Morae®
TechSmith®

Reading and Exposing Captured Events in Manager

Introduction:

For events that are captured during the recording process to be useful, they must be loaded into Manager and presented to the user in some meaningful way. Manager provides plug-ins with several opportunities for displaying data. These include drawing on the video during playback, allowing the user to search for events of interest, adding events to Manager's tree view and timeline display, and graphical display of event frequency.

Several interfaces are involved in the presentation of event data in Manager. They are described in this document:

1. IMoraeEventStream – This is implemented by a plug-in to read files created by a corresponding Recorder plug-in.

2. IMoraeEvent – This is the basic interface implemented by a plug-in to expose basic information about an individual event.

3. IMoraeEventSearch – This optional interface can be implemented by a plug-in to allow the user to search for particular events of interest and have them displayed in the form of a results table or a simple graph.

4. IMoraeSearchableEvent – If a plug-in supports IMoraeEventSearch, the events that are found in the search must implement this interface to allow proper display in the search results table.

5. IMoraeDisplayableStream – A plug-in can implement this optional interface to allow display of events in Manager's tree view and timeline. This may be appropriate for data streams that have a relatively small number of events.

6. IDisplayableItem – If a plug-in supports IMoraeDisplayableStream, it must also implement IDisplayableItem for each individual event in the data stream.

7. IMoraeObj – If a plug-in supports IMoraeDisplayableStream, it should also support IMoraeObj for each individual event. This allows Manager to properly notify application level extensions when the user selects an event in the tree view. (See Morae Manager Extensions documentation.)

8. IMoraeEditableStream – If a plug-in supports IMoraeDisplayableStream, it can optionally implement this interface if it wants to allow the user to edit events in the stream in Manager.

9. IMoraeEditableStream2 – If a plug-in supports IMoraeEditableStream, it should also support this interface. It replaces some deprecated methods in IMoraeEditableStream that are involved in undo / redo operations.

10. IMoraeVideoDraw – This optional interface allows the plug-in to draw on the main video during playback to provide a visual display of event data.

An individual plug-in can implement some or all of these interfaces, depending on how it wants to display event data to the user. You will note in the documentation below that the display interfaces such as IMoraeVideoDraw and IMoraeEventSearch do not derive from IMoraeEventStream. This allows development of plug-ins that use Morae's existing RRT data to display information in a specialized way to meet specific needs.

Manager discovers plug-ins in a couple of different ways. Plug-ins that implement IMoraeVideoDraw or IMoraeEventSearch must be registered on the system as implementing a specific category of object. (The category IDs are listed below with the interface documentation.) Other Plug-ins that implement IMoraeEventStream do not need to register as a particular category. When a session is recorded in Morae Recorder, it will ask all active event capture plug-ins for the CLSID of the object that Manager should use to read the event file(s) that the plug-in created. Manager will create an instance of that object and use its IMoraeEventStream interface to read the data file.

Interfaces:

1. IMoraeEventStream (derives from IUnknown)

A Plug-in must implement this interface which will be called by Morae Manager to read a data file that was created by an IEventCapture plug-in in Morae Recorder. The base functionality described in this interface simply reads the file(s) and allows creation of an enumerator that permits a caller to step through all of the captured events. Manager does not directly expose these events anywhere in the UI. To make these events accessible to the user, the object that implements IMoraeEventStream should also implement IMoraeEventSearch, IMoraeDisplayableStream, and / or IMoraeVideoDraw.

Objects that implement this interface need to have the CLSID registered with the system. The IMoraeEventStream interface is not associated with any implemented category in the registry. Note that the CLSID under which an object is registered should be the same as the GUID supplied by the event source in the IEventCapture::GetEventReaderInfo call (described in Capturing Event Data).

Interface ID:

IID_IMoraeEventStream = C8CC8AD7-F920-4167-A397-8FAC322D619D

Methods:

a. **HRESULT EnumEvents (**

IEnumMoraeEvent ppEnum)** **[out]**

This method requests an enumerator for all events in this stream for the current recording.

Arguments:

i. ppEnum – Pointer to a variable that holds a IEnumMoraeEvent pointer. (IEnumMoraeEvent is a standard IEnumXXX interface as documented in Morae Enumeration Interfaces.) The caller will call Release on this interface when it no longer needs it.

b. **HRESULT GetEventCount (**

ULONG* pCount) **[out]**

The plug-in should output the number of events that the enumerator acquired in EnumEvents contains.

Arguments:

i. pCount – Pointer to ULONG variable that the plug-in should set to the total number of events available in the current recording.

c. **HRESULT GetEventStreamID (**

IID* pID) **[out]**

This IID identifies the type of event in the stream. This should match the UUID that is output in the call to IMoraeEvent::GetEventTypeID for individual events from this stream.

Arguments:

i. pID – Pointer to an IID variable. It should be set to a unique ID that will be used to identify the data stream.

d. **HRESULT SetRecording (**

IMoraeRecording* pRecording) **[in]**

This provides the plug-in instance with an interface that can be used to get information about the recording in which it found. Note that the IMoraeRecording interface will not change during the object's lifetime.

Arguments:

i. pRecording – Pointer to an IMoraeRecording interface. This identifies the recording that the search applies to. (See Information Supplied by Manager for information on the methods that this interface exposes.) If the plug-in stores this interface pointer for later use, it should call AddRef on it. Each AddRef call must be balanced by a Release call when this interface is no longer needed. This pointer may be NULL to indicate that the plug-in should Release any stored IMoraeRecording pointer.

e. **HRESULT SetProject (**

IMoraeProject* pProject) [in]

This provides the plug-in object with an interface to the project in which its recording is found. (Note that the IMoraeProject interface relates to a Study rather than a project in the Manager UI.) This method may be called multiple times if the user drags a recording from one study to another in the Manager project. In that case, only the most current IMoraeProject interface pointer should be considered valid. If the plug-in stores this pointer for later use, it should call AddRef on it to ensure that it remains valid. In that case, the plug-in must call Release on this pointer when it is no longer using it.

Arguments:

i. pProject – Pointer to an IMoraeProject interface. This identifies the project in which this event stream's recording is found. (See Information Supplied by Manager for information on the methods that this interface exposes.) If the plug-in stores this interface pointer for later use, it should call AddRef on it. Each AddRef call must be balanced by a Release call when this interface is no longer needed. This pointer may be NULL to indicate that the plug-in should Release any stored IMoraeProject pointer.

f. **HRESULT ReadEventFile (**

BSTR bstrFile) [in]

The bstrFile will contain the full path to the event file that the plug-in should read. This may be called multiple times if the event capture object generated multiple files.

Arguments:

i. bstrFile – BSTR that contains the path. The plug-in should not store this BSTR. If it needs to retain the value it should make a copy of the string.

g. **HRESULT GetStreamName (**

LCID localeID, [in]
BSTR* pbstrName) [out]

This asks the plug-in for a name that could be displayed to the user.

Arguments:

i. localeID – LCID that specifies the language that the requested string should be in.
ii. pbstrName – Pointer to a BSTR value that should be set to the name of the data stream.

h. **HRESULT SetTempDirectory (**

BSTR bstrDir) [in]

This call directs the plug-in to a directory that it can use to store temporary data. The plug-in should clean up any files that it writes to this directory.

Arguments:

i. bstrDir – BSTR that provides the full path to the temp directory.

i. **HRESULT SetStartTime (**

VARIANT varTickCount, [in]
VARIANT varSysTime) [in]

This call is used to inform the plug-in of the start time for the recording in two different formats. It is intended to allow the plug-in to be able to offset any timestamps that were used internally when the data was captured to calculate the time into the recording at which each event occurred.

Arguments:

i. varTickCount – VARIANT that contains the tick count on the recording computer at the time that the recording started. Its vt is VT_I4.
ii. varSysTime – VARIANT that gives the system time of the recording computer at the time that the recording started. Its vt is VT_DATE.

2. IMoraeEvent (derives from IUnknown)

This interface is implemented by plug-ins for Morae Manager that either read event files created by Recorder event capture plug-ins or provide additional event search capability. The plug-in must implement this interface to provide basic information about an individual event. Events are accessed via an IEnumMoraeEvent interface. This enumerator can come from a call to IMoraeEventStream::EnumEvents or IMoraeEventSearch::RunSearch.

Interface ID:

IID_IMoraeEvent = 975F418C-AA02-468E-A398-A1BC1381B3FE

Methods:

a. **HRESULT GetTime (**

VARIANT* pvarElapsedTime, [out]
VARIANT* pvarTimeDate) [out]

This call requests the time at which the event occurred in two different formats.

Arguments:

i. pvarElapsedTime – Pointer to a VARIANT that should be set to the time of the event expressed as elapsed time into the video in 100 nanosecond units. The vt should be set to VT_I8. The llVal should be set to the elapsed time.

ii. pvarTimeDate – Pointer to a VARIANT that should be set to the date and time at which the event occurred. The vt should be set to VT_DATE. The date should be set to the OLEAUT DATE value that gives the date and time.

b. HRESULT GetTimeRange (

VARIANT* pVarStartTimes, [out]
VARIANT* pVarEndTimes) [out]

The variants should be set to SAFEARRAYs of type VT_I8. Each element of the array should indicate an elapsed time into the recording in 100-nanosecond units.

If the event represents a single time point, the pVarEndTimes should not be set. The pVarStartTimes should be set to a SAFEARRAY holding a single value.

If the event represents one or more blocks of time, the pVarStartTimes and pVarEndTimes should be set to SAFEARRAYS with equal numbers of elements.

Arguments:

i. pVarStartTimes – Pointer to a VARIANT that should be set to a SAFEARRAY of VT_I8 values. The vt of the VARIANT should be set to VT_ARRAY | VT_I8. The parray should be set to the pointer to the SAFEARRAY. The caller will dispose of this array.
ii. pVarEndTimes – Pointer to a VARIANT. If the event represents a single time point, this VARIANT's type should be left as VT_EMPTY. If the event represents one or more time spans, this VARIANT should be set to a SAFEARRAY similar to the pVarStartTimes argument, with the same number of elements as that argument. The caller will dispose of this array.

c. HRESULT GetAppTag (

int* pTag) [out]

Plug-ins are not required to support this call. If implemented, it should set the value of pTag to indicate the application that this event occurred in. For information on how to translate an application name to a tag, see the documentation of the IMoraeRecording interface in Information Supplied by Manager.

Arguments:

i. pTag – Pointer to an int value that should be set to the identifier of the application in which this event occurred.

d. **HRESULT GetWndClassTag (**

int* pTag) **[out]**

Plug-ins are not required to support this call. If implemented, pTag should be set to a value that indicates the class of window in which the event occurred. For information on how to translate a window class name to a tag, see the documentation of the IMoraeRecording interface in Information Supplied by Manager.

Arguments:

i. pTag – Pointer to an int value that should be set to the identifier of the window in which this event occurred.

e. **HRESULT GetHWND (**

VARIANT* pVar) **[out]**

Plug-ins are not required to support this call. If implemented, the pVar should be set to a VT_UI4 or VT_UI8 value that gives the HWND of the window in which this event occurred.

Arguments:

i. pVar = Pointer to a VARIANT that should be set to provide the HWND.

f. **HRESULT GetEventTypeID (**

IID* pID) **[out]**

The pID should be set to an IID value that uniquely identifies the type of event. It should be the same value as provided by the IMoraeEventStream::GetEventStreamID call. Manager uses these two calls to associate an individual event with its event stream source.

(For Morae's native event streams, the survey event is an exception to this. For this call, it outputs a unique IID value for the particular survey that it refers to.)

Arguments:

i. pID = Pointer to an IID value that should be set to a unique value that identifies the type of event.

3. IMoraeEventSearch (derives from IUnknown)

This interface allows a plug-in to provide the user with the ability to search for captured events of interest and display those events to the user in tabular or graphical form. The user will be able to export tabular search results to a file for further examination or analysis in other applications.

This interface can be implemented by a plug-in that also implements IMoraeEventStream to allow the user to access events from a non-standard event

stream that is captured by a Recorder plug-in. It can also be implemented by a plug-in that uses the events captured by multiple event streams to provide users with more specialized search capabilities than offered by the standard Morae RRT event streams.

Note: To be loaded by Manager and used for event searching, the plug-in must register its CLSID with the operating system and register itself as implementing the CATID_ManagerEventSearch category:

2D465CCB-9D4F-11DE-B70E-005056C00008

Interface ID:

IID_IMoraeEventSearch = A8827AE7-9D38-11DE-B70E-005056C00008

Methods:

a. HRESULT ShowSearchSetupDialog (

LCID localeID, [in]
VARIANT varHwndParent) [in]

This requests that the plug-in display a setup dialog to allow the user to configure search parameters. A return value of S_FALSE will indicate that the user has dismissed the dialog without changing the setup. If this method returns S_OK, Morae Manager will assume that the user has modified the search criteria.

Arguments:

i. localeID – LCID that indicates the language that the dialog text should be in.
ii. varHwndParent – VARIANT that provides the HWND of the window that the dialog should use as its owner. Plug-ins should check the vt of this VARIANT and be able to handle a signed or unsigned 32-bit or 64-bit number.

b. HRESULT SetSearchCriteria (

LPSAFEARRAY pCriteria) [in]

This will set the criteria to be used in performing a search. The criteria used in this call would have been acquired in a call to GetSearchCriteria. If this method is called before a call to ShowSearchSetupDialog, the settings in the dialog should reflect these settings. Note that the caller will de-allocate the SAFEARRAY after this method returns.

Arguments:

i. pCriteria – Pointer to a Windows SAFEARRAY structure that contains the search criteria.

c. HRESULT SetDefaultCriteria( )

This instructs the plug-in to use its default search criteria.

d. HRESULT GetSearchCriteria (

LPSAFEARRAY* ppCriteria) [out]

Manager will make this call to ask for search setup information that can be stored and applied to the plug-in in the SetSearchCriteria call.

Arguments:

i. ppCriteria – Address of a pointer to a SAFEARRAY structure. The plug-in allocates memory for the SAFEARRAY and populates the data. The caller will de-allocate the array resources. All elements of the array must be of the same fixed size. The datatype cannot be VT_DISPATCH, VT_UNKNOWN, or VT_BSTR.

e. HRESULT RunSearch (

IEnumMoraeEvent ppEnum, [out]**
ULONG* pFoundCount) [out]

This instructs the plug-in to run a search and provide an enumerator to give access to the search results.

Arguments:

i. ppEnum – Pointer to an IEnumMoraeEvent interface pointer that Manger can use to enumerate through all of the events that match the search criteria. (IEnumXXX interfaces are described in a Morae Enumeration Interfaces.) Manager will release this pointer.

ii. pFoundCount – Pointer to a ULONG variable that should be set to the number of events found in the search.

f. HRESULT GetSearchCriteriaString (

LCID localeID, [in]
BSTR* pstrCriteria) [out]

This method requests a brief string that describes the search criteria. This string will only be used to display information to the user.

Arguments:

i. localeID – LDIC that indicates the language that the string should be in.

ii. pstrCriteria – Pointer to a BSTR variable. The plug-in allocates this string. The caller frees it.

g. HRESULT SetSearchTimeRange (

VARIANT_BOOL bAllowAll, [in]
VARIANT varStartTimes, [in]
VARIANT varEndTimes) [in]

This will be used to set the time periods within the recording that should be included in the search. This method will be called before RunSearch is called. If bAllowAll is VARIANT_TRUE, the entire recording should be searched, and the remaining arguments can be ignored. Otherwise, varStartTimes and varEndTimes will each contain a SAFEARRAY of VT_I8 values. The varStartTimes and varEndTimes

arrays will have the same number of elements. Each start time should be paired with a corresponding end time to identify a segment of the recording.

Arguments:

i. bAllowAll – VARIANT_BOOL that will be set to VARIANT_TRUE if the entire recording should be searched. In this case the other inputs can be ignored. If bAllowAll is set to VARIANT_FALSE, varStartTimes and varEndTimes will set the search time periods.
ii. varStartTimes – VARIANT that contains a SAFEARRAY of VT_I8 values. Each value in this array represents the start time of a segment of the recording to be searched. The time is in 100 nanosecond units from the start of the recording.
iii. varEndTimes – VARIANT that contains a SAFEARRAY of VT_I8 values. The number of elements in this array will equal the number of elements in the varStartTimes array. Each value represents the end time a segment of the recording, corresponding to each start time in varStartTimes.

h. HRESULT SetSearchApplications (

VARIANT_BOOL bAllowAll, [in]
VARIANT varApps) [in]

When the user runs a search in Manager, he can restrict the search to find events that occurred in specific applications. This may not be applicable to all event types. In that case, the plug-in may disregard this call, but should still return a success code such as S_OK.

Arguments:

i. bAllowAll – VARIANT_BOOL value that indicates whether or not the search should be restricted to events that occurred in particular applications. If bAllowAll is set to VARIANT_TRUE, varApps can be ignored.
ii. varApps – VARIANT that contains a SAFEARRAY of VT_I4 values. Each of these values is an "application tag" that identifies an application. These values can be translated into application names by calling the IMoraeRecording::GetAppName method described in Information Supplied by Manager.

i. HRESULT GetItemAttributeCount (

int* pCount) [out]

This method is used by Manager to determine how to set up columns in the search results table for the display of search results. The plug-in should set the count to indicate how many columns of data that it needs to display the details of each event. Note that the time should not be counted as a column. For each attribute, Manager will make calls to GetItemAttributeName and SetItemAttributeDisplayColID as described below. This attribute count is assumed to be independent of the search criteria and of the particular events that are found in a given search.

Arguments:

i. pCount – Pointer to an int variable that should be set to the number of display columns needed.

j. HRESULT GetItemAttributeName (

LCID localeID,	**[in]**
int nIndex,	**[in]**
BSTR* pbstrName)	**[out]**

This call will be made for each item attribute (as determined in the GetItemAttributeCount method) to ask for a name to be used in the column header for each item attribute. The localeID indicates the language that should be used. The nIndex is a 1-based index into the array of attributes. pbstrName should be set to the address of a BSTR that contains the string that will be used as the column header.

Arguments:

i. localeID – LDIC that indicates the language that the string should be in.
ii. nIndex – 1-based index indicating which column header is requested.
iii. pbstrName – Pointer to a BSTR variable. The plug-in allocates this BSTR. The caller will de-allocate it.

k. HRESULT SetItemAttributeDisplayColID (

int nIndex,	**[in]**
int nColID)	**[in]**

This call is used to map a column in the search results table onto each attribute of the events found in the search. The nIndex input is a 1-based index into the array of item attributes for each event. The nColID is an arbitrary ID number. It will be used in calls to IMoraeSearchableEvent::GetAttributeString to request information to be displayed in the search results table for each event.

Arguments:

i. nIndex – 1-based index indicating which attribute this call applies to.
ii. nColID – An arbitrary int value that should be mapped to the attribute indicated by nIndex.

l. HRESULT SetRecording (

IMoraeRecording* pRecording)	**[in]**

This provides an interface to the Recording that the current instance of the plug-in can search within.

Arguments:

i. pRecording – Pointer to an IMoraeRecording interface. This identifies the recording that the search applies to. (See Information Supplied by Manager for information on the methods that this interface exposes.) If the plug-in stores this interface pointer for later use, it should call AddRef on it. Each AddRef call must be balanced by a Release call when this interface is no

longer needed. This pointer may be NULL to indicate that the plug-in should Release any stored IMoraeRecording pointer.

m. HRESULT SetProject (

IMoraeProject* pProject) [in]

This provides an interface to the project in which the recording is found. Note that the IMoraeProject interface corresponds to a study in the Morae Manager user interface. This method may be called multiple times if the user drags a recording from one study to another in the Manager project. In that case, only the most current IMoraeProject interface pointer should be considered valid.

Arguments:

i. pProject – Pointer to an IMoraeProject interface. This identifies the project in which the search recording is found. (See Information Supplied by Manager for information on the methods that this interface exposes.) If the plug-in stores this interface pointer for later use, it should call AddRef on it. Each AddRef call must be balanced by a Release call when this interface is no longer needed. This pointer may be NULL to indicate that the plug-in should Release any stored IMoraeProject pointer.

n. HRESULT GetName (

LCID localeID, [in]
BSTR* pbstrName) [out]

This requests the name that will be displayed to the user to identify this plug-in.

Arguments:

i. localeID – LDIC that indicates the language that the string should be in.
ii. pbstrName – Pointer to a BSTR value that should be set to the name of the plug-in. The plug-in will allocate resources for this string. The caller will de-allocate.

o. HRESULT SetTempDirectory (

BSTR bstrDir) [in]

This provides a path to a directory that the plug-in may use to store temporary files if necessary. Any files created in this directory must not be used to store persistent data.

Arguments:

i. bstrDir – BSTR that contains the full path to the directory that the plug-in may use for temporary data storage. The plug-in is responsible for deleting any files that it creates when they are no longer needed.

p. HRESULT GetSearchTypeID (

IID* pID) [out]

This requests a unique identifier for the plug-in type. It can be, but is not necessarily the same value as the CLSID of the plug-in object. Manager will use this ID value to match up plug-in types with search criteria that it stores off for later retrieval.

Arguments:

i. pID – Pointer to an IID variable. The plug-in should set this to a unique identifier.

q. HRESULT HasSetupDialog (

VARIANT_BOOL* pBool) [out]

Manager will call this method to determine whether the plug-in has a setup dialog to display.

Arguments:

i. pBool – Pointer to a VARIANT_BOOL variable. The plug-in should set this value to VARIANT_TRUE if it supports the ShowSearchSetupDialog method.

4. IMoraeSearchableEvent (derives from IMoraeEvent)

This interface provides additional information about an event that was found in a search (from a call to IMoraeEventSearch::RunSearch). This information is necessary for display of events in the search results table and chart views. Any plug-in that implements IMoraeEventSearch must implement IMoraeSearchableEvent for each of the events found in a search.

Interface ID:

IID_IMoraeSearchableEvent = 62925057-535C-4EEE-9685-3010ED75330E

Methods:

a. HRESULT GetAttributeString (

LCID localeID, [in]
int nColID, [in]
BSTR* pbstrAttribute) [out]

This call requests a string that can be displayed to the user that describes some attribute of the event. The localeID input indicates the language that Manger is using. The nColID input is the value that was set in the IMoraeEventSearch::SetItemAttributeDisplayColID method call to map a column ID to an attribute index for the search plug-in.

Arguments:

i. localeID – LDIC that indicates the language that the string should be in.
ii. nColID – Int value that identifies the display column.

iii. pbstrAttribute – Pointer to a BSTR variable that should be set to the string that will be displayed in the UI. The caller will be responsible for freeing this BSTR.

b. HRESULT GetItemRectangle (

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

This call asks where the event occurred on the main video within the recording. Implementation of this method is optional. Not all event types are expected to have a location. The units are in pixels relative to the recorded video area.

Arguments:

i. pLeft = Pointer to an int value that should be set to the left side of the event's bounding rectangle (expressed in screen pixel coordinates).
ii. pTop = Pointer to an int value that should be set to the top of the event's bounding rectangle (expressed in screen pixel coordinates).
iii. pRight = Pointer to an int value that should be set to the right side of the event's bounding rectangle (expressed in screen pixel coordinates).
iv. pBottom = Pointer to an int value that should be set to the bottom of the event's bounding rectangle (expressed in screen pixel coordinates).

5. IMoraeDisplayableStream (derives from IMoraeEventStream)

This interface can be implemented by a plug-in to add events to Manager's tree view and timeline displays. Events that are displayed in this way can be selected in the tree view. A context menu can optionally be shown when the user right-clicks on the item. If the events are editable, the timeline display can allow the user to directly manipulate the timing and duration of individual events.

Interface ID:

IID_IMoraeDisplayableStream = 4F5D5AEE-2B60-4543-AE2F-253C0581F6AB

Enumerations:

a. TimelineDisplayType

This is used in the GetTimelineDisplayType method to specify how the events from this stream will be displayed in the timeline of the player in Manager. Current valid values are:

i. TimelineDisplay_None – Events from this stream are not displayed in the timeline.
ii. TimelineDisplay_Span – Events from this stream are displayed as one or more spans of time.
iii. TimelineDisplay_Point – Events from this stream are displayed as individual points in the timeline.

Methods:

a. **HRESULT GetMainFolder (**
IDisplayableItem ppItem) [out]**
This call requests an interface to the main folder for the events from this stream. In order to be displayed in the tree view, events must belong to a main folder item that will be located under the recording node in the tree view. (The IDisplayableItem interface is described below.)

Arguments:

i. ppItem – Pointer to a variable that holds a pointer to an IDisplayableItem interface. Morae will use this interface to get the display attributes of the folder itself and to get an enumerator to its subitems. Manager will call Release on this pointer when it no longer needs it.

b. **HRESULT GetTimelineDisplayType (**
TimelineDisplayType* pType) [out]
This call asks the stream how its items will be displayed in the timeline. This type will apply to all events from the stream.

Arguments:

i. pType – Pointer to an TimelineDisplayType value that the plug-in should set to one of the values listed above.

c. **HRESULT GetTimelineFilter (**
CLSID* pclsidSearch) [out]
If a stream's timeline display type is TimelineDisplay_Point, the plug-in can provide the ability to allow the user to filter the displayed events. This allows the easily identify events of interest and navigate to them in the video.

Arguments:

i. pclisdSearch – Pointer to a CLSID. This should be set to the CLSID of a plug-in that implements IMoraeEventSearch that can be used to filter the timeline display.

6. IDisplayableItem (derives from IUnknown)

This interface must be implemented for individual events in a stream that implements IMoraeDisplayableStream for display in Manager's tree view and timeline. In the tree view, events can be organized into folders. Drag-and-drop can be supported to allow the user to re-organize the display of events. Events can optionally be displayed on the timeline as well as the tree view.

Interface ID:

IID_IDisplayableItem = B9D7F672-A20E-4FB8-A24A-25581158BCC3

Enumerations:

a. UI_ElementType

This is used to identify the UI context for requests made to the item for display information. It is used in the GetHIcon and GetTooltipText methods. Current valid values are:

i. UI_Element_Treeview_Analyze – The call is made from the tree view in Manager's Analyze tab.
ii. UI_Element_Treeview_Present - The call is made from the tree view in Manager's Present tab.
iii. UI_Element_Treeview_Graph - The call is made from the tree view in Manager's Graph tab.
iv. UI_Element_Timeline - The call is made from the timeline of Manager's media player. The tab is not specified.
v. UI_Element_TimelineFiltered - The call is made from the timeline of Manager's media player. The item has been filtered by the user. In other words, the user has run a search for items of interest. This item was not included in the search results. The tab is not specified.
vi. UI_Element_TimelineSelected - The call is made from the timeline of Manager's media player. The item has been selected by the user by positioning the cursor over the item.

b. DisplayItemOp

This is used to identify operations that the user may want to perform. It is used in the IsOperationAllowed method to determine whether to enable or disable particular commands in the Manager UI when an item is selected. Current valid values are:

i. ItemOp_TreeviewDrag – Identifies the initiation of a drag operation in the tree view.
ii. ItemOp_EditName – Identifies the action of renaming an item in the tree view by clicking on an item that is already selected.
iii. ItemOp_Delete – Identifies the action of deleting an item in the tree view.
iv. ItemOp_AddSubfolder – Identifies the action of adding a subfolder to an item in the tree view.
v. ItemOp_TimeChange – Identifies the action of modifying the time of an event. This can be applied to an event that represents a single time point or a collection of time spans.
vi. ItemOP_TimeAddOrRemoveSegments – Identifies the action of adding or removing time spans. This only applies to an event that represents one or

more time spans. It can be used to ask an event whether the user is allowed to remove a time span from the middle of an existing span, for example.

vii. ItemOp_ShowEditDlg – Identifies the action of displaying an edit dialog for the event.

c. DisplayItemMenuAttr

This enumeration is used to identify attributes of an item in a context menu. It is used in the GetContectMenuItemAttribute method to determine how the item should be displayed in the context menu. Current valid values are:

i. ItemMenuAttr_HasSubmenu –Used to ask whether this menu item has a sub-menu.

ii. ItemMenuAttr_IsChecked – Used to ask whether a particular menu item should be checked.

iii. ItemMenuAttr_IsSeparator – Used to determine whether a menu item is a separator or a normal menu item.

iv. ItemMenuAttr_IsDisabled – Used to determine whether a menu item is enabled or disabled.

Methods:

a. HRESULT GetItemName (

BSTR* pStrName, [out]
LCID localeID) [in]

This call requests the name of the item as it will be displayed in the tree view.

Arguments:

i. pStrName – Pointer to a BSTR variable that the plug-in should set to a system string that it allocates. The caller will de-allocate the system resources.

ii. localeID – An LCID that can be used to identify the language that should be used.

b. HRESULT GetItemDetails (

BSTR* pStrDetails, [out]
LCID localeID) [in]

When an item in Manager's tree view is selected, details about the item appear in the Details pane below it. This method requests the string to be shown in the Details pane.

Arguments:

i. pStrDetails – Pointer to a BSTR variable that the plug-in should set to a system string that it allocates. The caller will de-allocate the system resources.

ii. localeID – An LCID that can be used to identify the language that should be used.

c. **HRESULT GetHIcon (**

UI_ElementType type, **[in]**
HICON* pIcon) **[out]**

This call requests a Windows icon that is used to display the item in the tree view (or on the timeline in the case of an event that represents an individual time point).

Arguments:

i. type – Enumerated type (described above) that indicates which UI element this method is requesting an icon for.
ii. pIcon = Pointer to Windows HICON value. The plug-in creates this icon. The caller is responsible for destroying it. The plug-in should create an icon each time this method is called because Morae will destroy it after it draws it onto the screen.

d. **HRESULT EnumSubItems(**

IEnumDisplayableItem ppEnum)** **[out]**

This call asks for an interface that can be used to enumerate through the subitems that should be displayed under this item in the tree view. The IEnumDisplayableItem is one of the Morae enumeration interfaces described in Morae Enumeration Interfaces.

Arguments:

i. ppEnum – Address of an IEnumDisplayableItem pointer that the plug-in should set to a valid enumerator. The caller will call Release on this interface when it is no longer needed.

e. **HRESULT IsOperationAllowed (**

DisplayItemOp nOperation, **[in]**
VARIANT_BOOL* pBool) **[out]**

This call determines whether an operation is allowable on a particular event. It is used by Manager to determine whether to enable or disable particular UI operations when an event is selected.

Arguments:

i. nOperation – DisplayItemOp variable that identifies the operation. Possible values are listed above.
ii. pBool - Pointer to a VARIANT_BOOL variable that should be set to either VARIANT_TRUE or VARIANT_FALSE.

f. **HRESULT IsDropAllowed (**

IDisplayableItem* pDragItem, **[in]**
VARIANT_BOOL* pBool) **[out]**

When a drag-drop operation is in progress in the tree view, this call is made whenever the item being dragged is moved over another item. This method is called on the potential drop target to determine whether the drop operation should be allowed.

Arguments:

i. pDragItem – Pointer to an IDisplayableItem interface for the item being dragged.
ii. pBool - Pointer to a VARIANT_BOOL variable that should be set to either VARIANT_TRUE or VARIANT_FALSE to indicate whether the drop operation should be allowed.

g. HRESULT DropItem (

IDisplayableItem* pDragItem) [in]

This call indicates that the user has completed a drag-and-drop operation, dropping another item on this one.

Arguments:

i. pDragItem – Pointer to an IDisplayableItem interface that represents the item that is being dropped onto the callee.

h. HRESULT SetItemName (

BSTR bstrName) [in]

This call sets the name of the item in response to user input.

Arguments:

i. bstrName –BSTR variable that contains the new name for the item. If the plug-in needs to store this string, it must make its own copy. The caller will de-allocate resources for this string after this method returns.

i. HRESULT DeleteItem ( )

This call deletes the current item in response to user input.

j. HRESULT ShowEditDialog (

VARIANT varHwndParent, [in]
LCID localeID) [in]

This call requests that the plug-in display an edit dialog to allow the user to modify the attributes of this item.

Arguments:

i. varHwndParent – VARIANT that holds the HWND of the main window of Manager. Its vt will be set to VT_UI4 or VT_UI8.
ii. localeID – An LCID that can be used to identify the language that should be used.

k. HRESULT AddSubfolder (

IDisplayableItem ppNewFolder,**	**[out]**
LCID localeID)	**[in]**

This call asks the plug-in to create a subfolder under the current item.

Arguments:

i. ppNewFolder – Address of an IDisplayableItem pointer that the plug-in should set to a new subfolder that it creates. The caller is responsible for calling Release on this pointer.
ii. localeID – An LCID that can be used to identify the language that should be used.

l. HRESULT GetTooltipText (

UI_ElementType type,	**[in]**
BSTR* pStrTip,	**[out]**
LCID localeID)	**[in]**

This call requests text that will be displayed in a tooltop when the user hovers the cursor over an item.

Arguments:

i. type – UI_ElementType value that identifies the context of the tooltip request.
ii. pStrTip - Pointer to a BSTR variable that the plug-in should set to a system string that it allocates. The caller will de-allocate the system resources.
iii. localeID – An LCID that can be used to identify the language that should be used.

m. HRESULT GetContextMenuItemIDs (

int nOwnerID,	**[in]**
SAFEARRAY ppArray)**	**[out]**

This call requests an array of command IDs for items to be displayed in a context menu when the user right-clicks on a displayable item. These IDs can represent individual commands or owners of submenus.

Arguments:

i. nOwnerID – int value that can either be one of the values listed above in the UI_ElementType enumeration or the ID of a menu item that owns a submenu, from a previous call.
ii. ppArray – Address of a SAFEARRAY pointer that should be set to a SAFEARRAY of type VT_I4. All of the values in the array should be positive integers. Each can represent an individual command or the ID of a menu item that owns a submenu. These ID values will be used in subsequent calls to get the attributes of each item and to indicate when the user has selected a command. The caller will de-allocate this array.

n. **HRESULT GetContextMenuItemText (**

int nID,	**[in]**
BSTR* pStrName,	**[out]**
LCID localeID)	**[in]**

This call requests the text of a context menu item.

Arguments:

i. nID – int value that identifies the menu item. This ID came from a previous call to GetContextMenuItemIDs.
ii. pStrName – Pointer to a BSTR variable that the plug-in should set to a system string that it allocates. The caller will de-allocate the system resources.
iii. localeID – An LCID that can be used to identify the language that should be used.

o. **HRESULT GetContextMenuItemAttribute (**

int nID,	**[in]**
DisplayItemMenuAttr nAttribute,	**[in]**
VARIANT* pAttr)	**[out]**

This call requests an attribute of the menu item. Manager uses these attributes to determine how to display the menu item.

Arguments:

i. nID – int value that identifies the menu item. This ID came from a previous call to GetContextMenuItemIDs.
ii. nAttribute – DisplayItemMenuAttr value (enumerated values listed above) that specifies the attribute requested.
iii. pAttr – Pointer to a VARIANT that the receives the value from the plug-in. The VARIANT type expected depends on the value of nAttribute:
    a. ItemMenuAttr_HasSubmenu – vt = VT_BOOL
    b. ItemMenuAttr_IsChecked – vt = VT_BOOL
    c. ItemMenuAttr_IsSeparator – vt = VT_BOOL
    d. ItemMenuAttr_IsDisabled – vt = VT_BOOL

p. **HRESULT ExecuteContextMenuCommand (**

int nID,	**[in]**
VARIANT varHwndParent,	**[in]**
LCID localeID)	**[in]**

This method is called to indicate that the user has selected a command from a context menu. The HWND of Manager's main window and an LCID are included as arguments in this call in case the plug-in needs to show a dialog in executing this command.

Arguments:

i. nID – int value that identifies the menu item. This ID came from a previous call to GetContextMenuItemIDs.

ii. varHwndParent – VARIANT that holds the HWND of the main window of Manager. Its vt will be set to VT_UI4 or VT_UI8.
iii. localeID – An LCID that can be used to identify the language that should be used.

q. HRESULT GetItemTimelineColor (
unsigned int* pColor) [out]

This applies to items from displayable item streams with a timeline display type of TimelineDisplay_Span, indicating that the individual events encompass one or more spans of time. Time spans on the timeline are tinted by the color output from this method.

Arguments:
i. pColor – Pointer to an unsigned int. This value will be cast to a Windows COLORREF value.

r. HRESULT GetOwnerStream (
IMoraeDisplayableStream ppStream) [out]**

This call requests a pointer to the event stream that owns this event.

Arguments:
i. pStream – Address of an IMoraeDisplayableStream interface. The caller will call Release on this pointer.

7. IMoraeObj (derives from IUnknown)

Any event that implements IDisplayableItem should also implement IMoraeObj. Whenever an item in the tree view is selected, modified, or deleted, any plug-ins that are registered to listen for application-level events receive notification that includes a pointer to the item's IMoraeObj interface. (See documentation on Morae Manager Extensions.) This interface provides methods to determine what type of object this is and to get its "owner", allowing the plug-in receiving notification to "walk" up the hierarchy of object displayed in the tree view.

Interface ID:

IID_IMoraeObj = AB220340-FA69-4097-83D6-858620DE9678

Methods:

a. HRESULT GetObjName (
BSTR* pbstrName) [out]

This call requests the name of the object. Note that the language is not specified here because objects in the tree view typically have names that are defined by the user.

Arguments:

i. pbstrName – Pointer to a BSTR variable that the plug-in should set to a system string that it allocates. The caller will de-allocate the system resources.

b. HRESULT GetObjType (

IID* piidType) [out]

This call requests an IID that identifies the type of the object. Typically, the object will return the IID of a more specialized interface that this object implements. Examples are IID_IMoraeProject and IID_IMoraeRecording.

Arguments:

i. ppiidType – Pointer to an IID variable that the plug-in should set to the IID of an interface that it implements.

c. HRESULT GetObjOwner (

IMoraeObj ppMoraeObj) [out]**

This call requests the IMoraeObj interface of the owner of this object.

Arguments:

i. ppMoraeObj – Address of a pointer to an IMoraeObject. The caller will call Release on this pointer.

8. IMoraeEditableStream (derives from IMoraeEventStream)

This optional interface can be implemented by any event stream that allows the user to create, modify, or delete events. None of these changes should be saved to disk until the SaveEdits method is called. Note that several methods in this interface are deprecated, having been replaced by method calls in other interfaces.

Interface ID:

IID_IMoraeEditableStream = 6995B1C2-80CC-469f-9C3C-E50C60CDAD9F

Methods:

a. HRESULT SetModifiedFileDirectory (

BSTR bstrDir) [in]

This method supplies the full path to a directory that the plug-in can use to store modified files. This is not a directory for temporary files. It is intended to be the directory that the plug-in uses for files that store modifications made by the user to events in this stream.

Arguments:

i. bstrDir –BSTR variable that provides the full path to a folder into which the plug-in can place any files it needs to store data.

b. HRESULT CreateNewEvent (

VARIANT varTimeStart,	**[in]**
VARIANT varTimeEnd,	**[in]**
IMoraeEvent ppEvent)**	**[out]**

This method asks the stream to create a new event with the specified start and end times. The end time can be disregarded if it is not applicable. A pointer to the IMoraeEvent interface of the new event is output.

Arguments:

i. varTimeStart – VARIANT that contains a SAFEARRAY of one or more 64-bit integers. The vt of the VARIANT will be set to VT_ARRAY | VT_I8. Each element will represent the time into the video in 100 nanosecond units.
ii. varTimeEnd – VARIANT that contains a SAFEARRAY of one or more 64-bit integers. The vt of the VARIANT will be set to VT_ARRAY | VT_I8. Each element will represent the time into the video in 100 nanosecond units.
iii. ppEvent – Address of a pointer to an IMoraeEvent interface. This should be set to a pointer to the event that is created as a result of this call. Manager will call Release on this interface.

c. HRESULT DeleteEvent (

IMoraeEvent* pEvent)	**[in]**

This method instructs this stream to delete this event.

Arguments:

i. pEvent – Pointer to an IMoraeEvent interface that identifies the event to be deleted from the stream.

d. HRESULT EditEvent (

LCID localeID,	**[in]**
VARIANT varWndParent,	**[in]**
SAFEARRAY* ppArrayPlayerTimes,	**[in]**
IMoraeEvent* pEvent)	**[in]**

This method requests the plug-in to display a dialog allowing the user to edit the event.

Arguments:

i. localeID – An LCID that can be used to identify the language that should be used.
ii. varWndParent – VARIANT that contains the HWND of the main window of Manager. The plug-in should be able to accept types of VT_UI4, VT_I4, VT_UI8, or VT_I8.
iii. pArrayPlayerTimes – Pointer to a SAFEARRAY with a data type of VT_I8. This array will contain two values – the mark in point and the mark out point timeline controls.
iv. pEvent = Pointer to the IMoraeEvent interface of the event to be edited.

e. HRESULT SaveEdits ()

This method commands the plug-in to save all modifications to the event stream to disk.

f. HRESULT IsEditPending (

VARIANT_BOOL* pBool) [out]

This method is deprecated. GetPendingUndoCount and GetPendingRedoCount in the IMoraeEditableStream2 interface are used instead of this method to determine whether there are any pending operations.

g. HRESULT UndoLastEdit ()

This method is deprecated. IMoraeEditableStream2::UndoLastOperation has replaced it.

h. HRESULT RedoLastUndoneEdit ()

This method commands the plug-in to redo the last operation that was previously undone.

i. HRESULT SetUndoLevelLimit (

int* pLimit) [in, out]

This method suggests the number of undo operations that should be supported by the plug-in. If the plug-in is unable to support this number of operations, it should set the pLimit value to the maximum number of operations that it can support.

Arguments:

i. pLimit – Pointer to an int variable that suggests the number of operations that this plug-in should support. Upon successful return, this value should be set to the number of operations that are actually supported by the plug-in.

j. HRESULT SetRedoLevelLimit (

int* pLimit) [in, out]

This method suggests the number of redo operations that should be supported by the plug-in. If the plug-in is unable to support this number of operations, it should set the pLimit value to the maximum number of operations that it can support.

Arguments:

i. pLimit – Pointer to an int variable that suggests the number of operations that this plug-in should support. Upon successful return, this value should be set to the number of operations that are actually supported by the plug-in.

k. HRESULT GetEventDisplayName (

LCID localeID,	**[in]**
IMoraeEvent* pEvent,	**[in]**
BSTR* pbstrName)	**[out]**

This method is deprecated. The IMoraeDisplayableStream and IDisplayableEvent interfaces should be implemented by plug-ins that wish to display events in Manager's tree view or timeline.

l. HRESULT GetEventDisplayImage (

IMoraeEvent* pEvent,	**[in]**
VARIANT* pVarImage)	**[out]**

This method is deprecated. The IMoraeDisplayableStream and IDisplayableEvent interfaces should be implemented by plug-ins that wish to display events in Manager's tree view or timeline.

m. HRESULT SetEventDisplayName (

IMoraeEvent* pEvent,	**[in]**
BSTR bstrDir)	**[in]**

This method is deprecated. IDisplayableItem::SetItemName now takes its place.

n. HRESULT SetEventDisplayPath (

IMoraeEvent* pEvent,	**[in]**
BSTR bstrPath)	**[in]**

This method is deprecated. Any plug-in that wants to support the ability of the user to modify its display path in the tree view should support drag-and-drop operations via the IDisplayableItem interface.

o. HRESULT SetEventTimes (

IMoraeEvent* pEvent,	**[in]**
VARIANT varTimeStart,	**[in]**
VARIANT varTimeEnd)	**[in]**

This method sets new times for an event as a result of user interaction with the representation of an event that also implements IDisplayableItem in the timeline.

Arguments:

i. pEvent – Pointer to the IMoraeEvent interface of the event to be modified.
ii. varTimeStart – VARIANT that contains one or more start times for the event in the form of a SAFEARRAY. The vt of this VARIANT will be VT_ARRAY | VT_I8. Values will be start times for all time segments of the event.
iii. varTimeStart – VARIANT that contains one or more end times for the event in the form of a SAFEARRAY. The vt of this VARIANT will be VT_ARRAY | VT_I8. Values will be end times for all time segments of the event.

9. IMoraeEditableStream2 (derives from IMoraeEditableStream)

This optional interface can be implemented by any event stream that allows the user to create, modify, or delete events. It supports more advanced undo / redo capabilities than IMoraeEditableStream.

Interface ID:

IID_IMoraeEditableStream2 = 62844542-64DF-435b-B44B-9719CC5F0B46

Methods:

a. HRESULT UndoLastOperation (

VARIANT_BOOL bRedoable, [in]
int* pRemainingUndoCount) [out]

This method instructs the plug-in to undo the previous operation.

Arguments:

i. bRedoable – VARIANT_BOOL value that indicates whether this operation should be reversible by a subsequent IMoraeEditableStream::RedoLastUndoneEdit call.
ii. pRemainingUndoCount – Pointer to an int value that should be set to the number of available undoable operations remaining when this method returns.

b. HRESULT GetPendingUndoCount (

int* pRemainingCount) [out]

This method asks the plug-in how many undo operations are currently available.

Arguments:

i. pRemainingCount – Pointer to an int that should be set to the number of undoable operations that are pending.

c. HRESULT GetPendingRedoCount (

int* pRemainingCount) [out]

This method asks the plug-in how many redo operations are currently available.

Arguments:

i. pRemainingCount – Pointer to an int that should be set to the number of redoable operations that are pending.

d. HRESULT ClearPendingUndoOperations ()

This method instructs the plug-in to clear all pending undo operations.

e. HRESULT ClearPendingRedoOperations ()

This method instructs the plug-in to clear all pending redo operations.

f. HRESULT GetPendingUndoOpDescription (

int nIndex,	**[in]**
LCID localeID,	**[in]**
BSTR* pStrOp)	**[out]**

This method asks the plug-in for a string that describes a pending undo operation.

Arguments:

i. nIndex – 1-based index into the array of undoable operations for the event stream. Note that the last operation added should be at index "1".
ii. localeID – LCID that sets the language that the string should be in.
iii. pStrOp – Pointer to a BSTR variable. The plug-in should allocate system resources for this string. The caller will de-allocate resources.

g. HRESULT GetPendingRedoOpDescription (

int nIndex,	**[in]**
LCID localeID,	**[in]**
BSTR* pStrOp)	**[out]**

This method asks the plug-in for a string that describes a pending redo operation.

Arguments:

i. nIndex – 1-based index into the array of undoable operations for the event stream. Note that the last operation added should be at index "1".
ii. localeID – LCID that sets the language that the string should be in.
iii. pStrOp – Pointer to a BSTR variable. The plug-in should allocate system resources for this string. The caller will de-allocate resources.

10. IMoraeVideoDraw (derives from IUnknown)

This interface can be implemented by a plug-in to draw on the main video as a recording is played back in Manager. Events can be portrayed on the video for easy visualization.

A plug-in that implements this interface can display information about custom events captured by a Recorder plug-in (in which case it would also implement IMoraeEventStream) or it can use any of Morae's existing RRT data.

Note: To be loaded and used by Manager, the plug-in must register its CLSID with the operating system and register itself as implementing the CATID_ManagerVideoDraw category:

5A245B48-94A3-4A37-9468-57F3807348E6

Interface ID:

IID_IMoraeVideoDraw = E9215EFD-F193-47ef-A2C7-B8C0D0505470

Enumerations:

a. RenderingHints

This is used in the GiveUpcomingFrame method to specify whether rendering is subject to time constraints of live video playback. Current valid values are:

i. RenderingNotRealtime – If this bit is set, the video is not being played live, so time is not critical.

Methods:

a. HRESULT GetName (

BSTR* pStr, [out]
long localeID) [in]

This requests the name that will be displayed to the user to identify this plug-in. The localeID should be used to determine the language. The plug-in must allocate the system resources for the BSTR. The caller will be responsible for de-allocation of recources.

Arguments:

i. pStr – Pointer to a BSTR value that should be set to the name of the plug-in.
ii. localeID – LCID that specifies the language that the requested string should be in.

b. HRESULT GetDescription (

BSTR* pStr, [out]
long localeID) [in]

This requests a description of what this plug-in will do. If this plug-in implements a setup dialog that allows the user to modify its behavior, the description can be used to indicate the current settings. The localeID should be used to determine the language. The plug-in must allocate the system resources for the BSTR. The caller will be responsible for de-allocation of recources.

Arguments:

i. pStr – Pointer to a BSTR value that should be set to the description of the plug-in.
ii. localeID – LCID that specifies the language that the requested string should be in.

c. HRESULT HasSetupDialog (

VARIANT_BOOL* pBool) [out]

Manager will call this method to determine whether the plug-in has a setup dialog to display. If the plug-in sets pBool to VARIANT_TRUE, it must be able to show a dialog in the ShowSetupDialog method call. Otherwise, ShowSetupDialog will not be called.

Arguments:

i. pBool – Pointer to a VARIANT_BOOL variable. The plug-in should set this value to VARIANT_TRUE if it supports the ShowSetupDialog method.

d. HRESULT ShowSetupDialog (

VARIANT varHwndParent, [in]
LCID localeID) [in]

This requests that the plug-in display a setup dialog to allow the user to configure draw parameters. The localeID input should determine the language that the dialog should employ. The varHwndParent input gives the HWND that the dialog should use as a parent window. A return value of S_FALSE will indicate that the user has dismissed the dialog without changing the setup. If this method returns S_OK, Morae Manager will assume that the user has modified the draw criteria.

Arguments:

i. varHwndParent – VARIANT that provides the HWND of the window that the dialog should use as its owner. Plug-ins should check the vt of this VARIANT and be able to handle a signed or unsigned 32-bit or 64-bit number.
ii. localeID – LCID that indicates the language that the dialog text should be in.

e. HRESULT SetRecording (

IMoraeRecording* pRecording) [in]

This provides an interface to the Recording that the current instance of the plug-in can draw on. See the documentation on the IMoraeRecording interface for information on the methods that can be called on this interface. If the plug-in stores this interface pointer, it should call AddRef on it. Each AddRef call should be balanced by a Release call when the plug-in no longer needs this interface.

Arguments:

i. pRecording – Pointer to an IMoraeRecording interface. (See Information Supplied by Manager for information on the methods that this interface exposes.) If the plug-in stores this interface pointer for later use, it should call AddRef on it. Each AddRef call must be balanced by a Release call when this interface is no longer needed. This pointer may be NULL to indicate that the plug-in should Release any stored IMoraeRecording pointer.

f. HRESULT SetProject (

IMoraeProject* pProject) [in]

This provides an interface to the project in which the recording is found. (See Information Supplied by Manager for information on the methods that can be called on this interface.) Note that the IMoraeProject interface corresponds to a study in the Morae Manager user interface.

Arguments:

i. pProject – Pointer to an IMoraeProject interface. If the plug-in stores this interface pointer for later use, it should call AddRef on it. Each AddRef call must be balanced by a Release call when this interface is no longer needed. This pointer may be NULL to indicate that the plug-in should Release any stored IMoraeProject pointer.

g. HRESULT GiveUpcomingFrame (

BSTRBLOB* pBitmapInfo,	**[in]**
BSTRBLOB* pBitmapBytes,	**[in,out]**
unsigned hyper rtStart,	**[in]**
unsigned hyper rtEnd,	**[in]**
int nOriginX,	**[in]**
int nOriginY,	**[in]**
double dblScaleX,	**[in]**
double dblScaleY,	**[in]**
ULONG uRenderingHints)	**[in]**

This call is made to provide a video frame (in the form of a bitmap) to the plug-in for drawing. pBitmapInfo provides information about the bitmap. The pData member of the *pBitmapBytes is a pointer to the actual bitmap data that the plug-in is allowed to modify. rtStart and rtEnd refer to the start and end time for the frame, respectively. The bitmap may be offset and scaled from the original recording depending on player options that the user has selected. The uRenderingHints will be a bitwise combination of RenderingHints enum values. Note that if the RenderingNotRealtime flag is not set, the plug-in should perform its operations quickly enough to not fall behind the video playback. If drawing lags, the user will be prompted to turn off one or more draw plug-ins.

Arguments:

i. pBitmapInfo – Pointer to a BSTRBLOB that contains information on the bitmap that makes up the video frame. Its pData member can be cast to a pointer to a windows BITMAPINFO structure.
ii. pBitmapBytes – Pointer to a BSTRBLOB structure that allows access to the actual video frame data.
iii. rtStart – 64-bit number that gives the start time for this video frame in 100 nanosecond units from the start of the recording.
iv. rtEnd – 64-bit number that gives the end time for this video frame in 100 nanosecond units from the start of the recording.
v. nOriginX – int that provides an offset in the X direction (in pixels) necessary to translate from the origin in the bitmap to the origin in the actual video frame.
vi. nOriginY – int that provides an offset in the Y direction (in pixels) necessary to translate from the origin in the bitmap to the origin in the actual video frame.
vii. dblScaleX – double that provides a scaling factor for the video in the horizontal direction. A value of 1.0 indicates no scaling. Smaller values indicate that the bitmap is scaled down.
viii. dblScaleY – double that provides a scaling factor for the video in the vertical direction. A value of 1.0 indicates no scaling. Smaller values indicate that the bitmap is scaled down.
ix. uRenderingHints – unsigned long made up of a bitwise combination of RenderingHints enum values.

h. HRESULT GetDrawSetupParameters (

SAFEARRAY ppParams) [out]**

Manager will make this call to ask for draw setup information that can be stored and applied to the plug-in in the SetDrawSetupParameters call. The plug-in will allocate resources for the array. The caller will be responsible for de-allocation of resources.

Arguments:

i. ppParams – Address of a pointer to a SAFEARRAY structure. The plug-in allocates memory for the SAFEARRAY and populates the data. The caller will de-allocate the array resources. All elements of the array must be of the same fixed size. The datatype cannot be VT_DISPATCH, VT_UNKNOWN, or VT_VARIANT. VT_BSTR is an acceptable datatype.

i. HRESULT SetDrawSetupParameters (

SAFEARRAY* pParams) [in]

This will set the criteria to be used to perform drawing. The criteria used in this call would have been acquired in a call to GetDrawSetupParameters. If this method is called before a call to ShowSetupDialog, the settings in the dialog should reflect these settings. Note that the caller is responsible for de-allocation of the SAFEARRAY.

Arguments:

i. pParams – Pointer to a Windows SAFEARRAY structure that contains the draw criteria.