "SHOWACCELERATORS", TRUE$ )
```

See also

N/a.

SHUTDOWN property

Description

This property returns TRUE$ if the SYSTEM DESTROY method has been invoked and all Presentation Server windows have been destroyed.

When set to TRUE$ this property causes the system to shut down (in the same way as the SYSTEM DESTROY method). Once this has been set to TRUE$ this property cannot be used to cancel a shutdown request.

Property Value

This property is a boolean value.

Property Traits

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

Remarks.

N/a.

Example

```
// Check if the Presentation Server is shutting down

IsShutdown = Get_Property( "SYSTEM", "SHUTDOWN" )

// Shut the Presentation Server down

Call Set_Property( "SYSTEM", "SHUTDOWN", TRUE$ )
```

See also

SYSTEM DESTROY method.

SIZE property

Description

Returns the width and height of the primary monitor along with the width and height of its work-area (i.e. the space not occupied by the taskbar).

Property Value

This property is an @Fm-delimited array formatted as follows:

```
<1> Monitor Width (SM_CXSCREEN)
<2> Monitor Height (SM_CYSCREEN)
<3> Monitor Workarea Width (SM_CXFULLSCREEN)
<4> Monitor Workarea Height (SM_CYFULLSCREEN)
```

All dimensions are usually returned in DIPs, scaled to the "current form" (i.e. "@Window"), but this can be overridden by specifying a property index value:

- If the property index value is "*" then the returned array values are in pixels.
- If the property index value contains the name of a valid form or control then the array values are returned in DIPs, scaled to that object.

Property Traits

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

Remarks

This property is implemented for backwards compatibility only. New applications should use the SYSTEM MONITORLIST and WINDOW MONITOR properties to ensure that they work smoothly with multi-monitor scenarios.

The method uses the Windows GetSystemMetrics function to obtain the details for this property, so please refer to the documentation on the Microsoft website for further information on this. The "SM_" flags used with the function are as noted above.

Example

```
SystemSize = Get_Property( "SYSTEM", "SIZE" )        ; // DIPs - Use @Window
SystemSize = Get_Property( "SYSTEM", "SIZE", winID ) ; // DIPs - Use winID
SystemSize = Get_Property( "SYSTEM", "SIZE", "*" )   ; // Pixels
```

See also

SYSTEM MONITORLIST property, SYSTEM METRICS property, Common GUI MONITOR property.

STATUSFONT property

Description

Returns the font used by Windows for displaying text in status line and tooltip controls.

Property Value

This property is an @Svm-delimited array containing the following font information.

```
<0,0,1> Facename
<0,0,2> Height
<0,0,3> Weight
<0,0,4> Italic
<0,0,5> Underline
<0,0,6> Width
<0,0,7> Charset
<0,0,8> PitchAndFamily
<0,0,9> Strikeout
<0,0,10> OutPrecision
<0,0,11> ClipPrecision
<0,0,12> Quality
```

Property Traits

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

Remarks

The font returned is always scaled to 96 DPI.

Example

```
// Example - Get the Windows Status font, scale to the correct DPI, and set
//           it for a static control called TXT_STATUS

StatusFont = Get_Property( "SYSTEM", "STATUSFONT" )
ScaledFont  = Exec_Method( @Window : ".TXT_STATUS", "SCALEFONT", StatusFont )
Call Set_Property_Only( @Window : ".TXT_STATUS", "FONT", ScaledFont )
```

See also

Common GUI FONT property, Common GUI SCALEFONT method, SYSTEM MESSAGEFONT property, SYSTEM SYSTEMFONTS property.

SUPPRESSAUTODESTROY property

Description

When the Presentation Server detects that there are no more visible windows in the application, and if the System Monitor is hidden, then it automatically shuts down the application and exits. Setting this value to TRUE$ prevents this behaviour.

Property Value

This property is a boolean value.

Property Traits

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

Remarks.

Care should be taken when using this property to ensure that there are no hidden windows that prevent the system from shutting down. If you suppress the auto-destroy process then you are responsible for closing down the system cleanly.

Example

```
// Check if the Presentation Server can auto-destroy itself

IsSuppressAutoDestroy = Get_Property( "SYSTEM", "SUPPRESSAUTODESTROY" )

// Stop the Presentation Server from automatically shutting down

Call Set_Property( "SYSTEM", "SUPPRESSAUTODESTROY", TRUE$ )
```

See also

SYSTEM DESTROY method, SYSTEM SHUTDOWN property.

SYSTEMFONTS property

Description

Returns the fonts used by Windows for displaying text in various parts of the system.

Property Value

This property is an @Fm-delimited array containing the following font information:

```
<1> Caption Font
<2> Small Caption Font
<3> Menu Font
<4> Status Font
<5> Message Font
```

Each font is an @Svm-delimited array describing the font attributes:

```
<0,0,1> Facename
<0,0,2> Height
<0,0,3> Weight
<0,0,4> Italic
<0,0,5> Underline
<0,0,6> Width
<0,0,7> Charset
<0,0,8> PitchAndFamily
<0,0,9> Strikeout
<0,0,10> OutPrecision
<0,0,11> ClipPrecision
<0,0,12> Quality
```

Property Traits

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

Remarks

The font information returned is always scaled to 96 DPI.

Equated constants for use with this property can be found in the PS_SYSTEM_EQUATES insert record. Constants for use with font attributes can be found in the PS_FONT_EQUATES insert record.

Example

```
// Example - Get the Windows Menu font
$Insert PS_System_Equates

MenuFont = Get_Property( "SYSTEM", "SYSTEMFONTS" )<PS_SYSFONT_POS_MESSAGE$>
```

See also

Common GUI FONT property, SCALEFONT method, SYSTEM MESSAGEFONT property, SYSTEM STATUSFONT property.

TASKBARID property

Description

Returns the Taskbar identifier for the current Presentation Server instance. It can be set by the "taskBarID" option in the RXI file or the /TB command line switch.

Property Value

This property is a simple string containing the Taskbar identifier.

Property Traits

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

Remarks

Under Windows 7 and 8 multiple instances of an executable launched from the same directory are grouped under one icon on the taskbar, making navigating between them quite tedious. To allow an executable to override this behavior Microsoft introduced a new application property called the "Application User Model ID" – a simple string that can be applied to an executable instance to differentiate it from other sibling instances, thereby allowing them to be un-grouped.

Windows assigns a default value to this property at runtime based on the executable name and the starting directory. The OpenInsight /TB switch (or "taskBarID" element in an RXI file) allows you to set your own unique value for this property so that your application can be differentiated on the Windows Taskbar.

The Application User Model ID must be set before any forms are created by an executable which is why it can only be set via the RXI file or the command line switch.

If you wish to find out more details about the Application User Model ID please see the Microsoft documentation about the SetCurrentProcessExplicitAppUserModelID function.

Example

```
TaskBarID = Get_Property( "SYSTEM", "TASKBARID" )
```

See also

Starting the Presentation Server chapter, WINDOW TASKBARID property.

THEMED property

Description

Returns TRUE$ if the Presentation Server is running with Windows Visual styles enabled.

Property Value

This property is a boolean value.

Property Traits

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

Remarks

This property always returns TRUE$ on Windows 8.0 and higher as Windows visual styling cannot be turned off on those systems.

On Windows 7 this property returns FALSE$ if Windows Classic styling is applied.

Example

```
IsOSThemed = Get_Property( "SYSTEM", "THEMED" )
```

See also

N/a.

TIMEZONE property

Description

Returns time zone information the current Presentation Server instance.

Property Value

This property is an @Fm-delimited dynamic array formatted as follows:

```
<1> TimeZone ID
<2> Bias (minutes)
<3> Standard Name
<4> Standard DateTime
<5> Standard Bias
<6> Daylight Name
<7> Daylight DateTime
<8> Daylight Bias
```

Property Traits

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

Remarks

This property uses the GetTimeZoneInformation Windows API function internally, so please refer to the documentation on the Microsoft website for more information.

Constants for use with the TIMEZONE property can be found in the PS_SYSTEM_EQUATES insert record.

Example

```
TimeZoneInfo = Get_Property( "SYSTEM", "TIMEZONE" )
```

See also

N/a.

TYPES property

Description

Returns a list of all object types supported by the Presentation Server.

Property Value

This property is an @Fm-delimited dynamic array of type names.

Property Traits

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

Remarks

N/a.

Example

```
TypeArray = Get_Property( "SYSTEM", "TYPES" )
```

See also

TYPE Property.

UTF8 property

Description

Specifies if the application is running in ANSI mode or UTF8 mode.

Property Value

This property is a boolean value. If the PS is running in UTF8 mode this property is set to TRUE$, otherwise it is set to FALSE$.

Property Traits

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

Remarks

This property can be set at design time by using the Application Settings dialog from within the IDE Settings menu.

For further information please see Appendix I – UTF8 Processing at the end of this document.

Example

```
// Get the current UTF8 mode

IsUTF8 = Get_Property( "SYSTEM", "UTF8" )

// Turn on UTF8 mode

Call Set_Property( "SYSTEM", "UTF8", TRUE$ )
```

See also

SYSTEM DELIMCOUNT property, Appendix I – UTF8 processing.

VERSION property

Description

Returns Windows and Presentation Server version information.

Property Value

This property is an @Fm-delimited array formatted as follows:

```
<1> Windows Version Number in the format:
    <majorVersion> “.” <minorVersion>

<2> Presentation Server Product version in the format:
    <majorVersion> “.” <minorVersion> “.” <releaseNo> “.” <buildNo>

<3> Presentation Server File version in the format:
    <majorVersion> “.” <minorVersion> “.” <releaseNo> “.” <buildNo>
```

Property Traits

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

Remarks

N/a.

Example

```
VersionInfo = Get_Property( "SYSTEM", "VERSION" )
```

See also

N/a.

VISIBLE property

Description

Specifies if the OpenInsight IDE (i.e. the RTI_IDE form) is visible or not.

Property Value

This property is an integer value. It has the same qualities as the normal WINDOW VISIBLE property.

Property Traits

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

Remarks

N/a.

Example

```
IsVisibleIDE = Get_Property( "SYSTEM", "VISIBLE" )
```

See also

WINDOW VISIBLE property.

WIN64 property

Description

Returns a pair of flags denoting if the OS and the Presentation Server are 64-bit processes.

Property Value

This property is an @Fm-delimited array formatted as follows.

```
<1> OS 64-bit flag – Returns TRUE$ if Windows is 64-bit.
<2> PS 64-bit flag – Returns TRUE$ if the Presentation Server is a 64-
    bit process.
```

Property Traits

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

Remarks

N/a.

Example

```
Win64Flags = Get_Property( "SYSTEM", "WIN64" )
```

See also

N/a.

WINCOUNT property

Description

Returns the number of WINDOW objects (forms) running in the current Presentation Server instance.

Property Value

This property is integer value.

Property Traits

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

Remarks

N/a.

Example

```
WinCount = Get_Property( "SYSTEM", "WINCOUNT" )
```

See also

N/a.

WINDOWGHOSTING property

Description

Specifies if "Window Ghosting" is active for the current Presentation Server instance.

Property Value

This property is boolean value. It returns TRUE$ if Window Ghosting is active (the default), or FALSE$ otherwise. *Once it has been set to FALSE$ it cannot be turned back on until the Presentation Server is restarted.*

Property Traits

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

Remarks

"Window Ghosting" is a system function that Windows initiates when it thinks that a process has become unresponsive, typically because it gets involves in a long, time-consuming operation and doesn't check its message queue periodically by calling the Yield function. When this happens, Windows adds a "Not Responding" message to the form's caption, but unfortunately this can appear to end users to be an error condition. It also allows the user to move or close the form as well, which could lead to other problems.

The ideal way to deal with this situation is to refactor long operations in the application so that they call the Yield function periodically (every few seconds at least) so Windows is notified that the Presentation Server has not crashed or run into an infinite loop.

If this cannot be done the WINDOWGHOSTING property can be set to FALSE$ to stop the "Not Responding" behavior. However, in this case the user won't be able to minimize, move, or close the main window of the application if it is actually not responding due to an error condition and they will have to use the Task Manager to close it instead – this may give the impression that the application is unstable!

This method is essentially a wrapper around the DisableProcessWindowsGhosting Windows API function – for further information please see the Microsoft website.

Example

```
// Turn off Window Ghosting for the rest of the session
Call Set_Property_Only ( "SYSTEM", "WINDOWGHOSTING", FALSE$ )
```

See also

SYSTEM PROCESSWINMSGS method, Yield stored procedure.

SYSTEM Methods

The SYSTEM object supports the following methods:

Name	Description
ADDIDLEPROC	Adds a stored procedure to the IDLEPROC queue.
ALPHACOLOR	Calculates a new color value when a transparency value is applied to it.
ARRAY2LIST	Converts data from an EDITTABLE ARRAY format to an EDITABLE LIST format.
BEEP	Plays a Windows system sound.
CHOOSECOLOR	Executes the Windows Common Choose Color Dialog to select a color.
CHOOSEDIR	Executes the Windows Common Browse Folder dialog to select a folder name.
CHOOSEFILE	Executes the Windows Common File Dialog to select a file name.
CHOOSEFONT	Executes the Windows Common Font Dialog to select a font.
CREATE	Creates one or more GUI objects from a passed structure.
CREATEGUID	Creates and returns a string containing a GUID.
DARKENCOLOR	Returns a darkened version of a specified color.
DESTROY	Destroys a specified object.
FINDEXE	Returns name of an executable file associated with a specified document file.
FLUSH	Removes all pending events from the PS event queue.
FORMBYCURSOR	Returns the name of the form beneath the specified cursor position.
GETENVVAR	Returns the value of a Windows environment variable.
GETIMAGEINFO	Returns basic information about an image file.
HANDLEBYCURSOR	Returns the handle (HWND) of a control or form underneath the cursor.
LIGHTENCOLOR	Returns a lightened version of a specified color.
LIST2ARRAY	Converts data from an EDITTABLE LIST format to an EDITABLE ARRAY format.
LOGEVENT	Writes a message to the Windows Event Log.
MAPWINDOWPOINTS	Maps a set of points from one coordinate space to another.
MIXCOLORS	Returns the result of mixing two colors using weighted ratios and a luminosity value.
OBJECTID	Returns the name of a PS GUI object matching the passed handle (HWND).
OBJECTBYCURSOR	Returns the name of a PS GUI object underneath the cursor.
OBJECTLIST	Returns a list of objects matching the specified filter criteria.
OLEGETPICTUREPROPS	Returns properties for an OLE Picture object.

OLEIUNKNOWNRELEASE	Releases an OLE object.
OLELOADPICTURE	Loads an image file as an OLE Picture object.
POSTWINMSG	Posts a windows message to a GUI object.
PROCESSEVENTS	Processes all pending events in the event queue,
PROCESSWINMSGS	Processes all pending messages in the Windows message queue.
REFLECTEVENTS	Returns published event information for an object type.
REFLECTMETHODS	Returns published method information for an object type.
REFLECTPROPERTIES	Returns published property information for an object type.
RESTART	Restarts the Presentation Server.
RUNHELP	Displays a Windows Compiled HTMLHelp (CHM) file or an old-style WinHelp (HLP) file.
RUNWIN	Launches an application.
SENDWINMSG	Sends a windows message to a GUI object.
SETCURSOR	Sets the cursor to the specified image.
SETENVVAR	Sets the value of a Windows environment variable.
SHELLEXEC	Launches an application via an associated document name.
SHOWMESSAGE	Displays a non-owned message box.
SHOWPOPUP	Displays a non-owned popup box.
STARTFORM	Executes a non-owned form.
TEXTRECT	Calculates the size of the area needed to display a specified string.
TRANSLATEKEYDOWN	Synthesizes and sends a WM_KEYDOWN or a WM_SYSKEYDOWN window message to an object.

ADDIDLEPROC method

Description

The Presentation Server can maintain a list of stored procedures (the "IDLEPROC queue") that can be executed when the system is "idle" (i.e. there are no other Events or Basic+ stored procedures to execute), or at a specific time and date. These are referred to as "idle procedures".

The method appends a procedure and its associated data to the end of this queue.

Syntax

```
SuccessFlag = Exec_Method( "SYSTEM",       |
                           "ADDIDLEPROC",  |
                           ProcName,       |
                           Parameter,      |
                           ExecTime,       |
                           ExecDate )
```

Parameters

Name	Required	Description
ProcName	Yes	Name of the stored procedure to add to the queue.
Parameter	No	Parameter to pass to the stored procedure when it is executed.
ExecTime	No	Local time that the stored procedure should be executed, in the format "hh:mm:ss". If this is not specified, the stored procedure will be executed as and when the Presentation Server is idle.
ExecDate	No	Local date that the stored procedure should be executed, in the format "mm/dd/yy".

Returns

Returns TRUE$ if the item was added to the successfully, or FALSE$ otherwise.

Remarks

The contents of the queue can be examined via the IDLEPROCQUEUE property.

Example

```
// Example - add a request to the IDLEPROC queue Execute a stored procedure
// called MYPROC, passing it a parameter of "DoStuff" and execute it at midday

IsAdded = Exec_Method( "SYSTEM", "IDLEPROC", |
                       "MYPROC",             |
                       "DoStuff",            |
                       "12:00:00",           |
                       "" )
```

See also

SYSTEM IDLEPROC property, SYSTEM IDLEPROCQUEUE method, Common GUI TIMER event.

ALPHACOLOR method

Description

This method the calculates a new color value when a transparency value is applied to it.

Syntax

```
ColorAlpha = Exec_Method( "SYSTEM", "ALPHACOLOR", Color, AlphaValue )
```

Parameters

Name	Required	Description
Color	Yes	Color to adjust.
AlphaValue	Yes	Percentage transparency to apply (value from 0 to 100).

Returns

Contains the adjusted color or "-1" if the color cannot be adjusted.

Remarks

N/a.

Example

```
// Get the system BTNFACE color and create a 20% “transparent” version of it.

Color      = Get_Property( @Window, "VISUALSTYLECOLOR", SYSCOLOR_BTNFACE$ )
AlphaColor = Exec_Method( "SYSTEM", "ALPHACOLOR", Color, 20 )
```

See also

SYSTEM DARKENCOLOR method, SYSTEM LIGHTENCOLOR method, SYSTEM MIXCOLORS method.

ARRAY2LIST method

Description

Utility method to convert data stored in an EDITTABLE ARRAY property structure (Column/Row) to an EDITTABLE LIST property structure (Row/Column).

Syntax

```
List = Exec_Method( "SYSTEM",       |
                    "ARRAY2LIST",   |
                    Array,          |
                    ColSize,        |
                    RowSize,        |
                    PrimaryDelim,   |
                    SecondaryDelim )
```

Parameters

Name	Required	Description
Array	Yes	Data to convert. This should be in the same format as the data used with the EDITTABLE ARRAY property, i.e. columns are delimited by @Fm, and rows delimited by @Fm within each column.
ColSize	No	Number of columns to convert. If not specified the system determines the number of columns itself. Defaults to 0.
RowSize	No	Number of rows to convert. If not specified the system determines the number of rows itself. Defaults to 0.
PrimaryDelim	No	Delimiter to use when parsing the Array for columns, and when delimiting rows in the returned List. Defaults to @Fm.
SecondaryDelim	No	Delimiter to use when parsing the Array for rows, and when delimiting columns in the returned List. Defaults to @Vm.

Returns

Converted data structured as per the EDITTABLE LIST property, i.e. rows are delimited by the PrimaryDelim (@Fm) and columns are delimited by SecondaryDelim (@Vm) in each row.

Remarks

ColSize and RowSize are optional. They do not limit the actual number of columns and rows processed by this method – they are simply used as a guideline to help with some internal memory optimization, which may help performance when processing very large arrays.

Example

```
// Get the contents of an EDITTABLE control in ARRAY format and convert
// it to the LIST format  (Yes, you could just get the LIST property
// but this is for demo purposes).

ObjxArray =         @Window : ".EDT_MYDATA"
PropArray =         "ARRAY"

ObjxArray := @Rm : @Window : ".EDT_MYDATA"
PropArray := @Rm : "LIMIT"

DataArray         = Get_Property( ObjxArray, PropArray )

EdtArray          = DataArray[1,@Rm]
EdtDims           = DataArray[Col2()+1,@Rm]

// Convert the format
EdtList = Exec_Method( "SYSTEM", "ARRAY2LIST", |
                       EdtArray,               |
                       EdtDims<1>              |
                       EdtDims<2> )
```

See also

N/a.

BEEP method

Description

Method to play a Windows system sound.

Syntax

```
Call Exec_Method( "SYSTEM", "BEEP", SoundID )
```

Parameters

Name	Required	Description
SoundID	No	Identifies the sound to play. This is a numeric value that corresponds to one of the standard Windows system sounds. **Sound** / **Value** / **Equated to** **OK** / 0x00 / MB_OK$ **Stop** / 0x10 / MB_ICONSTOP$ **Question** / 0x20 / MB_ICONQUESTION$ **Warning** / 0x30 / MB_ICONWARNING$ **Information** / 0x40 / MB_ICONINFORMATION$ The default is "OK" (MB_OK$)

Returns

N/a.

Remarks

This method is a simple wrapper around the Windows API MessageBeep function so further information on this can be found on the Microsoft website.

Equated constants for use with the BEEP method can be found in the MSWIN_MESSAGEBOX_EQUATES insert record.

Example

```
// Emit the Windows Warning sound

$Insert MSWin_MessageBox_Equates

Call Exec_Method( "SYSTEM", "BEEP", MB_ICONWARNING$ )
```

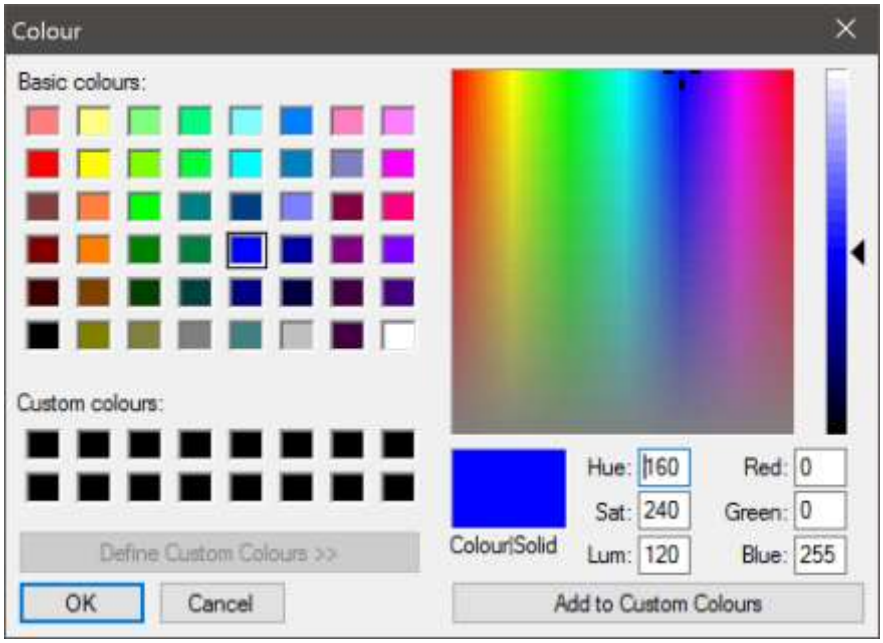
See also

N/a.

CHOOSECOLOR method

Description

Displays the "Choose Color" Windows Common Dialog Box to allow the user to select a color:



Syntax

```
Color = Exec_Method( "SYSTEM",          |
                     "CHOOSECOLOR",     |
                     OwnerWindow,       |
                     InitialColor )
```

Parameters

Name	Required	Description
OwnerWindow	No	ID of the parent form for the dialog. This can be null, in which case the parent window is the desktop.
InitialColor	No	Initial color to select when the dialog is displayed.

Returns

The color value the user selected, or null if the user clicked the "Cancel" button.

Remarks

The dialog supports a set of 16 custom colors that may be chosen by the user. They may be accessed at runtime via the SYSTEM CUSTOMCOLORS property and they are cached between instances of the dialog being executed and loaded automatically. They are not saved between OI sessions.

The CHOOSECOLOR method is basically a wrapper around the ChooseColor Windows API function, so further information on this can be found on the MSDN website.

Example

```
// Display the Windows ChooseColor dialog to allow the user to select a new
// FORECOLOR for a control, using the current window as the parent

InitColor = Get_Property( ctrlID, "FORECOLOR" )

NewColor  = Exec_Method( "SYSTEM", "CHOOSECOLOR", @Window, InitColor )

If BLen( NewColor ) Then
   // The user selected a color
   Call Set_Property_Only( ctrlID, "FORECOLOR", NewColor )
End
```

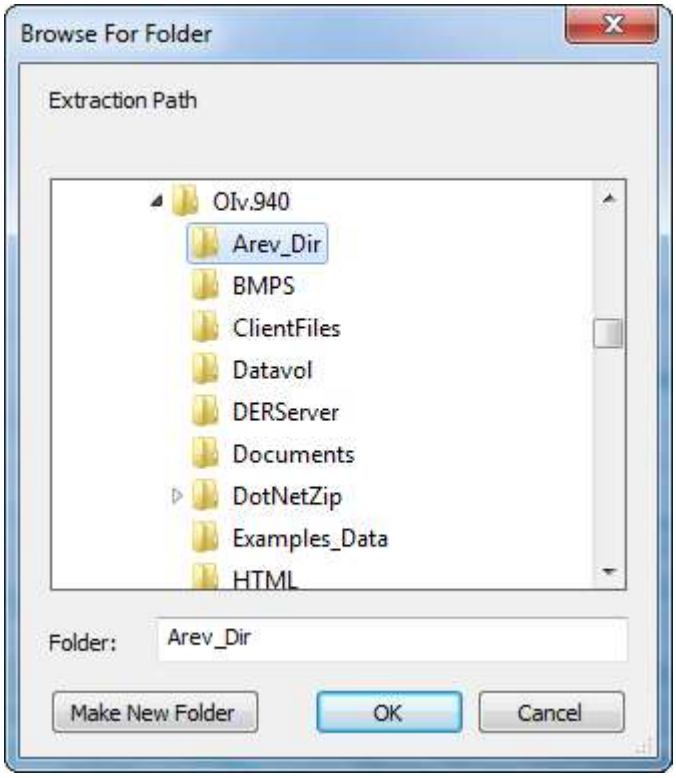
See also

SYSTEM CUSTOMCOLORS property.

CHOOSEDIR method

Description

Displays the "Select Folder" Windows Common Dialog Box to allow the user to select a directory or folder name:



Remarks

This method has been moved to the FILESYSTEM object, though it may still be called from the SYSTEM object for backwards compatibility purposes.

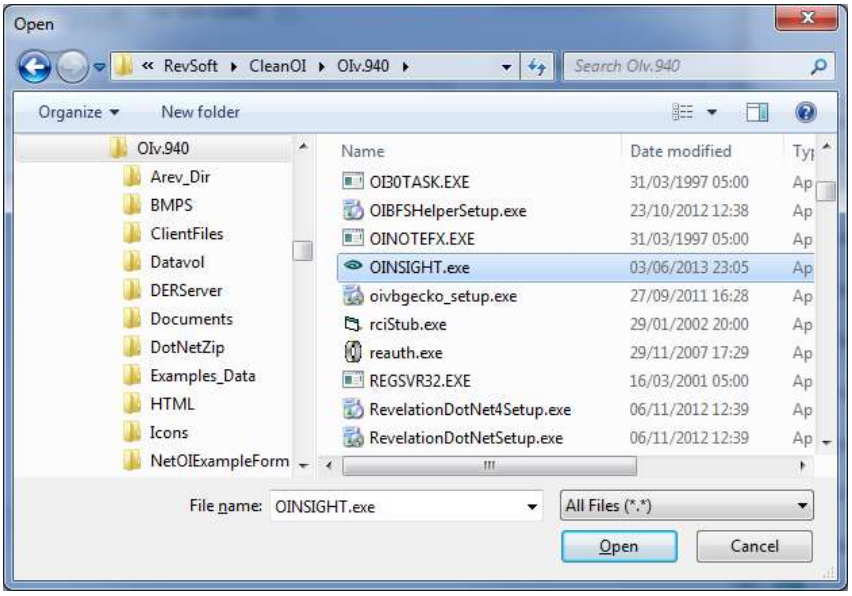
See also

FILESYSTEM CHOOSEDIR method.

CHOOSEFILE method

Description

Displays the "Open File" or "Save As" Windows Common Dialog Box to allow the user to select a file name:



Remarks

This method has been moved to the FILESYSTEM object, though it may still be called from the SYSTEM object for backwards compatibility purposes.

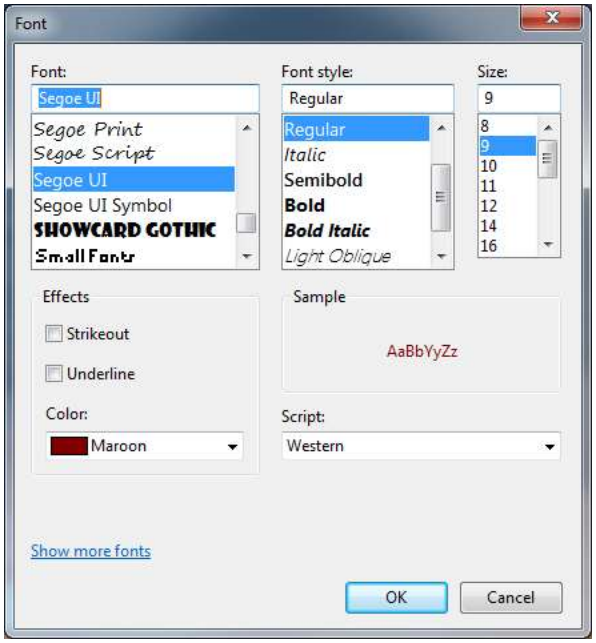
See also

FILESYSTEM CHOOSEFILE method.

CHOOSEFONT method

Description

Displays the "Choose Font" Windows Common Dialog Box to allow the user to select a font:



Syntax

```
Font = Exec_Method( "SYSTEM",          |
                    "CHOOSEFONT",      |
                    OwnerWindow,       |
                    ChooseFontOptions, |
                    HideColors )
```

Parameters

Name	Required	Description
OwnerWindow	No	ID of the parent form for the dialog. This can be null, in which case the parent is the desktop.

ChooseFontOptions	No	Contains an @Fm-delimited dynamic array of options that control the behavior of the dialog. It has the following structure:
		
		`<1> @Svm-delimited FONT structure`
		`<2> Flags`

HideColors	No	If TRUE$ then hide the color selection combobox.

Returns

An @Svm-delimited FONT structure in the format:

```
<0,0,1>  FaceName
<0,0,2>  Height
<0,0,3>  Weight
<0,0,4>  Italic
<0,0,5>  Underline
<0,0,6>  Width
<0,0,7>  CharSet
<0,0,8>  PitchAndFamily
<0,0,9>  Strikeout
<0,0,10> OutPrecision
<0,0,11> ClipPrecision
<0,0,12> Quality
```

If the HideColors option is not set then the selected color RGB values are appended to the structure like so:

```
<0,0,13> Red
<0,0,14> Green
<0,0,15> Blue
```

Remarks

The ChooseFontOptions argument is composed of two fields which control the behavior of the dialog – each field is described fully below:

Field	Name	Description
<1>	InitialFont	Specifies the font to preselect in the dialog. This can be: 1. An @Svm-delimited structure in the same format as the return value described above, or 2. The name of an existing control whose FONT property is used to preselect in the dialog.
<2>	Flags	A set of bitmask flags that specify the dialog behavior. These are fully described in the Microsoft documentation for the "flags" member of the CHOOSEFONT structure, and equates constants for these flags can be found in the MSWIN_CHOOSEFONT_EQUATES insert record (See below for an example of using them).

The CHOOSEFONT method is basically a wrapper around the ChooseFont Windows API function, so it is worth examining at the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Equated constants for use with the CHOOSEFONT method can be found in the PS_CHOOSEFONT_EQUATES and MSWIN_CHOOSEFONT_EQUATES insert records.

Example

```
// CHOOSEFONT example - get the FONT property of an existing control
// and allow the user to choose a new font ignoring colors.

CtrlFont = Get_Property( @Window : ".EDL_NAME", "FONT" )

NewFont = Exec_Method( "SYSTEM", "CHOOSEFONT", CtrlFont, TRUE$ )

If BLen( NewFont ) Then
   Call Set_Property_Only( @Window : ".EDL_NAME", "FONT", NewFont )
End
```

See also

Common GUI FONT property, SYSTEM MESSAGEFONT property, SYSTEM STATUSFONT property.

CREATE method

Description

This method creates one or more GUI objects from a passed structure.

Syntax

```
SuccessFlag = Exec_Method( "SYSTEM", "CREATE", CreateStruct, ScaleUnits )
```

Parameters

Name	Required	Description
CreateStruct	Yes	This is a @Fm-delimited dynamic array containing a list of object structures to create. Each object has its own @Vm-delimited structure that specifies its attributes such as Name, Type, Parent, Position and so on. Constants that describe the core structure of an object can be found in the PS_EQUATES insert record. There are further "PS_" records that cover each type in more detail.
ScaleUnits	No	Specifies if the units defined in the CreateStruct should be interpreted as DIPs (the default) or Pixels. This value is the same as the common GUI SCALEUNITS property.

Returns

This method returns TRUE$ if all objects were created successfully, or FALSE$ otherwise.

Remarks

The preferred way to check if an object was created successfully is to check that its HANDLE property is valid after the CREATE method returns.

This is a "raw" low-level method to instruct the Presentation Server to create one or more objects. It will not change the underlying "window common area" to add in any new entries for the newly created objects, such as synthetic properties (like VALID and CONV), or QuickEvent handlers. These must be updated via other means.

Example

```
// Example - create a new STATIC control called "TXT_INFO" based on the
//           structure of an existing one (TXT_TEMPLATE)

$Insert PS_Equates

// Get the original structure for the STATIC
CtrlStruct = Get_Property( @Window : ".TXT_TEMPLATE", "ORIG_STRUCT" )

// Update the name, text and position
CtrlStruct<PSPOS_NAME$> = @Window : ".TXT_INFO"
CtrlStruct<PSPOS_TEXT$> = "Here are the gory details"
CtrlStruct<PSPOS_X$>    = 10
CtrlStruct<PSPOS_Y$>    = 20

IsCreated = Exec_Method( "SYSTEM", "CREATE", CreateStruct )
```

See also

Common GUI HANDLE property, Common GUI ORIG_STRUCT property, Common GUI SCALEUNITS property, SYSTEM DESTROY method.

CREATEGUID method

Description

Method to create a GUID, a unique 128-bit integer used for CLSIDs and interface identifiers.

Syntax

```
GUID = Exec_Method( "SYSTEM", "CREATEGUID" )
```

Parameters

N/a.

Returns

A string containing a newly created GUID in printable hexadecimal characters. This has the format:

```
{nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn}
```

E.g.

```
{DA1E9F6A-4AD1-4777-AD81-5274D2252AF0}
```

Remarks

This method is a simple wrapper around the Windows API CoCreateGuid and StringFromGUID2 functions – for further information on GUIDs please see the MSDN website.

Example

```
// Create a string GUID to use as an ID

ThisID = Exec_Method( "SYSTEM", "CREATEGUID" )
```

See also

RTI_CreateGuid function.

DARKENCOLOR method

Description

This method returns a darkened version of the specified color.

Syntax

```
DkColor = Exec_Method( "SYSTEM", "DARKENCOLOR", Color, Amount )
```

Parameters

Name	Required	Description
Color	Yes	Color to darken.
Amount	Yes	Amount by which to darken (between 0 and 1.0)

Returns

The darkened color.

Remarks

N/a.

Example

```
// Get the system BTNFACE color and create a darker version of it.

Color   = Get_Property( @Window, "VISUALSTYLECOLOR", SYSCOLOR_BTNFACE$ )
DkColor = Exec_Method( "SYSTEM", "DARKENCOLOR", Color, 0.2 )
```

See also

SYSTEM ALPHACOLOR method, SYSTEM LIGHTENCOLOR method, SYSTEM MIXCOLORS method.

DESTROY method

Description

Destroys a specified object, optionally removing it from the Window Common Area if appropriate.

Syntax

```
bDestroyed = Exec_Method( "SYSTEM",        |
                          "DESTROY",       |
                          ObjectID,        |
                          CommRemoveFlag )
```

Parameters

Name	Required	Description
ObjectID	Yes	ID of the object to be destroyed. If this is "SYSTEM" then the Presentation Server will close the current application and exit. Use this to programmatically shut down the system.
CommRemoveFlag	No	If TRUE$ then the object is removed from the Window Common area if appropriate (see remarks below). Defaults to FALSE$

Returns

TRUE$ if the object was destroyed successfully, FALSE$ otherwise.

Remarks

If the object to be destroyed is a control hosted on a form then some extra details, such as database and validation information, are stored in an area of memory called the "Window Common Area". Setting the "CommRemoveFlag" to TRUE$ removes this information as well.

Do not use this method to close a form - use the CLOSE event or an appropriate function such as End_Dialog instead.

The following intrinsic objects cannot be destroyed with the SYSTEM DESTROY method:

- CLIPBOARD
- FILESYSTEM
- SYSTEMMONITOR

Note that there also are other "sub-objects", such as IMAGE, GLYPH, and TABS etc., which are managed by their respective owners and cannot be destroyed directly

by the SYSTEM DESTROY method. This behavior will be noted in the individual sections detailing them.

Example

```
$Insert Logical

// Destroy a control on a form and ensure data is removed
// from the window common area

ObjectID = @Window : ".EDL_NAME"
Call Exec_Method( "SYSTEM", "DESTROY", ObjectID, TRUE$ )

// Close the application
Call Exec_Method( "SYSTEM", "DESTROY", "SYSTEM" )
```

See also

SYSTEM CREATE method.

FLUSH method

Description

This method removes all pending events from the PS event queue.

Syntax

```
nRemoved = Exec_Method( "SYSTEM", "FLUSH" )
```

Parameters

N/a.

Returns

The number of events removed from the event queue.

Remarks

N/a.

Example

```
// Remove all pending events.

nEventsRemoved = Exec_Method( "SYSTEM", "FLUSH" )
```

See also

SYSTEM BLOCKEVENTS property, SYSTEM GETEVENT method, SYSTEM QUEUEDEVENTS property, Common POSTEVENT method, Common QUALIFYEVENT method, Common SENDEVENT method, Send_Event function, Post_Event function, Appendix C – Event Handling.

FINDEXE method

Description

This method returns name and path of an executable file (.exe) associated with the specified document file.

Syntax

```
ExeName = Exec_Method( "SYSTEM", "FINDEXE", File, Directory )
```

Parameters

Name	Required	Description
File	Yes	Name of the document file to find the executable for.
Directory	No	The default directory for the search.

Returns

The name and path of the executable file associated with the specified document. If no association can be found a numeric Windows "SE_ERR_" code is returned instead.

Remarks

This method is a simple wrapper around the Windows API FindExecutableW function - further information regarding this may be found on the Microsoft website.

Example

```
// Example - find the exe associated with a text file

TextExe = Exec_Method( "SYSTEM", "FINDEXE", "test.txt", "c:\temp" )
```

See also

SYSTEM SHELLEXEC method.

FORMBYCURSOR method

Description

This method returns the name of a PS form underneath the cursor.

Syntax

```
Form = Exec_Method( "SYSTEM", "FORMBYCURSOR", ScreenXY, DPIObject )
```

Parameters

Name	Required	Description
ScreenXY	No	Screen coordinates to use for the search. If these are not passed the current cursor position is used. <1> X coordinate <2> Y coordinate
DPIObject	No	Contains the name of a GUI object (control or form) to use for DPI scaling. If this parameter is passed then the ScreenXY is considered to be scaled to the DPIObject's DPI (i.e. DIPs rather than pixels), and the ScreenXY will be converted to pixels before the search is made. This parameter is ignored if the ScreenXY parameter is not passed.

Returns

The name of the PS form under the cursor (or passed screen coordinates), or null otherwise.

In previous versions of OpenInsight this method was called WINDOW_BY_POS. This name can still be used for backwards compatibility.

Remarks

This method is essentially a wrapper around the Windows API WindowFromPoint function – for further information please see the Microsoft website.

Example

```
// Example - find the PS form under the screen cursor

HotForm = Exec_Method( "SYSTEM", "FORMBYCURSOR", "", "", "" )
```

See also

SYSTEM FORMBYCURSOR method , SYSTEM HANDLEBYCURSOR method.

GETENVVAR method

Description

This method returns the value of a Windows environment variable.

Syntax

```
EnvVarValue = Exec_Method( "SYSTEM", "GETENVVAR", EnvVarName )
```

Parameters

Name	Required	Description
EnvVarValue	Yes	Name of the environment variable to query.

Returns

The value of the specified Windows environment variable.

Remarks

This method is a simple wrapper around the Windows API GetEnvironmentVariable function - further information regarding Windows environment variables may be found on the Microsoft website.

Example

```
// Get the value of the Windows "SystemDrive" variable

SessionName = Exec_Method( "SYSTEM", "GETENVVAR", "SystemDrive" )
```

See also

SYSTEM ENVVARLIST property, SYSTEM SETENVVAR method.

GETIMAGEINFO method

Description

This method returns basic information about an image file.

Syntax

```
ImageInfo = Exec_Method( "SYSTEM", "GETIMAGEINFO", FileName )
```

Parameters

Name	Required	Description
FileName	Yes	Name and path of the image file to query.

Returns

An @Fm-delimited array of image information formatted as follows:

```
<1> Image Width (pixels)
<2> Image Height (pixels)
<3> Image FrameCount
```

Remarks

N/a.

Example

```
// Get the dimensions of an image

ImageInfo = Exec_Method( "SYSTEM", "GETIMAGEINFO", "c:\temp\test.png"

ImageWidth  = ImageInfo<1>
ImageHeight = ImageInfo<2>
```

See also

The IMAGE Object API chapter.

HANDLEBYCURSOR method

Description

This method returns the handle (HWND) of a GUI object underneath the cursor.

Syntax

```
Hwnd = Exec_Method( "SYSTEM", "HANDLEBYCURSOR", ScreenXY, FormOnly, DPIObject )
```

Parameters

Name	Required	Description
ScreenXY	No	Screen coordinates to use for the search. If these are not passed the current cursor position is used.
FormOnly	No	If TRUE$ then this method returns the handle of the parent form under the cursor, rather than any child windows.
DPIObject	No	Contains the name of a GUI object (control or form) to use for DPI scaling. If this parameter is passed then the ScreenXY is considered to be scaled to the DPIObject's DPI (i.e. DIPs rather than pixels), and the ScreenXY will be converted to pixels before the search is made. This parameter is ignored if the ScreenXY parameter is not passed.

Returns

The handle (HWND) of the GUI object under the cursor (or passed screen coordinates). This handle can be a non-PS object.

Remarks

This method is essentially a wrapper around the Windows API ChildWindowFromPoint function – for further information please see the Microsoft website.

Example

```
// Example - find the object under the screen cursor and check if it's a
// PS object.

HwndHot = Exec_Method( "SYSTEM", "HANDLEBYCURSOR", "", "", "" )

If HwndHot Then
   HotID = Exec_Method( "SYSTEM", "OBJECTID", HwndHot )
   If BLen( HotID ) Then
      // We have a PS object underneath the cursor
   End
End
```

See also

Common GUI HANDLE property, SYSTEM FORMBYCURSOR method, SYSTEM OBJECTBYCURSOR method, SYSTEM OBJECTID method.

LIGHTENCOLOR method

Description

This method returns a lightened version of the specified RGB color value.

Syntax

```
LtColor = Exec_Method( "SYSTEM", "LIGHTENCOLOR", Color, Amount )
```

Parameters

Name	Required	Description
Color	Yes	RGB color value to lighten.
Amount	Yes	Amount by which to lighten (between 0 and 1.0)

Returns

The lightened RGB color value.

Remarks

N/a.

Example

```
// Get the system BTNFACE color and create a lighter version of it.

Color   = Get_Property( @Window, "VISUALSTYLECOLOR", SYSCOLOR_BTNFACE$ )
LtColor = Exec_Method( "SYSTEM", "LIGHTENCOLOR", Color, 0.1 )
```

See also

SYSTEM ALPHACOLOR method, SYSTEM LIGHTENCOLOR method, SYSTEM MIXCOLORS method.

LIST2ARRAY method

Description

Utility method to convert data stored in an EDITTABLE LIST property structure (Row/Column) to an EDITTABLE ARRAY property structure (Column)/Row.

Syntax

```
Array = Exec_Method( "SYSTEM",       |
                     "LIST2ARRAY",   |
                     List,           |
                     RowSize,        |
                     ColSize,        |
                     PrimaryDelim,   |
                     SecondaryDelim )
```

Parameters

Name	Required	Description
List	Yes	Data to convert. This should be in the same format as the data used with the EDITTABLE LIST property, i.e. rows are delimited by @Fm, and columns are delimited by @Vm within each row.
RowSize	No	Number of rows to convert. If not specified the system determines the number of rows itself. Defaults to 0.
ColSize	No	Number of columns to convert. If not specified the system determines the number of columns itself. Defaults to 0.
PrimaryDelim	No	Delimiter to use when parsing the List for rows, and when delimiting columns in the returned Array. Defaults to @Fm.
SecondaryDelim	No	Delimiter to use when parsing the List for columns, and when delimiting rows in the returned Array. Defaults to @Vm.

Returns

Converted data structured as per the EDITTABLE ARRAY property, i.e. columns are delimited by the PrimaryDelim (@Fm) and rows are delimited by SecondaryDelim (@Vm) in each column.

Remarks

ColSize and RowSize are optional. They do not limit the actual number of columns and rows processed by this method – they are simply used as a guideline to help with some internal memory optimization, which may help performance when processing very large arrays.

Example

```
// Get the contents of an EDITTABLE control in LIST format and convert
// it to the ARRAY format  (Yes, you could just get the ARRAY property
// but this is for demo purposes).

ObjxArray =        @Window : ".EDT_MYDATA"
PropArray =        "LIST"

ObjxArray := @Rm : @Window : ".EDT_MYDATA"
PropArray := @Rm : "LIMIT"

DataArray        = Get_Property( ObjxArray, PropArray )

EdtList          = DataArray[1,@Rm]
EdtDims          = DataArray[Col2()+1,@Rm]

// Convert the format
EdtArray = Exec_Method( "SYSTEM", "LIST2ARRAY", |
                        EdtList,                |
                        EdtDims<2>              |
                        EdtDims<1> )
```

See also

SYSTEM ARRAY2LIST method.

LOGEVENT method

Description

This method writes a message to the Windows Event Log.

Syntax

```
IsLogged = Exec_Method( "SYSTEM", "LOGEVENT", TypeInfo, SourceInfo, |
                        MessageText )
```

Parameters

Name	Required	Description
TypeInfo	See Description	An @Fm-delimited array containing the following fields: <1> Message Type (Required): Must be one of the following values: "ERROR" "WARNING" "INFO" <2> Event ID (Optional): Integer denoting the event. Defaults to 1. <3> Category ID (Optional) Integer identifying the category. Defaults to 0.
SourceInfo	See Description	An @Fm-delimited array containing the following fields: <1> Event Source (Required): Name of the Event Source (See notes on the Event Source below for more details). <2> Server Name (Optional): UNC name of the system to Post the message to. Defaults to the local Workstation.
MessageText	Yes	Message to write to the Event Log.

Returns

TRUE$ if the message is logged successfully, or FALSE$ otherwise.

Remarks

This method is essentially a wrapper around the Windows API ReportEvent and other Event Log related functions – for further information please see the Microsoft website.

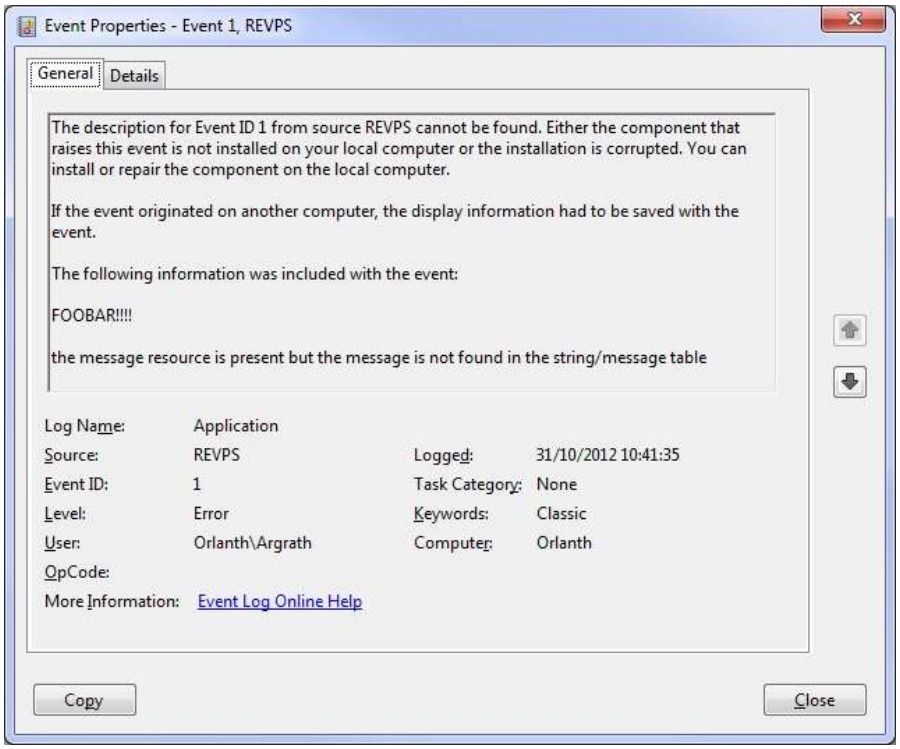
Example

```
// Example - Log a FOOBAR error message for RevPS in the Windows Event Log

IsLogged = Exec_Method( "SYSTEM", "LOGEVENT", "ERROR", "RevPS", "FOOBAR!!!!" )
```

The Event Source

If the above example is executed the message will be displayed in the Application Event Log, but Windows will prefix the message with some of its own text which refers to a missing Event ID description like so:



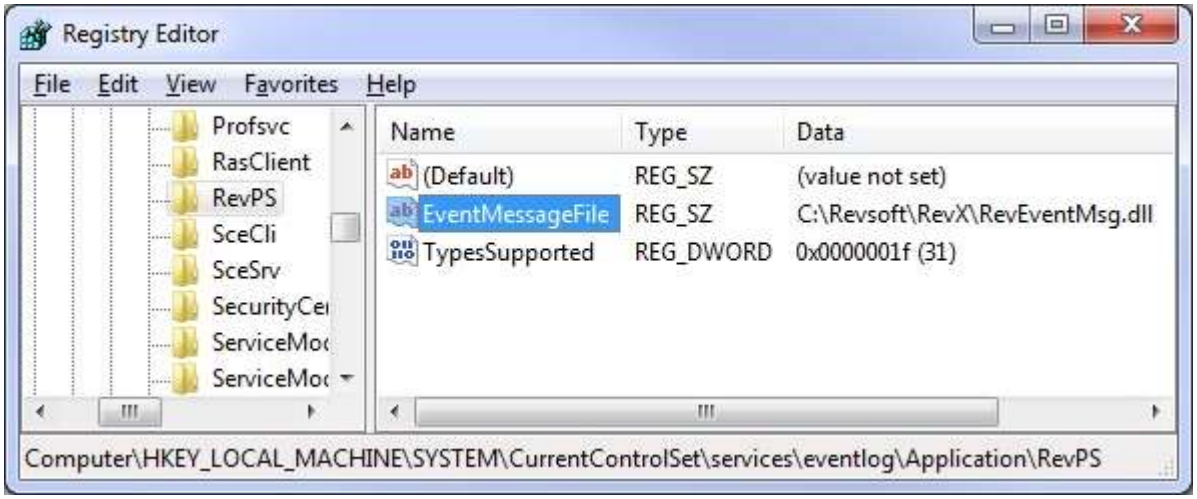
The reason for this is that Windows expects to find a "registered Event Source" containing the description for the Event ID that was specified. A registered Event Source is actually a DLL containing a set of strings, each of which corresponds to an Event ID. Without this DLL the warning text displayed above is prefixed, which could give the impression that something is missing from the application.

To avoid this OpenInsight provides a generic DLL called RevEventMsg.dll that can be registered on a workstation system under the desired Event Source name – if that name is then used in a LOGEVENT call the message will be logged without any of the warning text prefixed to it.

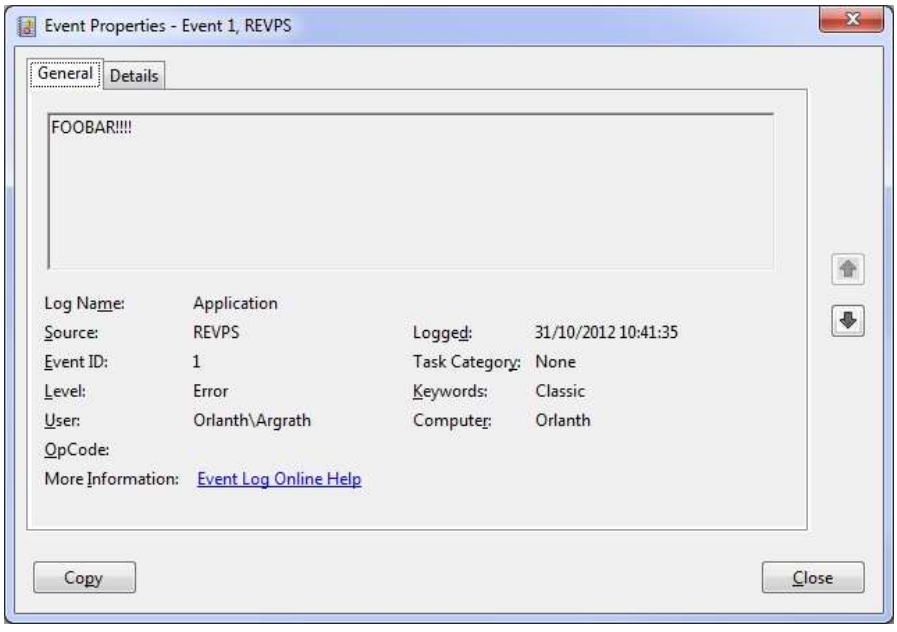
Registering the DLL is quite simple: create a new key with the name of the Event Source (The string "RevPS" was used for this example) under this path:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\eventlog\Application\
```

And then set the following values:



If the message is logged again the text is no longer prefixed:



See also

N/a.

MAPWINDOWPOINTS method

Description

This method converts (maps) a set of points from a coordinate space relative to a form or control to a coordinate space relative to another form or control.

Syntax

```
PointsTo = Exec_Method( "SYSTEM", "MAPWINDOWPOINTS", MapFrom, MapTo,
                        PointsFrom )
```

Parameters

Name	Required	Description
MapFrom	No	ID of the object to map from. The passed PointsFrom are assumed to be in DIPs relative to this object unless: • If MapFrom is null then PointsFrom are assumed to be Pixel coordinates relative to the desktop. • If MapFrom is numeric then it is assumed to be a window handle (HWND) and PointsFrom are assumed to be Pixel coordinates.
MapTo	No	ID of the object to map to. The returned PointsTo will be in DIPs relative to this object unless: • If MapTo is null then the returned PointsTo will be Pixel coordinates relative to the desktop. • If MapTo is numeric then it is assumed to be a window handle (HWND), and the returned PointsTo will be Pixel coordinates.
PointsFrom	Yes	An @Fm delimited list of points to map.

Returns

The mapped set of points in the same format as the PointsFrom parameter. These may be in DIPs or Pixels depending on the contents of the MapFrom and MapTo parameters

Remarks

This method is essentially a wrapper around the Windows API MapWindowPoints function – for further information please see the Microsoft website.

Example

```
// Example - map the RECT coordinates of an editline to another form.

MapFrom    = "MYWIN.EDL_DATA"
MapTo      = "ANOTHERWIN"

SourceRect = Get_Property( MapFrom, "RECT" )

DestRect = Exec_Method( "SYSTEM", "MAPWINDOWPOINTS", MapFrom, MapTo, SourceRect )
```

See also

Common GUI PARENT property, Common GUI RECT property, Common GUI SIZE property.

MIXCOLORS method

Description

This method returns the result of mixing two colors using weighted ratios and a luminosity value.

Syntax

```
MixedColor = Exec_Method( "SYSTEM", "MIXCOLORS", Color1, Color2, LumRatio, |
                          Weight1, Weight2 )
```

Parameters

Name	Required	Description
Color1	Yes	RBG value of the first color to mix.
Color2	Yes	RBG value of the second color to mix.
LumRatio	No	The ratio for the new color's luminosity. MIXCOLORS multiplies the luminosity of the mixed color by this ratio. Defaults to 1.
Weight1	No	The weighted ratio for the first color. Defaults to 1.
Weight2	No	The weighted ratio for the second color. Defaults to 1.

Returns

The mixed color value.

Remarks

N/a.

Example

```
// Mix yellow and green for a darker, muddy green color

Color1 = 65535 ; // Yellow
Color2 = 65280 ; // Green
LRatio = 0.6   ; // Make it darker

NewColor = Exec_Method( "SYSTEM", "MIXCOLORS", Color1, Color2, LRatio, 1, 1 )
```

See also

SYSTEM ALPHACOLOR method, SYSTEM DARKENCOLOR method, SYSTEM LIGHTENCOLOR method.

OBJECTBYCURSOR method

Description

This method returns the name of a PS GUI object underneath the cursor.

Syntax

```
Hwnd = Exec_Method( "SYSTEM", "OBJECTBYCURSOR", ScreenXY, DPIObject )
```

Parameters

Name	Required	Description
ScreenXY	No	Screen coordinates to use for the search. If these are not passed the current cursor position is used.
DPIObject	No	Contains the name of a GUI object (control or form) to use for DPI scaling. If this parameter is passed then the ScreenXY is considered to be scaled to the DPIObject's DPI (i.e. DIPs rather than pixels), and the ScreenXY will be converted to pixels before the search is made. This parameter is ignored if the ScreenXY parameter is not passed.

Returns

The name of the PS GUI object under the cursor (or passed screen coordinates), or null otherwise.

Remarks

This method is essentially a wrapper around the Windows API ChildWindowFromPoint function – for further information please see the Microsoft website.

Example

```
// Example - find the PS object under the screen cursor

hotID = Exec_Method( "SYSTEM", "OBJECTBYCURSOR", "", "", "" )
```

See also

SYSTEM FORMBYCURSOR method , SYSTEM HANDLEBYCURSOR method.

OBJECTID method

Description

This method returns the name of a PS GUI object matching the passed handle (HWND).

Syntax

```
ObjectID = Exec_Method( "SYSTEM", "OBJECTID", HwndObject )
```

Parameters

Name	Required	Description
HwndObject	Yes	Handle (HWND) of the GUI object to query.

Returns

The name of the PS GUI object matching the passed handle, or null otherwise.

Remarks

N/a.

Example

```
// Example - find the object under the screen cursor and check if it's a
// PS object.

HwndHot = Exec_Method( "SYSTEM", "HANDLEBYCURSOR", "", "", "" )

If HwndHot Then
   HotID = Exec_Method( "SYSTEM", "OBJECTID", HwndHot )
   If BLen( HotID ) then
      // We have a PS object underneath the cursor
   End
End
```

See also

HANDLE property, SYSTEM OBJECTBYCURSOR method, SYSTEM OBJECTID method.

OBJECTLIST method

Description

This method returns a list of object IDs that match the specified filter criteria and options.

Syntax

```
ObjList = Exec_Method( "SYSTEM", "OBJECTLIST", ParentID, TypeID, RecurseFlag, |
                       NoSortFlag, VisibleOnlyFlag, PageNumber )
```

Parameters

Name	Required	Description
ParentID	No	If specified then only objects that are children of this object will be returned.
TypeID	No	If specified then only objects of this type are returned, otherwise objects of any type may be returned.
RecurseFlag	No	Normally, when the ParentID parameter is set only direct children of that parent will be returned. If RecurseFlag is set to TRUE$ then all objects that are descendants of ParentID are returned.
NoSortFlag	No	By default the list of objects returned in alphabetical order – if this parameter is TRUE$ then the list is returned with respect to the Z-order instead.
VisibleOnlyFlag	No	If TRUE$ then only visible objects are returned.
PageNumber	No	If specified then only objects with the same PageNumber property will be returned.

Returns

An @Fm-delimited list of PS objects matching the filter criteria, or null if no matches are found.

Remarks

This method uses the Windows API EnumChildWindows function internally – for further information please see the Microsoft website.

Example

```
// Return a sorted list of all objects in the system
ObjList = Exec_Method( "SYSTEM", "OBJECTLIST", "", "" )

// Return a sorted list of all EDITFIELD objects in the system
ObjList = Exec_Method( "SYSTEM", "OBJECTLIST", "", "EDITFIELD" )

// Return a z-order list of direct child objects for the MYWIN form
ObjList = Exec_Method( "SYSTEM", "OBJECTLIST", "MYWIN", "", FALSE$, TRUE$ )

// Return a z-order list of ALL children for the MYWIN form
ObjList = Exec_Method( "SYSTEM", "OBJECTLIST", "MYWIN", "", TRUE$, TRUE$ )

// Return a sorted list of ALL EDITFIELD children for the MYWIN form
ObjList = Exec_Method( "SYSTEM", "OBJECTLIST", "MYWIN", "EDITFIELD", TRUE$, FALSE$ )

// Return a sorted list of ALL visible children for the MYWIN form
ObjList = Exec_Method( "SYSTEM", "OBJECTLIST", "MYWIN", "", TRUE$, TRUE$, TRUE$ )

// Return a z-order list of all visible children on page 2 of the MYWIN form
ObjList = Exec_Method( "SYSTEM", "OBJECTLIST", "MYWIN", "", TRUE$, FALSE$, TRUE$, 2 )
```

See also

System Monitor OL and LO commands, Common TYPE property, Common GUI PAGENUMBER property, Common GUI PARENT property, Common GUI PARENTFORM property, Common GUI VISIBLE property.

OLEGETPICTUREPROPS method

Description

Returns a dynamic array of properties for an OLE Picture object (IPicture) when passed its interface pointer.

Syntax

```
PicProps  = Exec_Method( "SYSTEM", "OLEGETPICTUREPROPS", pObject )
```

Parameters

Name	Required	Description
pObject	Yes	IUnknown/IID_Picture Interface pointer for the Picture object.

Returns

An @Fm-delimited array of properties for the picture:

```
<1> Picture handle (OLE_HANDLE)
<2> Palette handle (HPAL)
<3> Picture Type (PICTYPE)
<4> Width
<5> Height
<6> Current Device Context (HDC)
<7> KeepOriginalFormat flag (BOOL)
<8> Attributes
```

If the method fails nothing is returned.

Remarks

This method uses the OLE IPicture interface to retrieve the details for the picture object. For further information please see the IPicture topic on the Microsoft website.

Example

```
// Extract the width and height of an OLE Picture, given it's
// interface pointer
PicFile  = "C:\temp\myPic.bmp"
pPicture = Exec_Method( "SYSTEM", "OLELOADPICTURE", PicFile )

If pPicture Then
   PicProps = Exec_Method( "SYSTEM", "OLEGETPICTUREPROPS", pPicture )
   If BLen( PicProps ) Then
      PicW = PicProps<4>
      PicH = PicProps<5>
   End

   // Release the picture
   Exec_Method( "SYSTEM", "OLEIUNKNOWNRELEASE", pPicture )

End
```

See also

SYSTEM OLELOADPICTURE method, SYSTEM OLE IUKNOWNRELEASE method.

OLEIUNKNOWNRELEASE method

Description

Calls the Release method for an OLE object when passed its interface pointer.

Syntax

```
IsReleased  = Exec_Method( "SYSTEM", "OLEIUNKNOWNRELEASE", pObject )
```

Parameters

Name	Required	Description
pObject	Yes	IUnknown Interface pointer for the object.

Returns

TRUE$ if the object is released, or FALSE$ otherwise.

Remarks

This method uses the OLE Release method on the IUnknown interface. For further information please see the IUnknown topic on the Microsoft website.

You must ensure that the pointer passed to this method is valid, otherwise a GPF may occur.

Example

```
// Extract the width and height of an OLE Picture, given it's
// interface pointer
PicFile  = "C:\temp\myPic.bmp"
pPicture = Exec_Method( "SYSTEM", "OLELOADPICTURE", PicFile )

If pPicture Then
   PicProps = Exec_Method( "SYSTEM", "OLEGETPICTUREPROPS", pPicture )
   If BLen( PicProps ) Then
      PicW = PicProps<4>
      PicH = PicProps<5>
   End

   // Release the picture
   Exec_Method( "SYSTEM", "OLEIUNKNOWNRELEASE", pPicture )

End
```

See also

SYSTEM OLELOADPICTURE method.

OLELOADPICTURE method

Description

Loads an image file and an OLE Picture object (IPicture) and returns the its interface pointer.

Syntax

```
pPicture = Exec_Method( "SYSTEM", "OLELOADPICTURE", FileName )
```

Parameters

Name	Required	Description
FileName	Yes	Name and path of the image file to load.

Returns

The IPicture interface pointer if the image is loaded successfully, or 0 if the method fails.

Remarks

This method uses the OleCreatePictureIndirect or OleLoadPicturePath Windows API functions to load the picture object. For further information on these functions please see the Microsoft website.

Example

```
// Extract the width and height of an OLE Picture, given it's
// interface pointer
PicFile  = "C:\temp\myPic.bmp"
pPicture = Exec_Method( "SYSTEM", "OLELOADPICTURE", PicFile )

If pPicture Then
   PicProps = Exec_Method( "SYSTEM", "OLEGETPICTUREPROPS", pPicture )
   If BLen( PicProps ) Then
      PicW = PicProps<4>
      PicH = PicProps<5>
   End

   // Release the picture
   Exec_Method( "SYSTEM", "OLEIUNKNOWNRELEASE", pPicture )

End
```

See also

SYSTEM OLEGETPICTUREPROPS method, SYSTEM OLE IUKNOWNRELEASE method.

POSTWINMSG method

Description

This method uses the Windows API PostMessage function to place a message in the message queue associated with the thread that created the target object and returns without waiting for the thread to process the message.

Syntax

```
SuccessFlag = Exec_Method( "SYSTEM", "POSTWINMSG", hwnd, uMsg, wParam, lParam )
```

Parameters

Name	Required	Description
hwnd	Yes	A handle to the GUI object whose window procedure is to receive the message.
uMsg	Yes	Integer specifying the message to be posted.
wParam	No	Message-specific integer value. Defaults to 0.
lParam	No	Message-specific integer value. Defaults to 0.

Returns

TRUE$ if the message was posted successfully, or FALSE$ otherwise.

Remarks

Equates for the common window messages can be found the MSWin_WindowMessage_Equates insert record.

This method uses the Windows API PostMessage function internally – for further information please see the Microsoft website.

Example

```
// Post a WM_CLOSE message to a form (basically the same using the CLOSE method)
$Insert MSWin_WindowMessage_Equates

hwnd   = Get_Property( "MYWIN", "HANDLE" )
uMsg   = WM_CLOSE$
wParam = 0
lParam = 0

RetVal = Exec_Method( "SYSTEM", "POSTWINMSG", hwnd, uMsg, wParam, lParam )
```

See also

SYSTEM PROCESSWINMSGS method, SYSTEM SENDWINMSG method, Common GUI POSTWINMSG method, QUALIFYWINEVENT method, Common GUI SENDWINMSG method, WINMSG event.

PROCESSEVENTS method

Description

This method enables the Presentation Server to check and execute all pending messages in its event queue, returning when the queue is empty. This allows the system to remain responsive during a long running process.

Syntax

```
Call Exec_Method( "SYSTEM", "PROCESSEBNST", SaveVars )
```

Parameters

Name	Required	Description
SaveVars	No	If TRUE$ (1) then the following system variables will be saved and then restored after the events are processed: • @Dict • @Record • @ID • @RecCount • @Rn_Counter The state of Select Cursor 0 will also be saved (via the Push_Select/Pop_Select functions)

Returns

N/a.

Remarks

This method is basically a wrapper around the Yield stored procedure, and should be called during a long-running process for the following reasons:

1. It allows screen updates, so that changes to controls can be displayed.
2. It allows other events to fire so that the user interface remains responsive.
3. It allows Windows to determine that the process is not "dead" so it will refrain from displaying a "ghost" window with the "Not Responding" caption.

Example

```
$Insert Logical

Ctr = 0
Eof = FALSE$

Loop
   ReadNext ID Else Eof = TRUE$
Until Eof

   Ctr += 1

   // Update the progress bar
   Call Set_Property_Only( @Window : ".PRB_GASGAUGE", "VALUE", Ctr )

   // Allow the progress bar to update and save the
   // select state while we do this
   Call Exec_Method( "SYSTEM", "PROCESSEVENTS", TRUE$ )

   // Check that the window is still up because the user
   //  could have closed it during the PROCESSEVENTS call

While ( Get_Property( @Window, "HANDLE" ) )

   // Do processing etc ...

Repeat
```

See also

SYSTEM WINDOWGHOSTING property, SYSTEM POSTWINMSG method, SYSTEM PROCESSWINMSGS method, SYSTEM SENDWINMSG method, Common GUI POSTWINMSG method, QUALIFYWINEVENT method, Common GUI SENDWINMSG method, WINMSG event, Yield stored procedure.

PROCESSWINMSGS method

Description

This method uses the Windows API PeekMessage function to check the Windows message queue associated with the PS thread, optionally processing the messages and/or within a given range.

Syntax

```
MsgCount = Exec_Method( "SYSTEM", "PROCESSWINMSGS", CheckOnly, FilterMin, |
                                                    FilterMax )
```

Parameters

Name	Required	Description
CheckOnly	No	If TRUE$ then the message queue is checked but no messages are actually processed. If FALSE$ then all messages within the specified range are processed before the method returns.
FilterMin	No	Integer specifying the value of the first message in the range to be processed. Defaults to 0.
FilterMax	No	Integer specifying the value of the last message in the range to be processed. Defaults to 0 if CheckOnly is FALSE$, otherwise it defaults to 33768.

Returns

The number of messages processed or checked. A negative value indicates that a WM_QUIT message was found.

Remarks

This method can be used in place of the Yield() stored procedure (or SYSTEM PROCESSEVENTS method) to allow the PS to check its message queue without executing any pending OpenInsight events. This is useful when you wish to prevent the "Window Ghosting" effect (i.e. the "Not Responding" window state), without processing any events, which might cause them to execute out of the expected sequence.

For example, when a form is created the CREATE event is queued first, followed by the SIZE event. If the CREATE event calls the Yield() stored procedure then the SIZE event would be executed before the CREATE event has finished, which might cause problems.

Equates for the common window messages can be found the MSWin_WindowMessage_Equates insert record.

This method uses the Windows API PeekMessage function internally – for further information please see the Microsoft website.

Example

```
$Insert MSWin_WindowMessage_Equates
$Insert Logical

// Example: Check to see if the PS has any messages pending but don't
//          process them
MsgCount = Exec_Method( "SYSTEM", "PROCESSWINMSGS", TRUE$ )

// Example: Process all Windows Messages but do not process any queued
//          OpenInsight events
MsgCount = Exec_Method( "SYSTEM", "PROCESSWINMSGS", FALSE$ )

// Example: Process all WM_PAINT Messages - this will not process any
//          queued OpenInsight events
MsgCount = Exec_Method( "SYSTEM", "PROCESSWINMSGS", FALSE$, |
                                                    WM_PAINT$, WM_PAINT$ )

// Example: Process all Keyboard Messages - this will not process any
//          queued OpenInsight events
MsgCount = Exec_Method( "SYSTEM", "PROCESSWINMSGS", FALSE$, |
                                                    WM_KEYFIRST$, WM_KEYLAST$ )
```

See also

SYSTEM WINDOWGHOSTING property, SYSTEM POSTWINMSG method, SYSTEM PROCESSEVENTS method, SYSTEM SENDWINMSG method, Common GUI POSTWINMSG method, QUALIFYWINEVENT method, Common GUI SENDWINMSG method, WINMSG event, Yield stored procedure.

REFLECTEVENTS method

Description

Returns published event type information for a specified object type.

Syntax

```
EventInfo = Exec_Method( "SYSTEM", "REFLECTEVENTS", TypeName )
```

Parameters

Name	Required	Description
TypeName	Yes	Specifies the object type to return the information for.

Returns

An @Fm-delimited array of event type information. Each field in the array represents an event with the following @Vm-delimited structure:

```
<0,1> Event name (e.g. "BUTTONDOWN")
<0,2> Event display name (e.g. "ButtonDown" )
<0,3> Event parameter names (@Svm-delimited)
<0,4> Event parameter types (@Svm-delimited)
<0,5> Event parameter flags (@Svm-delimited)
<0,6> Event Category
<0,7> Event Flags
<0,8> Event Description

(Values 3,4 and 5 form an AMV group).
```

Remarks

Equated constants for use with this method are defined in the PS_EVENT_TYPEINFO_EQUATES insert record.

Example

```
// Get the event type information for the EDITTABLE type

TypeInfo = Exec_Method( "SYSTEM", "REFLECTEVENTS", "EDITTABLE" )
```

See also

SYSTEM TYPES property, SYSTEM REFLECTMETHODS method, SYSTEM REFLECTPROPERTIES method, Appendix F - Event Type Information.

REFLECTMETHODS method

Description

Returns published method type information for a specified object type.

Syntax

```
MethodInfo = Exec_Method( "SYSTEM", "REFLECTMETHODS", TypeName )
```

Parameters

Name	Required	Description
TypeName	Yes	Specifies the object type to return the information for.

Returns

An @Fm-delimited array of method type information. Each field in the array represents a method with the following @Vm-delimited structure:

```
<0,1> Method name (e.g. "GETPARENTFORM")
<0,2> Method display name (e.g. "GetParentForm" )
<0,3> Method parameter names (@Svm-delimited)
<0,4> Method parameter types (@Svm-delimited)
<0,5> Method parameter flags (@Svm-delimited)
<0,6> Method return value name
<0,7> Method return value type
<0,8> Method Description

(Values 3,4 and 5 form an AMV group).
```

Remarks

Equated constants for use with this method are defined in the PS_METHOD_TYPEINFO_EQUATES insert record.

Example

```
// Get the method type information for the LISTBOX type

TypeInfo = Exec_Method( "SYSTEM", "REFLECTMETHODS", "LISTBOX" )
```

See also

SYSTEM TYPES property, SYSTEM REFLECTEVENTS method, SYSTEM REFLECTPROPERTIES method, Appendix G - Method Type Information.

REFLECTPROPERTIES method

Description

Returns published property type information for a specified object type.

Syntax

```
PropertyInfo = Exec_Method( "SYSTEM", "REFLECTPROPERTIES", TypeName )
```

Parameters

Name	Required	Description
TypeName	Yes	Specifies the object type to return the information for.

Returns

An @Fm-delimited array of method type information. Each field in the array represents a method with the following @Vm-delimited structure:

```
<0,1> Property name (e.g. "PARENTFORM")
<0,2> Property display name (e.g. "ParentForm" )
<0,3> Property type
<0,4> Property flags
<0,5> Property category
<0,6> Property style
<0,7> Property description
```

Remarks

Equated constants for use with this method are defined in the PS_PROPERTY_TYPEINFO_EQUATES insert record.

Example

```
// Get the property type information for the PUSHBUTTON type

TypeInfo = Exec_Method( "SYSTEM", "REFLECTPROPERTIES", "PUSHBUTTON" )
```

See also

SYSTEM TYPES property, SYSTEM REFLECTEVENTS method, SYSTEM REFLECTMETHODS method, Appendix H - Property Type Information.

RESTART method

Description

Restarts the Presentation Server using the current command line, or a new set of parameters.

Syntax

```
Restarted = Exec_Method( "SYSTEM", "RESTART", UseCurrentArgs, NewArgs)
```

Parameters

Name	Required	Description
UseCurrentArgs	No	Boolean value – if TRUE$ then use the command line arguments passed to the current application.
NewArgs	No	If UseCurrentArgs is FALSE$ (or null) then this parameter is a string containing the new parameters to be passed to the restarted PS instance.

Returns

A boolean value of TRUE$ if the PS was restarted, or FALSE$ otherwise.

Remarks

N/a.

Example

```
// Restart the PS with  new command line arguments

CmdArgs = "/AP=EXAMPLES /UN=EXAMPLES /DV=1 /SM=1 /NS=1"

IsOK = Exec_Method( "SYSTEM", "RESTART", CmdArgs )
```

See also

SYSTEM CMDLINE property, SYSTEM LOGININFO property, SYSTEM RUNWIN method, SYSTEM SHELLEXEC method, SYSTEM INITIALIZE event, SYSTEM LOGIN event, Starting the Presentation Server chapter.

RUNHELP method

Description

Displays a Windows Compiled HTMLHelp (CHM) file or an old-style WinHelp (HLP) file.

Syntax

```
RetVal = Exec_Method( "SYSTEM", "RUNHELP", HelpFile, HelpParams)
```

Parameters

Name	Required	Description
HelpFile	Yes	Help file name and path to display.
HelpParams	No	An @Fm-delimited array of options: <1> OwnerForm <2> HelpCommand (see Remarks) <3> HelpParam <4> HTMLHelp flag – if TRUE$ then HelpFile is a CHM File, otherwise it is an HLP file.

Returns

If loading a HTMLHelp (CHM file) the handle of the HTMLHelp form is returned, or 0 if there is an error (A value of TRUE$ is always returned for the HH_CLOSE_ALL command).

If loading a WinHelp (HLP file) the return value is boolean – TRUE$ if the help file was loaded successfully, or FALSE$ otherwise.

Remarks

Supported commands for HTMLHelp are:

- HH_DISPLAY_INDEX (Index Tab)
- HH_DISPLAY_SEARCH (Search Tab)
- HH_HELP_CONTEXT (Display mapped numeric value in Help Param)
- HH_KEYWORD_LOOKUP (Keyword Search (for TopicSearch))
- HH_CLOSE_ALL (Close all Help windows)
- HH_DISPLAY_TOC

Supported commands for WinHelp are:

- HELP_KEY
- HELP_PARTIALKEY
- HELP_COMMAND
- HELP_CONTENTS

Please be aware that the WinHelp file format has been deprecated for some time and is no longer supported by Microsoft. HTMLHelp (v1.4) still has support but development was abandoned by Microsoft in 2012. Most current help systems have moved to online HTML.

Equated constants for used with HTMLHelp files may be found in the MSWIN_HTML_EQUATES insert record. Constants for use with the RUNHELP method may be found in the PS_SYSTEM_EQUATES insert record.

This method uses the Windows HTMLHelp and WinHelp WinAPI functions internally so please see the Microsoft website for more details.

Example

```
// Example - load the old OpenInsight PROGREF.CHM file passing it a word to find.
$Insert MSWin_HTMLHelp_Equates
$Insert PS_System_Equates
$Insert Logical

Word = "utility"

HelpParams = ""
HelpParams<PS_RHP_POS_OWNERFORM$> = @Window
HelpParams<PS_RHP_POS_HELPCMD$>   = HH_KEYWORD_LOOKUP$
HelpParams<PS_RHP_POS_HELPPARAM$> = Word
HelpParams<PS_RHP_POS_HTMLHELP$>  = TRUE$

HwndHH = Exec_Method( "SYSTEM", "RUNHELP", "ProgRef.chm", HelpParams )
```

See also

N/a.

RUNWIN method

Description

Launches a specified application.

Syntax

```
RetVal = Exec_Method( "SYSTEM", "RUNWIN", CmdLine, RunParams )
```

Parameters

Name	Required	Description
CmdLine	Yes	Name and path of the file to execute along with any necessary command line arguments.
RunParams	No	@Fm-delimited array of options: <1> ShowCmd value – determines how the application is launched and displayed (see Remarks). <2> Callback stored procedure. <3> Callback stored procedure argument. <4> Working directory to use. If null the current Working directory is used.

Returns

An @Fm-delimited array of process information:

```
<1> Exit Code (if the application was launched in modal fashion).
<2> Instance Handle
<3> Thread ID
<4> Process ID
```

Remarks

The ShowCmd value in the RunParams parameter determines how the new application is displayed when it is executed. It may be one of the following numeric values (defaults to 1):

Value	Name	Description
0	SW_HIDE	Hides the application.
1	SW_SHOWNORMAL	Shows the application.
2	SW_SHOWMINIMIZED	Minimizes the application and activates it.
3	SW_SHOWMAXIMIZED	Maximizes the application and activates it.
4	SW_SHOWNOACTIVATE	Shows the application but doesn't activate it.
5	SW_SHOW	Same as SW_SHOWNORMAL.
6	SW_MINIMIZE	Minimizes the application but doesn't activate it.
7	SW_SHOWMINNOACTIVE	Same as SW_MINIMIZE.
8	SW_SHOWNA	Same as SHOWNOACTIVATE.
9	SW_RESTORE	Same as SW_SHOWNORMAL.

(These correspond to the same values as per the Windows API ShowWindow function).

If the ShowCmd value is prefixed with a "-" symbol then the new application is launched in modal fashion, which means OpenInsight waits for the new application to close before it returns from the RUNWIN method.

e.g. Launch modal and minimized use "-2"
Launch modal and hidden use "-0"

If the new application is not launched in a modal fashion OpenInsight can be notified when it is terminated by passing the name of a stored procedure in field 2 of the RunParams parameter. This "callback" stored procedure will be executed when the new application is closed. It should take at least one argument which will be passed the value from field 3 of the RunParams parameter.

Equated constants for used with this method may be found in the PS_SYSTEM_EQUATES insert record. Equated constants for the ShowCmd value may be found in the MSWIN_SHOWWINDOW_EQUATES insert record.

This method uses the Windows CreateProcess function internally so please see the Microsoft website for more details on how new applications are launched.

Example

```
// Example - load the 7-Zip program - minimized, and wait for it to return
$Insert MSWin_ShowWindow_Equates
$Insert PS_System_Equates
$Insert Logical

CmdLine =  Quote( "C:\Program Files\7-Zip\7z.exe" )
CmdLine := " a -r -mx9 -mmt=on mystuff.7z @backup_include.txt"

RunParams = ""
RunParams<PS_RWP_POS_SHOWCMD$>    = "-" : SW_SHOWMINNOACTIVE$
RunParams<PS_RWP_POS_WORKINGDIR$> = "C:\MyStuff"

RetVal = Exec_Method( "SYSTEM", "RUNWIN", CmdLine, RunParams )

// Example - load the 7-Zip program - minimized, with a callback to let OI
// know when it's finished
CmdLine =  Quote( "C:\Program Files\7-Zip\7z.exe" )
CmdLine := " a -r -mx9 -mmt=on stuff.7z @backup_include.txt"

RunParams = ""
RunParams<PS_RWP_POS_SHOWCMD$>          = SW_SHOWMINNOACTIVE$
RunParams<PS_RWP_POS_CALLBACKPROC$>     = "MY_7_ZIP_CALLBACK_PROC"
RunParams<PS_RWP_POS_CALLBACKPROCARG$> = "Backed up MyStuff folder"
RunParams<PS_RWP_POS_WORKINGDIR$>       = "C:\MyStuff"

RetVal = Exec_Method( "SYSTEM", "RUNWIN", CmdLine, RunParams )
```

See also

SYSTEM EXITCODE property, WINDOW VISIBLE property, SYSTEM FINDEXE method, SYSTEM SHELLEXEC method, SYSTEM TASKEXIT event, RTI_RUN_COMMAND stored procedure.

SENDWINMSG Method

Description

This method uses the Windows API SendMessage function to call the window procedure of the target object directly and does not return until the window procedure has processed the message.

Syntax

```
RetVal = Exec_Method( "SYSTEM", "SENDWINMSG", hwnd, uMsg, wParam, lParam )
```

Parameters

Name	Required	Description
hwnd	Yes	A handle to the GUI object whose window procedure is to receive the message.
uMsg	Yes	Integer specifying the message to be sent.
wParam	No	Message-specific integer value. Defaults to 0.
lParam	No	Message-specific integer value. Defaults to 0.

Returns

An integer value specifying the result of the processing – this depends on the message sent.

Remarks

While it is possible to use the Windows API SendMessage call with a PS object directly from Basic+, this may not be ideal in all circumstances as some window messages are sensitive to the thread that they are executed from. In this case the SENDWINMSG event should be used as it will send the message from the PS thread instead.

Equates for the common window messages can be found the MSWin_WindowMessage_Equates insert record.

This method uses the Windows API SendMessage function internally – for further information please see the MSDN website.

Example

```
// Send a WM_CLOSE message to a form (this is basically the same as
// calling its CLOSE method).

$Insert MSWin_WindowMessage_Equates

hwnd   = Get_Property( "MYWIN", "HANDLE" )
uMsg   = WM_CLOSE$
wParam = 0
lParam = 0

RetVal = Exec_Method( "SYSTEM", "SENDWINMSG", hwnd, uMsg, wParam, lParam )
```

See also

SYSTEM POSTWINMSG method, SYSTEM PROCESSWINMSGS method, Common GUI POSTWINMSG method, QUALIFYWINEVENT method, Common GUI SENDWINMSG method, Common GUI WINMSG event.

SETCURSOR method

Description

Sets the cursor image to the specified image.

Syntax

```
CurrentCursor = Exec_Method( "SYSTEM", "SETCURSOR", NewCursor )
```

Parameters

Name	Required	Description
NewCursor	No	Specifies the new cursor to display. The format of the parameter is the same as the SYSTEM CURSOR property, i.e. • A path and file name of a cursor (.CUR) file. • A path and file name to a resource file (such as a DLL) containing the cursor image, along with its resource ID. • A symbol that specifies one of the standard Windows cursors. This value can also be NULL$ to let Windows display the default cursor.

Returns

The current cursor value if set, or NULL$ otherwise.

Remarks

Using this property to set the cursor changes it immediately, but this change is not persistent, i.e. the next time a control or window receives a WM_SETCURSOR message the cursor will revert to the default shape. If you wish to use a persistent value that is preserved across WM_SETCURSOR messages then use the CURSOR property instead.

Example

```
// Set the cursor to an hourglass

PrevCursor = Exec_Method( "SYSTEM", "SETCURSOR", "H" )
```

See also

Common GUI CURSOR property, SYSTEM CURSOR property.

SETENVVAR method

Description

This method sets the value of a Windows environment variable.

Syntax

```
SuccessFlag = Exec_Method( "SYSTEM", "SETENVVAR", EnvVarName, NewValue )
```

Parameters

Name	Required	Description
EnvVarValue	Yes	Name of the environment variable to update.
NewValue	No	New value for the variable. Defaults to null.

Returns

TRUE$ if the variable was set successfully, or FALSE$ otherwise.

Remarks

This method is a simple wrapper around the Windows API SetEnvironmentVariable function - further information regarding Windows environment variables may be found on the MSDN website.

Example

```
// Set the value of the Windows "MyVar" variable

SuccessFlag = Exec_Method( "SYSTEM", "SETENVVAR", "MyVar", "SomeNewValue" )
```

See also

SYSTEM GETENVVAR method, SYSTEM ENVVARLIST property

SHELLEXEC method

Description

Launches an application via an associated document name. For example, when passed the name of a Word document file this method can launch the Word application and open the specified document.

Syntax

```
RetVal = Exec_Method( "SYSTEM", "SHELLEXEC", OwnerForm, Operation, File, |
                      Parameters, WorkingDir, ShowCmd )
```

Parameters

Name	Required	Description
OwnerForm	No	Name of an OpenInsight form to use as a parent for displaying UI or error messages.

Operation	No	Specifies the operation to be performed on the file. Common operations are:
		
		`"edit"`
		`"explore"`
		`"find"`
		`"open"`
		`"print"`
		`"runas"`
		
		If not specified then the default operation for the file type is used. If this is not available then "open" is used. If neither are available then the system uses the first verb listed in the registry.

File	Yes	Specifies the file to perform the operation on. This can be the name of a document file or an executable file. Note that not all file types support all operations.
Parameters	No	If File specifies an executable file then this parameter should specify the command line parameters to pass to it. If File specifies a document file this parameter should be null.
WorkingDir	No	Specifies the default working directory for the operation. If null the current working directory is used.
ShowCmd	No	Determines how an application is displayed when it is opened (See remarks)

Returns

Returns a numeric value – if greater than 32 the method has succeeded, otherwise the returned value indicates an error:

Value	Name	Description
0	N/a.	The OS is out of resources.
2	SE_ERR_FNF	The specified file was not found.
3	SE_ERR_PNF	The specified path was not found.
5	SE_ERR_ACCESSDENIED	The operating system denied access to the specified file.
8	SE_ERR_OOM	There was not enough memory to complete the operation.
11	ERROR_BAD_FORMAT	The .exe file is invalid (non-Win32 .exe or error in .exe image).
26	SE_ERR_SHARE	A sharing violation occurred.
27	SE_ERR_ASSOCINCOMPLETE	The file name association is incomplete or invalid.
28	SE_ERR_DDETIMEOUT	The DDE transaction could not be completed because the request timed out.
29	SE_ERR_DDEFAIL	The DDE transaction failed.
30	SE_ERR_DDEBUSY	The DDE transaction could not be completed because other DDE transactions were being processed.
31	SE_ERR_NOASSOC	There is no application associated with the given file name extension. This error will also be returned if you attempt to print a file that is not printable.
32	SE_ERR_DLLNOTFOUND	The specified DLL was not found.

Remarks

The ShowCmd parameter determines how the new application is displayed when it is executed. It may be one of the following numeric values (defaults to 1):

Value	Name	Description
0	SW_HIDE	Hides the application.
1	SW_SHOWNORMAL	Shows the application.
2	SW_SHOWMINIMIZED	Minimizes the application and activates it.
3	SW_SHOWMAXIMIZED	Maximizes the application and activates it.
4	SW_SHOWNOACTIVATE	Shows the application but doesn't activate it.
5	SW_SHOW	Same as SW_SHOWNORMAL.
6	SW_MINIMIZE	Minimizes the application but doesn't activate it.
7	SW_SHOWMINNOACTIVE	Same as MINIMIZE.
8	SW_SHOWNA	Same as SHOWNOACTIVATE.
9	SW_RESTORE	Same as SW_SHOWNORMAL.

Equated constants for the ShowCmd value may be found in the MSWIN_SHOWWINDOW_EQUATES insert record. Equates constants for the SHELLEXEC return value may be found in the MSWIN_SHELLEXECUTE_EQUATES insert record.

This method uses the Windows ShellExecute function internally so please see the Microsoft website for more details on how applications and documents are launched.

Example

```
// Example - Open an Excel spreadsheet in the "My Documents" folder
$Insert MSWin_ShowWindow_Equates
$Insert MSWin_ShellExecute_Equates
$Insert Logical

MyDocsDir = Exec_Method( "FILESYSTEM", "GETSPECIALDIR", "PERSONAL" )
SEVal     = Exec_Method( "SYSTEM", "SHELLEXEC", @Window, "open",      |
                         "Cash.xls", "", MyDocsDir, SW_SHOWMAXIMIZED$ )

if ( SEVal <= SE_ERR_DLLNOTFOUND$ ) then
   // Handle Error ...
End

// Example - Print a Word document on the user's DeskTop
DesktopDir = Exec_Method( "FILESYSTEM", "GETSPECIALDIR", "DESKTOPDIRECTORY" )
SEVal      = Exec_Method( "SYSTEM", "SHELLEXEC", @Window, "print",     |
                          "Plans.docx", "", DeskTopDir, SW_SHOWNORMAL$ )

if ( SEVal <= SE_ERR_DLLNOTFOUND$ ) then
   // Handle Error ...
End
```

See also

WINDOW VISIBLE property, SYSTEM FINDEXE method, SYSTEM RUNWIN method, RTI_RUN_COMMAND stored procedure.

SHOWMESSAGE method

Description

Displays a non-owned message box (i.e. a message box with no parent form).

Syntax

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

Parameters

Name	Required	Description
MessageStruct	Maybe	This is an @Fm-delimited array that contains a message structure as per the Msg() stored procedure. Required if MessageName is null.
MessageName	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.

Example

```
// Example - display a simple non-owned message
$Insert Msg_Equates
MsgStruct             = ""
MsgStruct<MTEXT$>     = "Nobody owns me"
MsgStruct<MICON$>     = "*"
MsgStruct<MJUST$>     = "C"
MsgStruct<MCAPTION$> = "SYSTEM Message Box"

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

// Example display a non-owned message with an entity name
MsgVal = Exec_Method( "SYSTEM", "SHOWMESSAGE", "", "OI_ABOUT" )
```

See also

WINDOW SHOWMESSAGE method, Msg stored procedure.

SHOWPOPUP method

Description

Displays a non-owned popup box (i.e. a popup box with no parent form).

Syntax

```
PopupValue = Exec_Method( "SYSTEM", "SHOWPOPUP", PopupStruct, PopupName )
```

Parameters

Name	Required	Description
PopupStruct	Maybe	This is an @Fm-delimited array that contains a popup box structure as per the Popup() stored procedure. Required if PopupName is null.
PopupName	Maybe	Contains the name of an OpenInsight repository POPUP entity to display. The fields in the PopupStruct parameter override the fields from the stored entity. Required if PopupStruct is null.

Returns

The value as specified by the popup definition.

Remarks

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

Example

```
// Example - display a simple non-owned popup
$Insert Popup_Equates
PopupStruct             = ""
PopupStruct<PMODE$>     = "L"
PopupStruct<PFORMAT$>   = 1 : @svm : 10 : @vm : "C" : @svm : "C" : |
                          @svm : "B" : @svm : "Value"
PopupStruct<PDISPLAY$> = TRUE$ : @vm : FALSE$
PopupStruct<PTITLE$>    = "Simple Question"
PopupStruct<PTYPE$>     = "F"
PopupStryct<PFIELD$>    = 1

PopupVal = Exec_Method( "SYSTEM", "SHOWPOPUP", PopupStruct )

// Example display a non-owned popup with an entity name
PopupVal = Exec_Method( "SYSTEM", "SHOWPOPUP", "", "CUSTOMERS" )
```

See also

WINDOW SHOWPOPUP method, Popup stored procedure.

STARTFORM method

Description

Executes a standalone form (i.e. the form is launched without a specified parent form so it has no owner).

Syntax

```
FormInstanceID = Exec_Method( "SYSTEM", "STARTFORM", FormName, CreateParam )
```

Parameters

Name	Required	Description
FormName	Yes	Name of the form to start.
CreateParam	No	Parameter to pass to the form's CREATE event.

Returns

The instance name of the newly created form if successful, otherwise null is returned.

Remarks

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

Example

```
// Example - start the APP_MAIN form as a standalone form with no owner

CreateParam = "X1234"
FormID = Exec_Method( "SYSTEM", "STARTFORM", "APP_MAIN", CreateParam )
```

See also

WINDOW SHOWFORM method, Start_Window stored procedure.

TEXTRECT method

Description

Calculates the width and height needed to display a specified string.

Syntax

```
TextRect = Exec_Method( "SYSTEM", "TEXTRECT", ObjectID, TextParams )
```

Parameters

Name	Required	Description
ObjectID	No	Fully qualified ID of a form or control. If passed the calculation uses the FONT and CLIENTSIZE of the object for the calculation.

TextParams	Yes	An @Fm-delimited array specifying the parameters for the calculation.
		
		`<1> Text for the calculation (CRLF delimited for multiple lines) – Required`
		
		`<2> DrawText flags. A bitmask integer containing flags that control how the text should be drawn – Optional`
		
		`<3> Maximum width of the rectangle (only used if ObjectID is not specified) – Optional`
		
		`<4> Font (@Svm delimited) to use for the calculation rectangle (only used if ObjectID is not specified) – Optional`

Returns

The @fm-delimited width and height of the calculated rectangle:

```
<1> Width
<2> Height
```

If ObjectID was passed the returned values will be scaled according to its SCALEUNITS property. If an object is not passed then the Maximum Width and Font fields in the TextParams parameter are assumed to be in PIXEL coordinates. The caller is responsible for scaling them as necessary.

Remarks

Constants for use with the TextParams argument can be found in the PS_SYSTEM_EQUATES insert record.

This method uses the Windows API DrawText function internally to perform the calculation - further information regarding DrawText may be found on the Microsoft website. Constants for the DrawText flags can be found in the MSWIN_DRAWTEXT_EQUATES (The DT_CALCRECT$ bit is always set internally by this method so there is no need to specify it yourself).

Example

```
// Example - calculate the size of the area needed to display
// a multiline text string on a STATIC text control

$Insert PS_System_Equates
$Insert MSWin_DrawText_Equates
$Insert RTI_Text_Equates

Text = "Line 1"  : CRLF$ : |
     : "Line 2, and more text on the end"

ObjectID   = @window : ".TXT_FOO"
TextParams = ""

TextParams<PS_TP_POS_TEXT$>    = Text
TextParams<PS_TP_POS_DTFLAGS$> = DT_WORDBREAK$

TextRect = Exec_Method( "SYSTEM", ObjectID, TextParams )
```

See also

Common GUI FONT property.

TRANSLATEKEYDOWN method

Description

Synthesizes and sends a WM_KEYDOWN or WM_SYSKEYDOWN window message to the PS message dispatch handler to see if it matches a menu accelerator. If so a MENU event is raised for the matched menu item.

Syntax

```
Translated = Exec_Method( "SYSTEM", "TRANSLATEKEYDOWN", ObjectID,    |
                          VirtualKeyCode, KeyFlags, SysKey, KeyState )
```

Parameters

Name	Required	Description
ObjectID	Yes	Fully qualified ID of a form or control to send the message to.
VirtualKeyCode	Yes	Virtual key code to send.
KeyFlags	No	Key flags (Repeat count, scan code, extended-key flag, context code, previous key-state flag, and transition-state flag).
SysKey	No	Boolean – if TRUE$ then synthesize a WM_SYSKEYDOWN message rather than a WM_KEYDOWN message. Default is FALSE$.

KeyState	No	A bitmask of flags denoting which system keys are pressed down:
		
		`0x0100 – Shift key is down`
		`0x0200 – Control key is down`
		`0x0400 – Alt key is down`
		
		The default is 0.

Returns

Returns TRUE$ if the keydown message was found and translated, or FALSE$ otherwise.

Remarks

This is a low-level method designed to test and fire keystrokes at a form's main menu from a control that "eats" these keystrokes normally before the PS can get them (this can happen with certain OLE browser controls for example).

Virtual key codes are numeric constants Windows uses to identify keystrokes. Equated constants for these can be found in the MSWIN_VIRTUAL_EQUATES insert record.

This method uses the Windows WM_KEYDOWN and WM_SYSKEYDOWN window messages internally - further information regarding these may be found on the Microsoft website.

Example

```
// Example - Fire a WM_KEYDOWN Ctrl+S message to the current form's main menu to
// see if it triggers a menu event
$Insert MSWin_VirtualKey_Equates
$Insert Logical

KeyState = BitOr( 0, 0x200 ) ; // CtrlKey down
MenuTriggered = Exec_Method( "SYSTEM",            |
                             "TRANSLATEKEYDOWN", |
                             @Window,            |
                             VK_S$,              |
                             0,                  |
                             FALSE$,             |
                             KeyState )
```

See also

Common GUI CHAR event.

SYSTEM Events

The SYSTEM object supports the following events:

Name	Description
FINALIZE	Occurs when the application is shutting down.
IDLEPROC	Occurs when the system is executing an "idle procedure" from the IDPLEPROC queue.
INITIALIZE	Occurs when the application is being loaded.
LOGIN	Occurs when a user is logging into an application.
TASKEXIT	Occurs when a closed RUNWIN application executes a callback procedure.

FINALIZE event

Description

Occurs when the application is shutting down.

Syntax

```
bForward = FINALIZE( CtrlEntID, CtrlClassID, bSysShutdown )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of control receiving the event.
bSysShutdown	Boolean value – if TRUE$ then OpenInsight is closing, otherwise this is FALSE to signify that the application is changing.

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 FINALIZE performs the following tasks:

- Processes registered system shutdown tasks
- Processes the shutdown procedure set in the database environment record.

This event may be intercepted by creating a promoted event handler in your own application, however care should be taken when doing so as it may make your system unstable if you prevent the system level event handler from executing properly.

Example

N/a.

See Also

SYSTEM INITALIZE event, SYSTEM LOGIN event.

IDLEPROC event

Description

Occurs when the system is executing an "idle procedure" from the IDPLEPROC queue.

Syntax

```
bForward = IDLEPROC( CtrlEntID, CtrlClassID, ProcID, ProcParam )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of control receiving the event.
ProcID	Specifies the name of the stored procedure to be executed.
ProcParam	Contains the parameter to be passed the stored procedure.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

The system-level event handler for IDLEPROC performs the following tasks:

- Verifies that ProcID is a valid stored procedure and then executes it.

This event may be intercepted and overridden by creating a promoted event handler in your own application.

Example

```
Function IDLEPROC( CtrlEntID, CtrlClassID, ProcID, ProcParam )

   // Example - update the System Monitor when an IDLEPROC is executed
   // (Poor man's logging :) )

   LogText = TimeDate() : " - IDLEPROC calling " : ProcID : "(" : Quote( ProcParam ) : ")"
   Call Exec_Method( "SYSTEMMONITOR", "OUTPUT", LogText )

Return TRUE$ ; // Let the system run it...
```

See Also

SYSTEM IDLEPROC property, SYSTEM IDLEPROCQUEUE property, SYSTEM ADDIDLEPROC event.

INITIALIZE event

Description

Occurs when an application is being loaded.

Syntax

```
bForward = INITIALIZE( CtrlEntID, CtrlClassID, dwInitFlags)
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of control receiving the event.

dwInitFlags	Integer bitmask containing a set of initialization flags:
	
	`0x0001 – Primary Boot Flag`
	`0x0008 – Start IDE with clean workspace`
	`0x0010 – Start IDE with default workspace`

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 INITIALIZE performs the following tasks:

- Refreshes all configuration information.
- Initializes the type information system.
- Runs an "AutoExec" request if needed.
- Sets application-specific visual style attributes.
- Loads the IDE or the Application Entry Point form as appropriate.
- Starts the OpenInsight "chat window" if needed.

This event may be intercepted by creating a promoted event handler in your own application, however care should be taken when doing so as it may make your system un-bootable if you prevent the system level event handler from executing properly.

Example

N/a.

See Also

SYSTEM FINALIZE event, SYSTEM LOGIN event.

LOGIN event

Description

Occurs when the application processes user login requests.

Syntax

```
bForward = LOGIN( CtrlEntID, CtrlClassID, bPrimaryBoot, AppID, UserName
                  Password )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of control receiving the event.
bPrimaryBoot	Boolean value – if TRUE$ then this is the first login attempt.
AppID	Contains the name of the application to log into.
Username	Contains the name of user logging in.
Password	Contains the password used for the login attempt.

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 LOGIN performs the following tasks:

- Authenticates the passed credential information.

This event may be intercepted by creating a promoted event handler in your own application, however care should be taken when doing so as it may make your system inaccessible if you prevent the system level event handler from executing properly.

Example

N/a.

See Also

SYSTEM INITALIZE event, SYSTEM FINALIZE event.

TASKEXIT event

Description

Occurs when an application that was launched via the RUNWIN method is closed and executes a callback procedure.

Syntax

```
bForward = TASKEXIT( CtrlEntID, CtrlClassID, ProcID, ProcParam )
```

Parameters

Name	Description
CtrlEntID	Fully qualified name of the object receiving the event.
CtrlClassID	Type of control receiving the event.
ProcID	Specifies the name of the stored procedure to be executed.
ProcParam	Contains the parameter to be passed the stored procedure.

Returns

TRUE$ or FALSE$. If FALSE$, the program execution returns to the calling procedure. If TRUE$, the event processing goes to the next level.

Remarks

The system-level event handler for TASKEXIT performs the following tasks:

- Verifies that ProcID is a valid stored procedure and then executes it.

This event may be intercepted and overridden by creating a promoted event handler in your own application.

Example

```
Function TASKEXIT( CtrlEntID, CtrlClassID, ProcID, ProcParam )

   // Example - update the System Monitor when TASKEXIT is executed
   // (Poor man's logging :) )

   LogText = TimeDate() : " - TASKEXIT calling " : ProcID : "(" : Quote( ProcParam ) : ")"
   Call Exec_Method( "SYSTEMMONITOR", "OUTPUT", LogText )

Return TRUE$ ; // Let the system run it...
```

See Also

SYSTEM RUNWIN Method.