Morae RRT Event Interfaces

Introduction:

These interfaces provide access to the native RRT event streams that are normally captured in a Morae recording session. They can be used by various types of plug-ins, such as those that implement IMoraeEventSearch, IMoraeVideoDraw, or IMoraeAnalysis. This allows development of plug-ins that use the native Morae RRT data to provide enhanced visualization or analysis functionality in Morae Manager.

Plug-ins can get pointers to these event streams from the IMoraeRecording interface (described in Information Supplied by Manager). A call to IMoraeRecording::GetDataStream can be used to get an IMoraeEventStream interface for the data stream of interest. (The IID values that apply to each stream are given below.) IMoraeEventStream::EnumEvents can be called to get an enumerator for all of the events captured as part of the stream. The plug-in can call QueryInterface with the appropriate interface ID (given below) on each IMoraeEvent interface retrieved from the enumerator to get access to the full details captured with the event. The interfaces for individual RRT events described in this document are:

1. IMoraeEventMouse – This interface provides more detailed information about cursor events.

2. IMoraeEventAA – This interface provides detailed information about Active Accessibility events.

3. IMoraeEventKeystroke – This interface provides information about keystrokes.

4. IMoraeEventText – This interface provides information about captured text.

5. IMoraeEventBrowser – This interface provides information about page change events captured in web browsers

6. IMoraeEventMarker – This interface provides information about markers that were set either during the recording or in Manager during analysis.

7. IMoraeEventTask – This interface provides information about tasks that were logged either during the recording or in Manager.

8. IMoraeEventSurveyQuestion – This interface provides information about an individual survey question.

9. IMoraeEventStreamSurvey – This interface provides information about a survey. Surveys are handled differently from Morae's other event streams. A recording may use several surveys. To get at the survey information, a plug-in can call IMoraeRecording::GetDataStream to get an interface to an object that represents all

surveys. Calling IMoraeEventStream::EnumEvents retrieves an enumerator that allows access to the individual surveys. These individual survey objects implement both IMoraeEvent and IMoraeEventStream. IMoraeEvent::GetEventTypeID can be called to get the GUID that uniquely identifies a survey (not a GUID that identifies the event stream type). A call to IMoraeEventStream::EnumEvents on an individual survey object outputs an IEnumMoraeEvent interface that allows enumeration through the individual questions in that survey.

Interfaces:

1. IMoraeEventMouse (derives from IMoraeSearchableEvent)

This stream captures mouse clicks and movement.

Interface ID:

IID_ IMoraeEventMouse = 55bb8b15-e5e5-4778-90d0-c75a9861f29d

Data stream ID:

IID_ MOUSEEVENTTYPE = 87DFC9CA-BFBA-47a0-8A0F-60ED99813CA3

Methods:

a. **HRESULT FindString (**

BSTR str,	**[in]**
int* pResult,	**[out]**
int nCriteria)	**[in]**

This call searches for a string within the event. When a mouse event is recorded, the text of the window in which the event occurred is stored, as well as the text of the parent window. This can be useful, for example, to determine whether a mouse click occurred in a button with a particular label. Note that if a case-insensitive search is to be made and the full-string flag is not set, this should be a string of all lowercase characters.

Arguments:

i. str – BSTR variable that provides the string to search for.

ii. pResult – Pointer to an int value that will be set to the result. Possible values are:
    a. 0x00000000 – The text was not found.
    b. 0x00000001 – The text was found in the window text.
    c. 0x00000002 – The text was found in the parent window text.

iii. nCriteria – int value that is a bitwise combination of flags to specify how the comparison is conducted. These flags are:
    a. 0x00000001 – The search should be case-sensitive.
    b. 0x00000002 – Entire string must match.
    c. 0x00000004 – Search within the text of the window that received the event.
    d. 0x00000008 – Search within the parent window text.

b. **HRESULT GetLocation (**

int* pX, **[out]**
int* pY) **[out]**

This call gets the location (in pixels) at which the event occurred, relative to the upper left corner of the screen recording area. Note that X increases going to the right, Y increases going down.

Arguments:

i. pX – Pointer to an int that receives the X location.
ii. pY – Pointer to an int that receives the Y location.

c. **HRESULT GetButtonFlags (**

int* pFlags) **[out]**

This call retrieves the type of mouse event that was recorded. *pFlags will be set to a bitwise combination of flags described below.

Arguments:

i. pFlags – Pointer to an int value that will be set to the result. Possible values are:
   a. MCT_NC (0x01000000) indicates that this event occurred in the non-client area of the window. It can be combined with any of the other flags listed below.
   b. MCT_LDOWN (0x00000001) indicates a left-button down event.
   c. MCT_RDOWN (0x00000002) indicates a right-button down event.
   d. MCT_MDOWN (0x00000004) indicates a middle-button down event.
   e. MCT_XDOWN (0x00000008) indicates a button down event for either X button (in recordings made prior to Morae version 2.0).
   f. MCT_X1DOWN (0x00010000) indicates an X1-button down event.
   g. MCT_X2DOWN (0x00020000) indicates an X2-button down event.
   h. MCT_LUP (0x00000010) indicates a left-button up event.
   i. MCT_RUP (0x00000020) indicates a right-button up event.
   j. MCT_MUP (0x00000040) indicates a middle-button up event.
   k. MCT_XUP (0x00000080) indicates a button-up event for either X button (in recordings made prior to Morae version 2.0).
   l. MCT_X1UP (0x00040000) indicates an X1-button up event.
   m. MCT_X2UP (0x00080000) indicates an X2-button up event.
   n. MCT_LDBCLCK (0x00000100) indicates a left-button double-click event.
   o. MCT_RDBCLCK (0x00000200) indicates a right-button double-click event.
   p. MCT_MDBCLCK (0x00000400) indicates a middle-button double-click event.
   q. MCT_XDBCLCK (0x00000800) indicates a double-click event for either X button (in recordings made prior to Morae version 2.0).

r. MCT_X1DBCLCK (0x00100000) indicates an X1-button double-click event.
s. MCT_X2DBCLCK (0x00200000) indicates an X2-button double-click event.
t. MCT_WHEEL_PLUS (0x00001000) indicates a positive mouse wheel click event.
u. MCT_WHEEL_MINUS (0x00400000) indicates a negative mouse wheel click event.
v. MCT_MOVE (0x00002000) indicates a mouse move event.

d. HRESULT GetModKeyFlags (

int* pFlags) [out]

This call retrieves which, if any, modifier keys were depressed when the mouse event was recorded. *pFlags will be set to a bitwise combination of flags described below.

Arguments:

i. pFlags – Pointer to an int value that will be set to the result. Possible values are:
   a. KCF_SHIFT (0x00000004) indicates that the shift key was down.
   b. KCF_CONTROL (0x00000008) indicates that the control key was down.
   c. KCF_SHIFTLOCK (0x00010000) indicates that shift-lock was in effect.
   d. KCF_LWINDOW (0x00020000) indicates that the left windows key was down.
   e. KCF_RWINDOW (0x00040000) indicates that the right windows key was down.
   f. KCF_MENU (0x20000000) indicates that the menu (alt) key was down.

e. HRESULT GetParentHWND (

VARIANT* pVar) [out]

This gets the window handle of the parent to the window that received the mouse event notification.

Arguments:

i. pVar – Pointer to a VARIANT value that will be set to a type of VT_UI4. Its ulVal will be set to the HWND of the parent of the window that received the mouse event.

f. HRESULT GetWndText (

BSTR* pstr)	**[out]**

This gets the window text of the window that received the mouse event.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

g. HRESULT GetParentWndText (

BSTR* pstr)	**[out]**

This gets the window text of the parent of the window that received the mouse event.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

2. IMoraeEventAA (derives from IMoraeSearchableEvent)

This stream captures Active Accessibility events, including windows focus, resize, and move events.

Interface ID:

IID_ IMoraeEventAA = AFEE7ECF-713A-4801-B3BD-1C9BE90DBC56

Data stream ID:

IID_AAEVENTTYPE = 796ADB4D-E4A5-405c-85E0-6A5F57D65CF4

Methods:

a. HRESULT FindString (

BSTR str,	**[in]**
int* pResult,	**[out]**
int nCriteria)	**[in]**

This call searches for a string within the event. When an AA event is recorded, the text of the window in which the event occurred is stored, as well as the text of its parent window and the topmost parent. str is the string to find. Note that if a case-insensitive search is to be made and the full-string flag is not set, this should be a string of all lowercase characters. nCriteria is a bitwise set of flags that determine how the search is made:

Arguments:

i. str – BSTR variable that provides the string to search for.

ii. pResult – Pointer to an int value that will be set to the result. Possible values are:
    a. 0x00000000 – The text was not found.
    b. 0x00000001 – The text was found in the window text.
    c. 0x00000002 – The text was found in the parent window text.
    d. 0x00000004 – The text was found in the topmost parent window text.

iii. nCriteria – int value that is a bitwise combination of flags to specify how the comparison is condutcted. These flags are:
    a. 0x00000001 – The search should be case-sensitive.
    b. 0x00000002 – Entire string must match.
    c. 0x00000004 – Search within the text of the window that received the event.
    d. 0x00000008 – Search within the parent window text.
    e. 0x00000010 – Search within the topmost parent window text.

b. HRESULT GetLocation (

int* pLeft,	**[out]**
int* pTop,	**[out]**
int* pRight,	**[out]**
int* pBottom)	**[out]**

This call gets the location (in pixels) at which the event occurred, relative to the upper left corner of the screen recording area. Note that X increases going to the right, Y increases going down.

Arguments:

i. pLeft – Pointer to an int that will be set to the left side of the event rectangle (in pixels).

ii. pTop – Pointer to an int that will be set to the top of the event rectangle (in pixels).

iii. pRight – Pointer to an int that will be set to the right side of the event rectangle (in pixels).

iv. pBottom – Pointer to an int that will be set to the bottom of the event rectangle (in pixels).

c. HRESULT GetEventDetails (

int* pEventType,	**[out]**
int* pObjType,	**[out]**
int* pDepth)	**[out]**

This call retrieves details about the AA event.

Arguments:

i. pEventType – Pointer to an int value that will be set to the type of AA event. Possible values (defined in WinUser.h) are:
    a. EVENT_OBJECT_FOCUS (0x8005) indicates that an object has received focus.
    b. EVENT_SYSTEM_MOVESIZEEND (0x0000000B) indicates a window has been moved or resized.
ii. pObjType – Pointer to an int value that will be set to the AA object type. This may be application-specific or refer to a system object. (See documentation for the SetWinEventHook windows SDK function.) Possible values for system objects (defined in WinUser.h) include:
    a. OBJID_WINDOW (0x00000000)
    b. OBJID_SYSMENU (0xFFFFFFFF)
    c. OBJID_MENU (0xFFFFFFFD)
iii. pDepth – Pointer to an int that will be set to number of steps that the window that fired this event is removed from a top-level window. In other words, a value of 0 indicates that a top-level window was moved or resized. A value of 1 indicates that a child of a top-level window was moved or resized, etc.

d. HRESULT GetParentHWND (
VARIANT* pVar) **[out]**

This gets the window handle of the parent to the window that fired the AA event.

Arguments:

i. pVar – Pointer to a VARIANT value that will be set to a type of VT_UI4. Its ulVal will be set to the HWND of the parent of the window that received the mouse event.

e. HRESULT GetTopParentHWND (
VARIANT* pVar) **[out]**

This gets the window handle of the topmost parent to the window that fired the AA event.

Arguments:

i. pVar – Pointer to a VARIANT value that will be set to a type of VT_UI4. Its ulVal will be set to the HWND of the topmost parent of the window that fired the AA event.

f. HRESULT GetWndText (
BSTR* pstr) **[out]**

This gets the window text of the window that fired the AA event.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

g. **HRESULT GetParentWndText (**

BSTR* pstr) **[out]**

This gets the window text of the parent of the window that fired the AA event.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

h. **HRESULT GetTopParentWndText (**

BSTR* pstr) **[out]**

This gets the window text of the topmost parent of the window that fired the AA event.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

3. IMoraeEventKeystroke (derives from IMoraeSearchableEvent)

This stream captures Active Accessibility events, including windows focus, resize, and move events.

Interface ID:

IID IMoraeEventKeystroke = F8E9DAB6-4645-43A7-A82D-6AC56ADD108A

Data stream ID:

IID_KEYEVENTTYPE = 008D470B-493C-4974-9BD6-1112D90AC68B

Methods:

a. **HRESULT GetLocation (**

int* pLeft, **[out]**
int* pTop, **[out]**
int* pRight, **[out]**
int* pBottom) **[out]**

This call gets the location (in pixels) of the window that received the keystroke.

Arguments:

i. pLeft – Pointer to an int that will be set to the left side of the event rectangle (in pixels).
ii. pTop – Pointer to an int that will be set to the top of the event rectangle (in pixels).
iii. pRight – Pointer to an int that will be set to the right side of the event rectangle (in pixels).
iv. pBottom – Pointer to an int that will be set to the bottom of the event rectangle (in pixels).

b. HRESULT GetEventDetails (

```
int* pEventType,        [out]
int* pKey,              [out]
int* pModifiers,        [out]
int* pDepth,            [out]
int* pCodes )           [out]
```

This call retrieves details about the keystroke event.

Arguments:

i. pEventType – Pointer to an int value that will be whether this is a key-up or key-down event. A value of '1' indicates key down, '0' indicates key up. Currently, Morae only captures key down events.
ii. pKey – Pointer to an int value that will be set to the virtual key code of the keystroke. Values of virtual key codes are defined in WinUser.h.
iii. pModifiers – Pointer to an int value that receives bit-wise flags to indicate any modifier keys that were depressed along with the keystroke. Possible values are:
    a. KCF_SHIFT (0x00000004)
    b. KCF_CONTROL (0x00000008)
    c. KCF_SHIFTLOCK (0x00010000)
    d. KCF_LWINDOW (0x00020000)
    e. KCF_RWINDOW (0x00040000)
    f. KCF_MENU (0x20000000)
iv. pDepth – Pointer to an int that will be set to number of steps that the window that received this event is removed from a top-level window. In other words, a value of 0 indicates that a top-level window was received this keystroke. A value of 1 indicates that a child of a top-level window received the event, etc.
v. pCodes – Pointer to an int value that receives information about the repeat count, scan code, and other flags. For details, see the documentation for the SetWindowsHookEX Windows API call.

4. IMoraeEventText (derives from IMoraeSearchableEvent)

This stream captures text that applications display on the screen.

Interface ID:

IID_ IMoraeEventText = B47BEB4A-AE9E-451f-9CEF-7E9BC4D00825

Data stream ID:

IID_TEXTEVENTTYPE = 21351BCA-4590-40fc-B470-F6066D81C59A

Methods:

a. **HRESULT FindString (**

BSTR str,	**[in]**
int* pResult,	**[out]**
int nCriteria)	**[in]**

This call searches for a string within the captured text. When a text event is recorded, the text of the topmost parent window in which the text was written is also stored. str is the string to find. Note that if a case-insensitive search is to be made and the full-string flag is not set, this should be a string of all lowercase characters. nCriteria is a bitwise set of flags that determine how the search is made:

Arguments:

i. str – BSTR variable that provides the string to search for.
ii. pResult – Pointer to an int value that will be set to the result. Possible values are:
    a. 0x00000000 – The text was not found.
    b. 0x00000020 – The string was found in the captured text.
    c. 0x00000004 – The text was found in the topmost parent window text.
iii. nCriteria – int value that is a bitwise combination of flags to specify how the comparison is condutcted. These flags are:
    a. 0x00000001 – The search should be case-sensitive.
    b. 0x00000002 – Entire string must match.
    c. 0x00000020 – Search within the captured text.
    d. 0x00000010 – Search within the topmost parent window text.

b. **HRESULT GetLocation (**

int* pLeft,	**[out]**
int* pTop,	**[out]**
int* pRight,	**[out]**
int* pBottom)	**[out]**

This call gets the location (in pixels) at which the event occurred, relative to the upper left corner of the screen recording area. Note that X increases going to the right, Y increases going down.

Arguments:

i. pLeft – Pointer to an int that will be set to the left side of the event rectangle (in pixels).
ii. pTop – Pointer to an int that will be set to the top of the event rectangle (in pixels).
iii. pRight – Pointer to an int that will be set to the right side of the event rectangle (in pixels).
iv. pBottom – Pointer to an int that will be set to the bottom of the event rectangle (in pixels).

c. **HRESULT GetEventDetails (**

```
int* pFlags,          [out]
int* pDepth,          [out]
int* pAveWidth,       [out]
int* pAveHeight )     [out]
```

This call retrieves details about the text event.

Arguments:

i. pFlags – Pointer to an int value that whether this text output occurred in the active application or active window. Possible values include:
   a. 0x00000001 indicates that the application was active.
   b. 0x00000002 indicates this text was captured from the active window.
   c. 0x00000004 indicates that this text was captured as part of an Active Accessibility event.

ii. pAveWidth – Pointer to an int that will be set to the average width (in pixels) of the text characters as they were drawn on the screen.

iii. pAveHeight – Pointer to an int that will be set to the average height (in pixels) of the text characters as they were drawn on the screen.

d. **HRESULT GetTopParentHWND (**

```
VARIANT* pVar )       [out]
```

This gets the window handle of the topmost parent to the window that displayed the text.

Arguments:

i. pVar – Pointer to a VARIANT value that will be set to a type of VT_UI4. Its ulVal will be set to the HWND of the topmost parent of the window that fired the AA event.

e. **HRESULT GetText (**

```
BSTR* pstr )          [out]
```

This gets the text that was captured.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

f. **HRESULT GetTopParentWndText (**

```
BSTR* pstr )          [out]
```

This gets the window text of the parent of the window that displayed the text.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

5. IMoraeEventBrowser (derives from IMoraeSearchableEvent)

This stream captures navigation events in web browsers, including tab changes and page navigation.

Interface ID:

IID_IMoraeEventBrowser = 79E8074E-D643-4853-AF92-1DB74EF88AE5

Data stream ID:

IID_BROWSEREVENTTYPE = C18F48AF-2DE8-40fd-AA87-DAAC8D198625

Methods:

a. HRESULT FindString (

BSTR str,	**[in]**
int* pResult,	**[out]**
int nCriteria)	**[in]**

This call searches for a string within the captured text. When a browser event is recorded, both the URL and the title of the page are recorded. str is the string to find. Note that if a case-insensitive search is to be made and the full-string flag is not set, this should be a string of all lowercase characters. nCriteria is a bitwise set of flags that determine how the search is made.

Arguments:

i. str – BSTR variable that provides the string to search for.

ii. pResult – Pointer to an int value that will be set to the result. Possible values are:
   a. 0x00000000 – The text was not found.
   b. 0x00000010 – The string was found in the URL.
   c. 0x00000001 – The text was found in the page title.

iii. nCriteria – int value that is a bitwise combination of flags to specify how the comparison is condutcted. These flags are:
   a. 0x00000001 – The search should be case-sensitive.
   b. 0x00000002 – Entire string must match.
   c. 0x00000040 – Search within the URL.
   d. 0x0000001C – Search within the page title.

b. HRESULT GetLocation (

int* pLeft,	**[out]**
int* pTop,	**[out]**
int* pRight,	**[out]**
int* pBottom)	**[out]**

This call gets the location (in pixels) at which the event occurred, relative to the upper left corner of the screen recording area. Note that X increases going to the right, Y increases going down.

Arguments:

i. pLeft – Pointer to an int that will be set to the left side of the event rectangle (in pixels).
ii. pTop – Pointer to an int that will be set to the top of the event rectangle (in pixels).
iii. pRight – Pointer to an int that will be set to the right side of the event rectangle (in pixels).
iv. pBottom – Pointer to an int that will be set to the bottom of the event rectangle (in pixels).

c. **HRESULT GetEventType (**

int* pType)	**[out]**

This call gets the type of event that was recorded.

Arguments:

i. pType – Pointer to an int value that will be set to the result. Possible values:
   a. 0x00000080 – The user switched to a different tab.
   b. 0x00000001 – The browser finished loading a page.

d. **HRESULT GetEventDetails (**

int* pEventDetails,	**[out]**
int* pEventFlags,	**[out]**
int* pTargetFlags)	**[out]**

This method is not currently implemented.

e. **HRESULT GetURL (**

BSTR* pstr)	**[out]**

This gets the URL of the navigation event.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

f. **HRESULT GetLinkURL (**

BSTR* pstr)	**[out]**

This method is not currently implemented.

g. **HRESULT GetObjectText (**

BSTR* pstr)	**[out]**

This method is not currently implemented.

h. **HRESULT GetObjectAltText (**

BSTR* pstr)	**[out]**

This method is not currently implemented.

i. **HRESULT GetObjectTipText (**

    BSTR* pstr) **[out]**

This method is not currently implemented.

j. **HRESULT GetWndTitleText (**

    BSTR* pstr) **[out]**

This gets the page title of the URL site for the navigation event.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

6. IMoraeEventMarker (derives from IMoraeSearchableEvent)

This stream records markers that are entered either during a recording or analysis.

Interface ID:

IID_IMoraeEventMarker = D69059AF-0660-4768-AEEB-2E8764015E53

Data stream ID:

IID_MARKEREVENTTYPE = 84B82AB8-D096-43ef-A043-D30E2382C448

Methods:

a. **HRESULT FindString (**

    BSTR str, **[in]**
    int* pResult, **[out]**
    int nCriteria) **[in]**

This call searches for a string within the marker. Note that if a case-insensitive search is to be made and the full-string flag is not set, this should be a string of all lowercase characters.

Arguments:

i. str – BSTR variable that provides the string to search for.
ii. pResult – Pointer to an int that will be set to the result. Possible values are:
    a. 0x00000000 – The text was not found.
    b. 0x00000200 – The string was found in the name of the marker.
    c. 0x00000400 – The text was found in the note of the marker.
    d. 0x00000800 – The text was found in the marker creator’s name.
iii. nCriteria – int value that is a bitwise combination of flags to specify how the comparison is conducted. These flags are:
    a. 0x00000001 – The search should be case-sensitive.
    b. 0x00000002 – Entire string must match.
    c. 0x00000800 – Search within the marker name.
    d. 0x00001000 – Search within the marker note.
    e. 0x00002000 – Search within the name of the marker creator.

b. **HRESULT GetMarkerType (**

int* pType) **[out]**

This call gets the type of marker that was recorded.

Arguments:

i. pType – Pointer to an int value that will be set to the result.

c. **HRESULT GetName (**

BSTR* pstr) **[out]**

This gets the name of the marker.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

d. **HRESULT GetCreatorName (**

BSTR* pstr) **[out]**

This gets the name of the creator of the marker.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

e. **HRESULT GetAttr (**

int nAttrID, **[in]**

int* pAttr) **[out]**

This method retrieves a specific attribute of the marker.

Arguments:

i. nAttrID – int value that defines the attribute. Currently supported values are:
   a. 0x00000001 – Marker type (outputs same value as GetMarkerType).
   b. 0x00000002 – Creator type. Possible output values are:
      i. 0x00000000 – Marker was created in Manager.
      ii. 0x00000001 – Marker was created by an Obsever.
      iii. 0x00000004 – Marker was created by the participant.
      iv. 0x00000003 – Marker was created by a local logger.
      v. 0x00000002 – Marker was created by a COM client.

ii. pAttr – Pointer to an int value that will be set to the result.

f. **HRESULT GetNote (**

BSTR* pstr) **[out]**

This gets the note of the marker.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

g. **HRESULT GetDisplayPath (**

BSTR* pstr) **[out]**

This method is deprecated.

h. **HRESULT GetAudioNoteFileName (**

BSTR* pstr) **[out]**

This gets the full path to the audio note that is associated with this marker. Note that this path will change if a new audio note is recorded for the marker.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

i. **HRESULT GetMarkerScoreID (**

int* pScore) **[out]**

This call gets the score that has been set for the marker.

Arguments:

i. pType – Pointer to an int value that will be set to the result.

j. **HRESULT GetMarkerScoreName (**

BSTR* pstr) **[out]**

This gets the string that describes the score that has been set for the marker.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

k. **HRESULT SetMarkerType (**

int nType) **[in]**

This call sets the type of marker.

Arguments:

i. pType – int value that defines the marker type.

l. **HRESULT SetMarkerScoreID (**

int nType) **[in]**

This call sets the score of marker.

Arguments:

i. pType – int value that defines the marker score.

7. IMoraeEventTask (derives from IMoraeSearchableEvent)

This stream records tasks that are logged either during a recording or analysis.

Interface ID:

IID_IMoraeEventTask = 904CB9D1-8A61-4326-9949-D84B887741F1

Data stream ID:

IID_TASKEVENTTYPE = D0F89A2B-DF21-47f9-8507-9E538B6A41A7

Methods:

a. **HRESULT FindString (**

BSTR str,	**[in]**
int* pResult,	**[out]**
int nCriteria)	**[in]**

This call searches for a string within the task. str is the string to find. Note that if a case-insensitive search is to be made and the full-string flag is not set, this should be a string of all lowercase characters. nCriteria is a bitwise set of flags that determine how the search is made.

Arguments:

i. str – BSTR variable that provides the string to search for.
ii. pResult – Pointer to an int value that will be set to the result. Possible values are:
    a. 0x00000000 – The text was not found.
    b. 0x00000200 – The string was found in the name of the task.
    c. 0x00000400 – The text was found in the note of the task.
iii. nCriteria – int value that is a bitwise combination of flags to specify how the comparison is condutcted. These flags are:
    a. 0x00000001 – The search should be case-sensitive.
    b. 0x00000002 – Entire string must match.
    c. 0x00000800 – Search within the task name.
    d. 0x00001000 – Search within the task note.

b. **HRESULT GetTaskID (**

UINT* pType)	**[out]**

This call gets the type of task that was logged.

Arguments:

i. pType – Pointer to an int value that will be set to the result.

c. **HRESULT GetName (**

BSTR* pstr)	**[out]**

This gets the name of the task.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

d. HRESULT GetTaskScoreID (

int* pScore) **[out]**

This call gets the score that has been set for the task.

Arguments:

i. pType – Pointer to an int value that will be set to the result.

e. HRESULT GetTaskScoreName (

BSTR* pstr) **[out]**

This gets the string that describes the score that has been set for the task.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

f. HRESULT GetTimePeriodCount (

int* pCount) **[out]**

This gets the number of time periods that make up the task. A task instance can span multiple blocks of time.

Arguments:

i. pCount – Pointer to an int value that will be set to the result.

g. HRESULT GetTimePeriodTimes (

int nPeriod, **[in]**

VARIANT* pVarStartTime, **[out]**

VARIANT* pVarEndTime) **[out]**

This retrieves the start and end times for a single time period within the task.

Arguments:

i. nPeriod – 1-based index that identifies the time period of interest.

ii. pVarStartTime – Pointer to a VARIANT that will be set to a VT_I8 type. Its llVal will be set to the start time of the period expressed in 100 nanosecond units after the start of the recording.

iii. pVarEndTime – Pointer to a VARIANT that will be set to a VT_I8 type. Its llVal will be set to the end time of the period expressed in 100 nanosecond units after the start of the recording.

h. HRESULT GetNote (

BSTR* pstr) **[out]**

This gets the task note.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

i. HRESULT GetAttr (

```
                    int nAttrID,                    [in]
                    int* pAttr )                    [out]
```

This method retrieves a specific attribute of the task.

Arguments:

i. nAttrID – int value that defines the attribute. Currently supported values are:
   a. 0x00000001 – Task type (outputs same value as GetTaskID).
   b. 0x00000002 – Task score (outputs same value as GetTaskScoreID).
   c. 0x00000003 – Period count (outputs same value as GetTimePeriodCount).

ii. pAttr – Pointer to an int value that will be set to the result.

j. HRESULT GetDisplayPath (

```
                    BSTR* pstr )                    [out]
```

This method is deprecated.

k. HRESULT GetAudioNoteFileName (

```
                    BSTR* pstr )                    [out]
```

This gets the full path to the audio note that is associated with this task. Note that this path will change if a new audio note is recorded for the task.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

l. HRESULT SetTaskID (

```
                    UINT nType )                    [in]
```

This call sets the type of this task.

Arguments:

i. pType – UINT value that defines the task type.

m. HRESULT SetTaskScoreID (

```
                    int nType )                     [in]
```

This call sets the score of task.

Arguments:

i. pType – int value that defines the task score.

n. HRESULT GetTaskDuration (
VARIANT* pVarDuration) **[out]**

This gets the total duration of the task, summing up the duration of all time periods that make up the task.

Arguments:

i. pVarDuration – Pointer to a VARIANT that will be set to a type of VT_I8. The llVal will be set to the total duration of the task in 100 nanosecond units.

8. IMoraeEventSurveyQuestion (derives from IMoraeEvent)

This interface provides access to the responses to individual questions from a survey that presented to test participants.

Interface ID:

IID_IMoraeEventSurveyQuestion = B5A0AF63-59D4-4d4b-9133-2858F402BBF2

Data stream ID:

IID_SURVEYEVENTTYPE = 95CF0D9D-F231-4EF2-B41E-A0C8FBFF6533

Methods:

a. HRESULT GetStyle (
VARIANT* pVarStyle) **[out]**

This call requests the style settings of the survey question. This is a combination of bitwise flags defined below.

Arguments:

i. pVarStyle – Pointer to a VARIANT value that will be set to a type of VT_I4. Its lVal will be set to a bitwise combination of:
   a. 0x00000001 – Radio button style (only one selection allowed). This bit cannot be combined with the check box style.
   b. 0x00000002 – Check box style. The participant was able to select one or more checkboxes. This bit cannot be combined with the radio button style.
   c. 0x10000000 – Edit box style. The participant was able to enter text into an edit control.
   d. 0x01000000 – Horizontal style. The inputs for the user were laid out horizontally.

b. HRESULT GetQuestion (
BSTR* pstr) **[out]**

This gets the text of the survey question.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

c. **HRESULT GetChoices (**

SAFEARRAY pparChoices)** **[out]**

This gets the array of choices presented to the participant for the question.

Arguments:

i. pparChoices – Address of a SAFEARRAY pointer that will be set to a SAFEARRAY of BSTRs. The caller must free all resources.

d. **HRESULT GetChoiceCount (**

ULONG* pCount) **[out]**

This gets the number of choices that are displayed to the participant for a question that displays radio buttons or checkboxes.

Arguments:

i. pstr – Pointer to a ULONG variable that will be set to the count.

e. **HRESULT GetLeftTitle (**

BSTR* pstr) **[out]**

This gets the left label for a scale type survey question.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

f. **HRESULT GetRightTitle (**

BSTR* pstr) **[out]**

This gets the right label for a scale type survey question.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

g. **HRESULT GetEditAnswer (**

BSTR* pstr) **[out]**

This gets the text that the user entered for a survey question that displays a text entry field.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.

h. **HRESULT GetAnswers (**

SAFEARRAY ppArray)** **[out]**

This gets the array of answers that the participant selected for the question.

Arguments:

i. ppArray – Pointer to a SAFEARRAY variable that will be set to an array of VT_R8 values. The caller must free the resources.

9. IMoraeEventStreamSurvey (derives from IMoraeEventStream)

This stream provides access to a survey that was presented to test participants.

Interface ID:

IID_IMoraeEventStreamSurvey = 32CFA33C-6828-4aeb-B205-F76574D736E9

Data stream ID:

IID_SURVEYEVENTTYPE = 95CF0D9D-F231-4EF2-B41E-A0C8FBFF6533

Methods:

a. HRESULT GetStyle (

int* pStyle) **[out]**

This call requests the style settings of the survey. This can be one of the values defined below.

Arguments:

i. pStyle – Pointer to an int value that will be set to one of the following:
   a. 0x00000001 – SUS – This survey is a standard SUS questionaire.
   b. 0x00000002 – Custom survey.

b. HRESULT GetInstructions (

BSTR* pstr) **[out]**

This gets the instructional text displayed with the survey.

Arguments:

i. pstr – Pointer to a BSTR variable that will be set to the string. The caller must free the string.