|  |  |  |
| --- | --- | --- |
| [logo_longhorn](file:///\\team\sites\LHSDK\specs\Forms\Priority%20View.htm) | Record & Playback Framework  APIs for MSAA-based Playback  v0.1 | |
|  | Last updated: 4/7/2016 7:43 AM | |
| **Program Manager** |  | **Usability** | Name |
| **Developer** |  | **Product Planning** | Name |
| **Tester** |  | **Operations** | Name |
| **International** | Name | **Support** | Name |
| **User Assistance** | Name | **Design** | Name |

**MSAA-based playback APIs can be divided into two groups: General and IScreenElement methods. General APIs methods can be called without using any objects that correspond to a UI element. Each interaction is applied to the last UI element found by** FindScreenElement **or** FindScreenElementEx **methods.**

**General APIs**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT InitPlayBack();

**Description:** Initializes MSAA-based playback. Should be called once before any of RPF’s APIs is used. In order to free all the resources allocated by this method call FinishPlayBack before a test finishes. Only one function from the InitPlayBack family should be called in the beginning of a test. If the same or any other function from the InitPlayBack family is called between first InitPlayBack and FinishPlayBack then this second call will fail and will have no effect.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT InitPlayBackForTargetWindow([in] BSTR bstrMainWindowClass, [in] BSTR bstrMainWindowCaption, [in] BOOL bShowInfoWindow);

**Parameters:**

[in] BSTR bstrMainWindowClass – class name of a target window

[in] BSTR bstrMainWindowCaption – caption of a target window

[in] BOOL bShowInfoWindow – flag that defines if RPF’s Info window should be displayed.

**Description:** Initializes MSAA-based playback and attaches to the process that owns the target window. One can use SearchTargetProcessElementsOnly so only UI elements that belong to the target process are searched by RPF. If both bstrMainWindowClass and bstrMainWindowCaption are NULLs or empty strings, no target process is set by RPF. If there is not target process set then calling SearchTargetProcessElementsOnly has no effect and RPF will be always searching for UI elements system-wide.

InitPlayBackForTargetWindow should be called once before any of RPF’s APIs is used. In order to free all the resources allocated by this method call FinishPlayBack before a test finishes. Only one function from the InitPlayBack family should be called in the beginning of a test. If the same or any other function from the InitPlayBack family is called between first InitPlayBack and FinishPlayBack then this second call will fail and will have no effect.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT InitPlayBackForTargetProcess([in] INT nDevenvProcessId, [in] INT

hWndDevenvMainWindow, [in] BOOL bShowInfoWindow);

**Parameters:**

[in] INT nDevenvProcessId – a target process Id

[in] INT hWndDevenvMainWindow – a window handle of a window that belongs to the target process

[in] BOOL bShowInfoWindow – flag that defines if RPF’s Info window should be displayed.

**Description:** Initializes MSAA-based playback and attaches to the target process. One can use SearchTargetProcessElementsOnly so only UI elements that belong to the target process are searched by RPF. If both nDevenvProcessId and hWndDevenvMainWindow are not positive integers, no target process is set by RPF. If there is not target process set then calling SearchTargetProcessElementsOnly has no effect and RPF will be always searching for UI elements system-wide.

InitPlayBackForTargetProcess should be called once before any of RPF’s APIs is used. In order to free all the resources allocated by this method call FinishPlayBack before a test finishes. Only one function from the InitPlayBack family should be called in the beginning of a test. If the same or any other function from the InitPlayBack family is called between first InitPlayBack and FinishPlayBack then this second call will fail and will have no effect.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT InitPlayBackForTargetExecutable([in] BSTR bstrExecutablePath, [in] BOOL bShowInfoWindow);

**Parameters:**

[in] INT bstrExecutablePath – a full path to the target application

process

[in] BOOL bShowInfoWindow – flag that defines if RPF’s Info window should be displayed.

**Description:** Initializes MSAA-based playback, starts target application and attaches to it. One can use SearchTargetProcessElementsOnly so only UI elements that belong to the target process are searched by RPF.

InitPlayBackForTargetExecutable should be called once before any of RPF’s APIs is used. In order to free all the resources allocated by this method call FinishPlayBack before a test finishes. Only one function from the InitPlayBack family should be called in the beginning of a test. If the same or any other function from the InitPlayBack family is called between first InitPlayBack and FinishPlayBack then this second call will fail and will have no effect.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT FinishPlayBack();

**Description:** Releases all the resources allocated by any of InitPlayBack function. Should be called once before a test finishes. If two FinishPlayBack functions are called in a row the second call will fail and will have no effect.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT FindScreenElementEx([in] IScreenElement\* screenElementStart, [in] BSTR bstrQueryId, [in] VARIANT\* resKeys, [in] INT ccResKeys, [out] INT \*elementsFound, [out, retval] IScreenElement\*\* screenElement);

**Parameters:**

[in] IScreenElement\* screenElementStart – a UI element that the search starts from

[in] BSTR bstrQueryId – a QueryId of a UI element that RPF should search for

[in] VARIANT\* resKeys – resource keys that correspond to a strings used in the QueryId

[in] INT ccResKeys – number of resource keys

[out] INT \*elementsFound – total number of found UI elements that match QueryId

[out, retval] IScreenElement\*\* screenElement – first of the found UI elements that match QueryId.

**Description:** Searches for a UI element that is defined by a QueryId. This routine is language neutral when correct resource Keys are used.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT FindScreenElement([in] BSTR bstrQueryId, [out, retval] IScreenElement\*\* screenElement);

**Parameters:**

[in] BSTR bstrQueryId – a QueryId of a UI element that RPF should search for

[out, retval] IScreenElement\*\* screenElement – first of the found UI elements that match QueryId.

**Description:** Searches for a UI element that is defined by a QueryId starting from the desktop. This routine is not language neutral.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT FetchFoundDescendants([out] VARIANT\*\* ppAcc, [out] VARIANT\*\* childId, [out] VARIANT\*\* hWnd, [out] int\* elementsFound);

**Parameters:**

[out] VARIANT\*\* ppAcc – an array of IAccessible\* pointers that correspond to fetched descendants

[out] VARIANT\*\* childId – an array of ChildIds that correspond to fetched descendants

[out] VARIANT\*\* hWnd – an array of window handles that correspond to fetched descendants

[out] int\* elementsFound - total number of descendants

**Description:** Retrieves all UI elements that were found by the last called FindScreenElementEx or FindScreenElement.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT LeftButtonClickBasic([in] INT x, [in] INT y, [in] BOOL bEnsureVisible, [in] BSTR bstrKeyModifiers);

**Parameters:**

[in] INT x – an X coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

[in] INT y – a Y coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

[in] BOOL bEnsureVisible – flag that defines if RPF should verify that the point (x, y) is a valid visible point that belongs to the current UI element. If this parameter is TRUE and the point (x, y) is not clickable then RPF will try to find a clickable point and if such point does not exist then RPF will try to make current UI element visible by scrolling.

[in] BSTR bstrKeyModifiers – keyboard modifiers. Can be any combination of “Ctrl”, “Shift” and “Alt” separated by comma. If NULL or empty string then no keyboard modifiers are applied.

**Description:** Perfroms a mouse left button click on current UI element.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT LeftButtonClick([in] INT x, [in] INT y);

**Parameters:**

[in] INT x – an X coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

[in] INT y – a Y coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

**Description:** Performs a mouse left button click on current UI element. Point defined by (x, y) is just a suggestion for RPF where to click. If this point is not clickable then RPF will try to find a clickable point and if such point does not exist then RPF will try to make current UI element visible by scrolling.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT DoubleClick([in] INT x, [in] INT y);

**Parameters:**

[in] INT x – an X coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

[in] INT y – a Y coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

**Description:** Performs a mouse left button double-click on current UI element. Point defined by (x, y) is just a suggestion for RPF where to click. If this point is not clickable then RPF will try to find a clickable point and if such point does not exist then RPF will try to make current UI element visible by scrolling.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT RightButtonClick([in] INT x, [in] INT y);

**Parameters:**

[in] INT x – an X coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

[in] INT y – a Y coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

**Description:** Performs a mouse right button click on current UI element. Point defined by (x, y) is just a suggestion for RPF where to click. If this point is not clickable then RPF will try to find a clickable point and if such point does not exist then RPF will try to make current UI element visible by scrolling.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT Expand();

**Description:** Expands current UI element if it’s expandable, fails otherwise.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT Collapse();

**Description:** Collapses current UI element if it’s expandable, fails otherwise.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT Check();

**Description:** Checks current UI element.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT Uncheck ();

**Description:** Un-checks current UI element.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

// defaultvalue(0x1) means BY\_SCROLLING

HRESULT EnsureVisible([in, defaultvalue(0x1)] int nEnsureVisibleFlag );

**Parameters:**

int nEnsureVisibleFlag – a flag that defines what mechanism should be used to make the current UI element visible. Can be any OR combination of the following flags:

EVF\_BY\_SCROLLING = 0x00000001 – scrolls the container

EVF\_BY\_EXPANDING\_ANCESTORS = 0x00000002 – expands the ancestors

EVF\_BY\_EXPANDING\_SIBLINGS = 0x00000004 – expands the siblings that have decrementing values

EVF\_BY\_SETTING\_FOCUS = 0x00000008 – sets the focus to the container

**Description:** Tries to make the current UI element visible. This functionality with flag EVF\_BY\_SCROLLING is built-in for each primitive.

Flags EVF\_BY\_EXPANDING\_ANCESTORS and EVF\_BY\_EXPANDING\_SIBLINGS reflect the situation when IAccessible for tree view nodes can be implemented in different ways.  In one case it can preserve parent-child relations and in the other case it (MSAA tree) can be a flat tree where a value reflects the depth of a node. Use EVF\_BY\_SETTING\_FOCUS with care since it can close popup menus.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

// defaultvalue(0x2) means SELFLAG\_TAKESELECTION

HRESULT Select([in, defaultvalue(0x2)] int nSelectionFlag);

**Parameters:**

int nSelectionFlag – a flag that defines what mechanism should be used to select the current UI element. Can be any combination of SELFLAGs. See accSelect and SELFLAG for more details.

**Description:** Selects the current UI element.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

// defaultvalue(0x6) means CF\_USE\_SET\_AS\_EDIT | CF\_LEFT\_DROP\_DOWN\_BUTTON

HRESULT SetValueAsComboBox([in]BSTR newValue, [in, defaultvalue(0x00000006)] INT fSetValueAsComboBoxFlag);

**Parameters:**

BSTR newValue – a value to be set for the current UI element.

INT fSetValueAsComboBoxFlag – flag that defines behavior of SetValueAsComboBox. Can be any OR combination of the following flags:

CF\_LEFT\_DROP\_DOWN\_BUTTON = 0x00000001 – use drop-down button located on the left side of the combo box. This flag has effect only if CF\_USE\_SELECT is set.

CF\_USE\_SELECT = 0x00000002 – drops down a combo box’s list and tryes to select the requested item

CF\_USE\_SET\_AS\_EDIT = 0x00000004 – calls SetValueAsEditBox if selection didn’t work.

**Description:** Sets a new value for the current UI element. First RPF tries to find the new value in the list that belongs to the current UI element and then to select the new value. If this didn’t work RPF calls SetValueAsEditBox.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

// defaultvalue(0x1F) means EF\_DELETE\_CONTENT | EF\_USE\_MSAA | EF\_USE\_WM\_SETTEXT | EF\_USE\_COPY\_PASTE | EF\_USE\_SENDKEYS

HRESULT SetValueAsEditBox ([in]BSTR newValue, [in, defaultvalue(0x0000001F)] INT fSetValueAsEditBoxFlag);

**Parameters:**

BSTR newValue – a value to be set for the current UI element.

INT fSetValueAsComboBoxFlag – flag that defines behavior of SetValueAsEditBox. Can be any OR combination of the following flags:

EF\_DELETE\_CONTENT = 0x00000001 – delete content of the edit box before setting a value

EF\_USE\_MSAA = 0x00000002 – set value via MSAA

EF\_USE\_WM\_SETTEXT = 0x00000004 – set value by sending WM\_SETTEXT message

EF\_USE\_COPY\_PASTE = 0x00000008 – set value via copy/paste

EF\_USE\_SENDKEYS = 0x00000010 – set value by calling SendKeys

**Description:** Sets a new value for the current UI element. First RPF tries to set the new value using IAccessible::put\_accValue. If this didn’t work then RPF tries to set value via copy/paste. If this the latter didn’t work then RPF sends WM\_SETTEXT message. And as the last resort RPF uses SendKeys to set the new value.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT StartDragging([in] INT x, [in] INT y);

**Parameters:**

[in] INT x – an X coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

[in] INT y – a Y coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

**Description:** Moves the mouse pointer to the point defined by (x, y) and pushes left mouse button (without releasing it). Point is just a suggestion for RPF where to set the mouse pointer. If this point is not clickable then RPF will try to find a clickable point and if such point does not exist then RPF will try to make current UI element visible by scrolling.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT StopDragging ([in] INT x, [in] INT y, [in, defaultvalue(-1)] INT nSpeedPixelPerSecond);

**Parameters:**

[in] INT x – an X coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

[in] INT y – a Y coordinate relative to the left-top corner of a rectangle that corresponds to current UI element.

[in, defaultvalue(-1)] INT nSpeedPixelPerSecond – speed at which the mouse moves while performing drag-and-drop

**Description:** Moves the mouse pointer to the point defined by (x, y) and releases left mouse button. Point is just a suggestion for RPF where to set the mouse pointer. If this point is not clickable then RPF will try to find a clickable point and if such point does not exist then RPF will try to make current UI element visible by scrolling.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT WaitForReady();

**Description:** Waits for the current UI element to be ready for an interaction based on the foreground thread or other threads state(s). See SetRobustnessLevel for more details. This function is built-in for each primitive, so a user shouldn’t call it without a specific reason.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT VerifyScreenElement([in]BSTR bstrState, [in] BSTR bstrValue);

**Parameters:**

[in]BSTR bstrState – a state of the current UI element that needs to be verified. Can be any combination of MSAA states separated by comma.

[in] BSTR bstrValue – a value of the current UI element that needs to be verified

**Description:** Verifies state and value of the current UI element. See SetVerificationFlag for more details.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT GetScreenElementInfo([in,out] BSTR \*pbstrUIElementName, [in,out] BSTR \*pbstrUIElementRole, [in,out] BSTR \*pbstrUIElementClass, [in,out] BSTR \*pbstrValue, [in,out] BSTR\* pbstrState);

**Parameters:**

[in,out] BSTR \*pbstrUIElementName – a name of the current UI element

[in,out] BSTR \*pbstrUIElementRole – a role of the current UI element

[in,out] BSTR \*pbstrUIElementClass – a class name of the current UI element

[in,out] BSTR \*pbstrValue – a value of the current UI element

[in,out] BSTR\* pbstrState – a state of the current UI element

**Description:** Returns name, role, class name, value and state for the current UI element.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT BringScreenElementUp();

**Description:** Brings the current UI element to the top in z-order and sets the focus to it.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SendKeys([in]BSTR keys);

**Parameters:**

[in]BSTR keys – key strokes to be typed in the current UI element.

**Description:** Sends the key strokes to the current UI element. If the focus is not set to the current UI element or if it’s lost during typing this function fails. See help for **System.Windows.Forms.SendKeys** for more details on the syntax. This function supports Unicode.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

TypeString([in] BSTR bstrKeys, [in] int nSleepBetweenActions, [in] BOOL bLiteral);

**Parameters:**

[in] BSTR keys – key strokes to be typed in the current UI element.

[in] int nSleepBetweenActions – a time that RPF pauses for between each key stroke is typed [in] BOOL bLiteral – flag that forces to emulate each key stroke literaly if TRUE

**Description:** Types the key strokes without focus verification. See help for **System.Windows.Forms.SendKeys** for more details on the syntax. This function supports Unicode**.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT WaitForScreenElementGone([in] INT nElapse, [in] INT nTimeOut);

**Parameters:**

([in] INT nElapse – time between verifications

[in] INT nTimeOut – time out

**Description:** Waits until the current UI element becomes unavailable using the polling mechanism

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT GetClickablePoint([out] POINT\* clickablePoint);

**Parameters:**

POINT\* clickablePoint – a point that belogs to the current UI element and that is available for a mouse click

**Description:** Gets a clickable point for the current UI element.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT EnableInfoWindow([in] BOOL bEnable);

**Parameters:**

[in] BOOL bEnable – flag that defines if RPF’s Info window should be displayed

**Description:** Enables/Disables RPF’s Info window.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SetDebugMode([in] INT nDebugMode);

**Parameters:**

[in] INT nDebugMode – flag that sets the level of the debug mode. If 0 then debug mode is turned off

**Description:** Enables/Disables RPF’s debug mode. If greater than zero defines the level of the debug mode. When level equals 1 RPF displays information related to the search of the first UI element in the QueryId, with level equals 2 RPF displays information for the second element, etc.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SetLogFailureAsInfo([in] int nLogFailureAsInfo);

**Parameters:**

[in] int nLogFailureAsInfo – flag that defines if a failure is recorded as information. If greater than zero then a failure is recorded as information not as a failure.

**Description:** Sets the flag that that defines if a failure is recorded as information.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SetVerificationFlag([in] int nVerificationFlag);

**Parameters:**

[in] int nVerificationFlag – flag that defines the behavior of VerifyScreenElement. Can be any OR combination of the following flags:

VF\_VALUE\_STATE\_EXACT = 0x00000000 – VerifyScreenElement returns S\_OK only if value and state match requested value and state exactly.

VF\_VALUE\_DONTCARE = 0x00000001 – VerifyScreenElement verifies only state

VF\_STATE\_DONTCARE = 0x00000002 – VerifyScreenElement verifies only value

VF\_STATE\_HAS = 0x00000004 – VerifyScreenElement verifies that state of the current UI element contains the requested state

**Description:** Sets the flag that that defines if a failure is recorded as information.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SetRobustnessLevel([in] int nRobustnessLevel);

**Parameters:**

[in] int nRobustnessLevel – if positive sets the number of quantums that constitute a total time used to verify that the foreground state is in the Wait state and the reason for this state is WaitForUserInput.

If -1 waits until all threads in the current process are in the correct state

If -2 waits until all threads in the whole system are in the correct state

**Description:** Defines the behavior of the built-in WaitForReady (WFR) functionality. Usually called with nRobustnessLevel = -1 if the default behavior doesn’t provide enough robustness. Since WFR with nRobustnessLevel = -1 is a very expensive procedure set nRobustnessLevel = 1 as soon as you don’t need it to be -1.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SetLoggingFlag([in] int nLoggingFlag);

**Parameters:**

[in] int nLoggingFlag – flag that defines the behavior of LogStep and LogInformation. Can be any OR combination of the following flags:

FL\_DISABLE\_LOGGING = 0xFFFFFFFF - disables all logging

FL\_LOG\_SEARCH\_AND\_PRIMITIVES = 0x00000001 – only information on search and primitives will be logged

FL\_LOG\_TIME = 0x00000002 – time will be logged

FL\_LOG\_INTERNAL\_INFO = 0x00000004 - internal steps performed by the playback engine will be logged

FL\_DUMP\_SCREEN = 0x00000008 - screen snapshot will be taken and save in a jpg file in the case of a failure

FL\_LOG\_FAILURE\_AS\_INFO = 0x00000010 – a failure will be logged as info. This is mostly used for negative verifications

**Description:** Defines the behavior of LogStep and LogInformation. The default value for nLoggingFlag is

FL\_LOG\_SEARCH\_AND\_PRIMITIVES | FL\_LOG\_INTERNAL\_INFO | FL\_DUMP\_SCREEN | FL\_LOG\_TIME

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SetRobustnessLevelEx([in] int nRobustnessLevel, [in] BOOL bSystemWide, [in] BSTR bstrBackGroundThreadStates, [in] BSTR bstrBackGroundThreadWaitReasons, [in, defaultvalue(NULL)] BSTR bstrForeGroundThreadStates, [in, defaultvalue(NULL)] BSTR bstrForeGroundThreadWaitReasons);

**Parameters:**

**Description:** To be added

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SetSearchTimeOut([in] INT nSearchTimeOut, [out]INT \*nPreviousSearchTimeOut);

**Parameters:**

[in] INT nSearchTimeOut – a timeout for search routines FindScreenElement and FindScreenElementEx.

[out]INT \*nPreviousSearchTimeOut – a value previously set to the timeout

**Description:** Defines the timeout for FindScreenElement and FindScreenElementEx.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT LockNextSearches([in] BOOL bLock);

**Parameters:**

[in] BOOL bLock – flag that defines if a next search should be disabled

**Description:** Disables all searches if bLock is set to TRUE

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT AddMonitoringInfo([in] IMonitoringWindowsCallback\* MonitoringWindowsCallback);

**Parameters:**

([in] IMonitoringWindowsCallback\* MonitoringWindowsCallback – an object that implements IMonitoringWindowsCallback. Both managed and native wrappers have a class that implements IMonitoringWindowsCallback, see an example below.

**Description:** Adds the monitoring information to the RPF’s internal list. This list will be processed after function StartMonitoringWindows is called. As many AddMonitoringInfo can be called as necessary.

**Native C Example:** In the following example callback function AssertMessageHandler is called when a window with caption "Visual Studio Assertion" and class name "#32770" comes up.

void AssertMessageHandler(BSTR lpstrWindowClass, BSTR bstrlpstrWindowName)

{

FromPartialQueryId(…);

LeftButtonClick(-, -1);

LogInformation(L"[INFO] \"Assert\" dismissed\r\n", FALSE);

BringUpMainWindow();

}

int APIENTRY WinMain()

{

InitPlayBack();

AddMonitoringInfo(new MonitorWindowBase(L"#32770", L"Visual Studio Assertion", AssertMessageHandler));

StartMonitoringWindows(1000);

Test code

// Stop monitoring windows

StopMonitoringWindows();

FinishPlayBack();

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT StartMonitoringWindows([in] int nElapse);

**Parameters:**

[in] int nElapse – time between two monitoring events

**Description:** Starts monitoring windows based on the information defined by AddMonitoringInfo.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT StopMonitoringWindows();

**Description:** Stops monitoring windows.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SetCustomLogger([in] ILoggerCallback\* LoggerCallback);

**Parameters:**

([in] ILoggerCallback\* LoggerCallback – an object that implements ILoggerCallback. Both managed and native wrappers have a class that implements ILoggerCallback, see an example below.

**Description:** Redefines the logging functionality

**Native C Example:** In the following example callback function AssertMessageHandler is called when a window with caption "Visual Studio Assertion" and class name "#32770" comes up.

void CustomLoggingHandler(BSTR\* bstrLogString, BOOL \*bLogDefault, BOOL \*bDumpScreen)

{

// Set bLogDefault = false if you want to turn off the default RPF logging   
// Set bDumpScreen = false if you don't want a screen snapshot to be taken

if(wcsstr(\*bstrLogString, L"[FAILED]"))   
{   
 // Call SetCustomLogger(NULL); before you call   
 // RPF's LogInformation to valid stack overflow   
 SetCustomLogger(NULL);   
 LogInformation(\*bstrLogString, FALSE);   
 LogStep(E\_ABORT, L"Can not recover from previous failures");   
}

}  
  
int APIENTRY WinMain()

{

InitPlayBack();

SetCustomLogger(new LoggerBase(CustomLoggingHandler));

Test code

// Set the logger back to default

SetCustomLogger(NULL);

FinishPlayBack();

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT LogStep([in]HRESULT hr, [in]BSTR bstrStepInfo);

**Parameters:**

[in]HRESULT hr – HRESULT returned by a step performed by the playback engine

[in]BSTR bstrStepInfo – Description of a step performed by the playback engine

**Description:** Logs an information for a step step performed by the playback engine

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT LogInformation([in] BSTR bstrMessage, [in] BOOL bDeleteFile);

**Parameters:**

[in] BSTR bstrMessage – a message to be logged

[in] BOOL bDeleteFile – flag that defines if the log file should be cleared before the message is logged

**Description:** Logs a message to the log file

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SetClipBoard([in] BSTR bstrClipBoardString);

**Parameters:**

[in] BSTR bstrClipBoardString – a string to be set in the clipboard

**Description:** Sets the clipboard

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT GetClipBoard([out] BSTR \*pbstrClipBoardString);

**Parameters:**

[out] BSTR \*pbstrClipBoardString – a string stored in the clipboard.

**Description:** Gets a string stored in the clipboard

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT MoveDirectoryToRecycleBin([in] BSTR bstrDirectory);

**Parameters:**

([in] BSTR bstrDirectory – a directory to be deleted

**Description:** Deletes a directory

**Note:** This function may be removed in the final version of RPF’s MSAA-based playback

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SetTargetProcess([in] INT nTargetProcessId, [out]INT \*nCurrentTargetProcessId);

**Parameters:**

[in] INT nTargetProcessId – a new target process Id

[out]INT \*nCurrentTargetProcessId – previous target process Id

**Description:** Sets target process. One can use SearchTargetProcessElementsOnly so only UI elements that belong to the target process are searched by RPF

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT SearchTargetProcessElementsOnly([in] BOOL bEnable);

**Parameters:**

([in] BOOL bEnable – flag that defines that only UI elements that belong to the target process should be searched by RPF

**Description:** Defiines if only UI elements that belong to the target process are searched by RPF

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT TerminateProcessByModuleName([in] BSTR bstrModuleName, [in] int nExitCode);

**Parameters:**

[in] BSTR bstrModuleName - module name for the process to be terminated

[in] int nExitCode – exit code

**Description:** Terminates a process by the process’ module name

**Note:** This function may be removed in the final version of RPF’s MSAA-based playback

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT TerminateTargetProcess([in] int nTargetProcessId);

**Parameters:**

[in] int nTargetProcessId – if 0 then target process defined by SetTargetProcess or InitPlayBackForTargetProcess will be terminated. Otherwise a process defined by this value will be terminated.

**Description:** Terminates target process

**Note:** This function may be removed in the final version of RPF’s MSAA-based playback

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT WaitForTargetProcessIdle([in] int nDeltaTime, [in] int nTimeOut);

**Parameters:**

[in] int nDeltaTime – time between two measurements of the performance counters

[in] int nTimeOut – timeout

**Description:** Waits until certain performance counters don’t change within nDeltaTime. A user usually doesn’t need to call this function directly. This approach is used internally by the WaitForReady functionality if the information about the thread(s) is unavailable for some reason.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT WaitForMainWindowTitleContains([in]BSTR lpszTitle, [in] int nTimeOut);

**Parameters:**

[in]BSTR lpszTitle – a string to wait for in the title of the main window

[in] int nTimeOut – time out

**Description:** Waits for the main window’s title to contain the requested string. The main window is set by InitPlayBackForTargetWindow.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

HRESULT BringUpMainWindow();

**Parameters:**

**Description:** Brings the main window to the top in z-order and sets the focus to it. The main window is set by InitPlayBackForTargetWindow.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**The following are VS specific functions that will be documented later**

HRESULT LaunchIDE([in] BSTR bstrDevenvFullName);

HRESULT GetDevenvBaseDirectory([out] BSTR \*pbstrDevenvBaseDirectory);

HRESULT GetFrameworkBaseDirectory([out] BSTR \*pbstrFrameworkBaseDirectory);

HRESULT GetDevenvVersion([in] INT procID, [out] INT \*pProductMajorPart, [out] INT \*pProductMinorPart, [out] INT \*pProductBuildPart);

HRESULT PrepareForBuild([in] BSTR bstrOutput);

HRESULT WaitForBuildIsDone([in] int nTimeOut, [in,out] BSTR \*pbstrOutput, [in]int ccOutput);

logo_stamp