Morae®
TechSmith®

Information Supplied by Manager

Introduction:

Morae Manager exposes several interfaces that can provide information needed by plug-ins to perform various tasks. Plug-ins involved in event searching or video drawing, for example, often require data from Morae's RRT data streams within a given recording. Plug-ins that provide additional functionality for Manager at the application level need access to information about the current project and its components. This document describes several interfaces that allow plug-ins to get at this sort of information.

Interfaces described in this document:

1. IMoraeRecording – This interface provides information about a particular recording. It also allows access to events streams that are associated with the recording.

2. IMoraeProject – This interface supplies information about a set of recordings that are grouped together to form a study in Manager. It provides access to the individual recordings in the study and to various aspects of the study definition.

3. IMoraeProject2 – This interface extends IMoraeProject.

4. IMoraeWorkspace – This interface provides information to the project that is open in Manager. It allows enumeration of the studies that are contained in the project.

Interfaces:

1. IMoraeRecording (derives from IUnknown)

This interface provides a plug-in with information about a specific recording. It is implemented by Morae Manager. Several plug-in interfaces (IMoraeEventSearch, IMoraeEventStream, and IMoraeVideoDraw) are given an IMoraeRecording interface to provide access to details about the recording and other event streams.

Interface ID:

IID_IMoraeRecording = 91779701-2890-47f3-9FEE-4FB079A8F718

Methods:

a. **HRESULT GetName (**

BSTR* pStr) **[out]**

This call gets the name of the recording. Note that this is a string that was entered by the user, so no LCID is specified. The caller will be responsible for freeing the system resources associated with this string.

Arguments:

i. pStr – Pointer to a BSTR variable that will be set to a system string. The caller must de-allocate the system resources for the string.

b. **HRESULT GetDescription (**

BSTR* pStr) **[out]**

This call gets the description of the recording. Note that this is a string that was entered by the user, so no LCID is specified. The caller will be responsible for freeing the system resources associated with this string.

Arguments:

i. pStr – Pointer to a BSTR variable that will be set to a system string. The caller must de-allocate the system resources for the string.

c. **HRESULT GetRecordingUUID (**

IID* puuid) **[out]**

This call retrieves a unique identifier for this recording.

Arguments:

i. puuid – Pointer to an IID variable that will be set to a GUID that uniquely identifies the recording.

d. **HRESULT GetDataStream (**

REFGUID iid, **[in]**

IMoraeEventStream ppStream)** **[out]**

This allows the caller to get access to any event streams that were captured as part of this recording. The iid corresponds to the value output by the IMoraeEventStream::GetEventStreamID call for a given event stream (described in Reading and Displaying Events). See the {Manager Native Events} document for GUIDs that identify the RRT data captured by Morae. Note that this method can also be used to access an event stream that was captured by a Recorder plug-in.

Arguments:

i. iid – GUID that uniquely identifies an event data stream.

ii. ppStream – Address of an IMoraeEventStream interface pointer. The caller must call Release on this pointer when it no longer needs it.

e. **HRESULT GetAppTags (**

BSTR strName,	**[in]**
VARIANT_BOOL bFriendlyName,	**[in]**
VARIANT* pVar)	**[out]**

This call asks for all of the application tag IDs that were used to identify instances of a given application that were running during a recording. These tag IDs correspond to the IDs provided in the call to IMoraeEventSearch::SetSearchApplications which restricts searching to particular applications for plug-ins that implement IMoraeEventSearch (described in Reading and Displaying Events). The strName input should be set to the name of the application. bFriendlyName indicates whether the strName is the filename of the executable or the "friendly" name that Manager displays to the user for that application.

Arguments:

i. strName –BSTR variable that contains the name of the appliction.
ii. bFriendlyName – VARIANT_BOOL that indicates whether this is the actual filename of the executable file for the application or the "friendly" name that Manager displays to the user.
iii. pVar – Pointer to a VARIANT that will output all tags associated with instances of the application. The vt will be set to VT_ARRAY | VT_I4. It will hold a SAFEARRAY of VT_I4 values. The caller is responsible for freeing the system resources allocated for this array.

f. **HRESULT GetAppName (**

int nTag,	**[in]**
BSTR* pstrExeName,	**[out]**
BSTR* pstrFriendlyName)	**[out]**

This looks up the executable file name and friendly name that correspond to a given application tag. The tag corresponds to the IDs provided in the call to IMoraeEventSearch::SetSearchApplications which is used to restrict event search to particular applications (see Reading and Displaying Events). The caller is responsible for de-allocating resources used for the BSTR outputs. If the nTag value is not a valid application tag for this recording, the return value will be E_INVALIDARG. If the nTag is valid, but no "friendly" name is available, the return value will be S_FALSE. S_OK indicates that both the exe name and the friendly name were found.

Arguments:

i. nTag – int variable that designates an instance of an application that was running during a recording.
ii. pstrExeName – Pointer to a BSTR variable that will be set to the filename of the executable that this tag refers to. The caller must de-allocate the system resources for the string.
iii. pstrFriendlyName – Pointer to a BSTR variable that will be set to the "friendly" name of the application. The caller must de-allocate the system resources for the string.

g. HRESULT GetFrameDuration (

VARIANT* pDuration) [out]

This method outputs the nominal duration of frames in the video. Note that the duration of individual video frames may vary throughout the recording. *pDuration will be set to a type of VT_I8, with the llVal set to the expected frame duration in 100 nanosecond units.

Arguments:

i. pDuration – Pointer to a VARIANT that will be set to the expected frame duration in 100 nanosecond units, with a vt of VT_I8.

h. HRESULT GetRemoteViewerName (

int nTag, [in]
BSTR* pStr) [out]

This translates a tag to the name of a remote viewer. It is primarily intended for use by Morae's internal RRT event streams.

Arguments:

i. nTag – int that identifies an Observer.
ii. pStr – Pointer to a BSTR variable that will be set to the name of the Observer. The caller must de-allocate resources for this string.

i. HRESULT GetChatUserName (

int nTag, [in]
BSTR* pStr) [out]

This translates a tag to the name of a UserVue chat user. It is primarily intended for use by Morae's internal RRT event streams. UserVue is no longer supported by TechSmith.

Arguments:

i. nTag – int that identifies a chat user.
ii. pStr – Pointer to a BSTR variable that will be set to the name of the chat user. The caller must de-allocate resources for this string.

j. HRESULT GetTreeviewStreamFolders (

REFIID uuidStream, [in]
SAFEARRAY ppBstrArray) [out]**

This is capable of outputting an array that gives the path of all folders that the given data type can be organized into in the tree view of Manager.

Arguments:

i. uuidStream – IID value that identifies the stream. Currently, only IID_VIDEOCLIP (f0f05e37-ee0e-4bb4-b2ea-b4235222f14e) is supported.
ii. ppBstrArray – Address of a SAFEARRAY pointer that will be set to a SAFEARRAY of BSTRs. The caller is responsible for de-allocating this array's resources.

k. HRESULT AddTreeviewStreamFolder (

REFIID uuidStream, [in]
BSTR strPath) [in]

This method adds a new folder path to the UI tree view for the given event stream (applies only to video clips).

Arguments:

i. uuidStream – IID value that identifies the stream. Currently, only IID_VIDEOCLIP (f0f05e37-ee0e-4bb4-b2ea-b4235222f14e) is supported.
ii. strPath – BSTR that gives the path.

l. HRESULT GetRecordingLength (

VARIANT* pVarRefTime) [out]

This method sets *pVarRefTime to a type of VT_I8. The llVal is set to the duration of the recording in 100 nanosecond units.

Arguments:

i. pVarRefTime – Pointer to a VARIANT that will be set to the recording duration.

m. HRESULT WndClassNameToTag (

BSTR strName, [in]
int* pTag) [out]

This call translates a window class name to a tag. It is primarily intended for use by Morae's internal RRT event streams.

Arguments:

i. strName – BSTR that gives the window class name.
ii. pTag – Pointer to an int value that will be set to the tag number that is associated with this window class in the recording.

n. **HRESULT WndClassTagToName (**

int nTag,	**[in]**
LCID localeID,	**[in]**
VARIANT_BOOL bFriendlyName	**[in]**
BSTR* pstrName)	**[out]**

This call translates a window tag to a class name. It is primarily intended for use by Morae's internal RRT event streams.

Arguments:

i. nTag – int that indicates tag value to look up.
ii. localeID – This value is currently ignored.
iii. bFriendlyName – This value is currently ignored.
iv. bstrName = Pointer to a BSTR variable that is set to the window class name. The caller must de-allocate this string.

2. IMoraeProject (derives from IUnknown)

This interface provides a plug-in with information about a group of Morae recordings with a common set of configuration data. In the Morae Manager UI, this grouping is referred to as a Study. It is implemented by Morae Manager. Several plug-in interfaces (IMoraeEventSearch, IMoraeEventStream, and IMoraeVideoDraw) are given an IMoraeProject interface to provide access to details about the study and all of its recordings.

Interface ID:

IID_IMoraeProject = 48000048-23DD-426c-843B-D6A888F521FD

Methods:

a. **HRESULT GetRecording (**

REFIID uuidRec,	**[in]**
IMoraeRecording ppIRecording)**	**[out]**

This call gets an interface to a specific recording.

Arguments:

i. uuidRec – IID that uniquely identifies the recording.
ii. ppIRecording – Address of an IMoraeRecording interface pointer that will be set in this call. The caller must call Release on this pointer when it no longer needs it.

b. **HRESULT GetAllAppNames (**

VARIANT* pVar)	**[out]**

This call outputs the "friendly" names of all applications that were running within all recordings in a study. The "friendly" names are the names displayed by Manager to the user rather than the actual executable file names. If successful, *pVar will be set to a SAFEARRAY of BSTRs (type = VT_ARRAY | VT_BSTR). The caller is responsible for freeing all system resources.

Arguments:

i. pVar – Pointer it a VARIANT that will be set to a SAFEARRAY of BSTRs. The caller must de-allocate resources for all strings.

c. HRESULT EnumRecordings (

IEnumMoraeRecording ppIEnum) [out]**

This call retrieves an enumerator that can be used to access all of the recordings in the study. See Morae Enumeration Interfaces for documentation on IEnumMoraeRecording.

Arguments:

i. ppIEnum – Address of an IEnumMoraeRecording interface pointer. The caller must call Release on this interface when finished with it.

d. HRESULT GetMarkerStrings (

LCID localeID, [in]
int nMarkerID, [in]
BSTR* pstrType, [out]
BSTR* pstrDefn) [out]

For a given marker ID, this outputs the type and definition strings.

Arguments:

i. localeID – LCID that indicates the language that should be used for the type.
ii. nMarkerID – int that defines the marker type.
iii. pstrType – Pointer to a BSTR that will be set to a string representation of the marker type. The caller is responsible for freeing this string.
iv. pstrDefn - Pointer to a BSTR that will be set to a string representation of the marker definition. The caller is responsible for freeing this string.

e. HRESULT GetMarkerScores (

int nMarkerID, [in]
SAFEARRAY ppArrayScores) [out]**

For a given marker ID, this call outputs the valid score IDs as a SAFEARRAY of VT_I4 values.

Arguments:

i. nMarkerID – int that defines the marker type.
ii. ppArrayScores – Address of a SAFEARRAY pointer that will be set to a SAFEARRAY of VT_I4 values giving the range of scores possible for this marker type. The caller must de-allocate this array.

f. HRESULT GetMarkerScoreString (

int nMarkerID,	**[in]**
int nScore,	**[in]**
BSTR* pstr)	**[out]**

This looks up string that the user has assigned to describe a marker score ID. The caller is responsible for freeing the string.

Arguments:

i. nMarkerID – int that defines the marker type.
ii. nScore – int that defines the score. This should be one of the values output by the GetMarkerScores method.
iii. Pstr – Pointer to a BSTR that will be set to a string representation of this marker score. The caller must de-allocate this string.

g. HRESULT GetTaskIDs (

SAFEARRAY ppTaskIDs)**	**[out]**

This method outputs an array of all ID values for the tasks that are configured for the study.

Arguments:

i. ppTaskIDs – Address of a SAFEARRAY pointer that will be set to a SAFEARRAY with a data type of VT_UI4. The caller is responsible for freeing the array.

h. HRESULT GetTaskDefinition (

UINT nTaskID,	**[in]**
BSTR* pstrName,	**[out]**
BSTR* pstrDefn)	**[out]**

This outputs the name and definition that the user has assigned to a particular task.

Arguments:

i. nTaskID – UINT that identifies the task. This should be one of the values in the array output by the GetTaskIDs method.
ii. pstrName – Pointer to a BSTR that will be set to the name of the task. The caller is responsible for freeing this string.
iii. pstrDefn – Pointer to a BSTR that will be set to the definition of the task. The caller is responsible for freeing this string.

i. HRESULT GetTaskScores (

UINT nTaskID,	**[in]**
SAFEARRAY ppArrayScores)**	**[out]**

For a given task ID, this call outputs the valid score IDs as a SAFEARRAY.

Arguments:

i. nTaskID – UINT that identifies the task. This should be one of the values in the array output by the GetTaskIDs method.

ii. ppArrayScores – Address of a SAFEARRAY pointer that will be set to a SAFEARRAY with a data type of VT_UI4. The caller is responsible for freeing the array.

j. HRESULT GetTaskScoreString (

UINT nTaskID,	**[in]**
int nScore,	**[in]**
BSTR* pStr)	**[out]**

This looks up string that the user has assigned to describe a task score ID. The caller is responsible for freeing the string.

Arguments:

i. nTaskID – UINT that identifies the task. This should be one of the values in the array output by the GetTaskIDs method.
ii. nScore – int value that specifies the score. This should be one of the values output by the GetTaskScores method.
iii. pStr – Pointer to a BSTR that will be set to the description of the task score. The caller is responsible for freeing this string

k. HRESULT GetSurveyQuestions (

REFGUID uuidSurvey,	**[in]**
SAFEARRAY ppQuestions)**	**[out]**

This outputs an array of questions for a particular survey.

Arguments:

i. uuidSurvey – IID that uniquely identifies the survey.
ii. ppQuestions – Address of a SAFEARRAY pointer that will be set to a SAFEARRAY of BSTRs. The caller is responsible for freeing the resources.

l. HRESULT GetSurveyQuestionStyles (

REFGUID uuidSurvey,	**[in]**
SAFEARRAY ppStyles)**	**[out]**

This outputs an array of styles for the questions in a particular survey. The output is a SAFEARRAY of VT_I4 values. The caller is responsible for freeing the resources.

Arguments:

i. uuidSurvey – IID that uniquely identifies the survey.
ii. ppStyles – Address of a SAFEARRAY pointer that will be set to a SAFEARRAY of VT_I4. Each element will be the style for a particular question. The elements of this array will correspond to the elements in the array output from the GetSurveyQuestions call. The caller is responsible for freeing the resources. Survey question styles are bitwise combinations of these values:
    a. 0x00000001 – radio buttons (single selection)
    b. 0x00000002 – checkboxes (multiple selection)
    c. 0x10000000 – edit control included
    d. 0x01000000 – horizontal layout (vertical is default)

    "a" and "b" are mutually exclusive.

m. HRESULT GetSurveyQuestionChoices (

REFGUID uuidSurvey, [in]
SAFEARRAY ppChoices) [out]**

This method gets the choices for all questions for a particular survey. It outputs an array of VARIANTS. Each of these VARIANTs corresponds to a question. Each VARIANT contains a SAFEARRAY of BSTRs. Each BSTR is a survey choice for the question.

Arguments:

i. uuidSurvey – IID that uniquely identifies the survey.
ii. ppChoices – Address of a SAFEARRAY pointer that will be set to a SAFEARRAY of VARIANTS in which each variant is of type VT_ARRAY | VT_BSTR. Each element of the main array corresponds to a survey question. Each sub-array gives all of the possible choices for that question. The caller must free all resources.

n. HRESULT GetSurveyQuestionCount (

REFGUID uuidSurvey, [in]
unsigned long* pnCount) [out]

This method outputs the number of questions found in a particular survey.

Arguments:

i. uuidSurvey – IID that uniquely identifies the survey.
ii. pnCount – Pointer to an unsigned long value that will be set to the number of questions in the survey.

o. HRESULT GetSurveyChoiceCounts (

REFGUID uuidSurvey, [in]
SAFEARRAY ppChoiceCounts) [out]**

This call outputs the number of choices for each question in a particular survey. *ppChoiceCounts will be a SAFEARRAY of VT_UI4 values.

Arguments:

i. uuidSurvey – IID that uniquely identifies the survey.
ii. ppChoiceCounts – Address of a SAFEARRAY pointer that will be set to a SAFEARRAY of VT_UI4 values. Each value gives the number of available choices for its corresponding question. The caller must free all resources.

p. HRESULT GetSurveyStyle (

REFGUID uuidSurvey, [in]
int* pnStyle) [out]

This call outputs the style of a particular survey.

Arguments:

i. uuidSurvey – IID that uniquely identifies the survey.

ii. pnStyle – Pointer to an int that will be set to 1 for a SUS survey or 2 for a custom survey.

q. HRESULT GetSurveyName (

REFGUID uuidSurvey, [in]
BSTR* pstrName) [out]

This call outputs the name of a particular survey.

Arguments:

i. uuidSurvey – IID that uniquely identifies the survey.
ii. pstrName – Pointer to a BSTR that will be set to the name of the survey. The caller is responsible for freeing the resources for the string.

r. HRESULT GetSurveyIDs (

SAFEARRAY ppSurveyIDs) [out]**

This call outputs the ID values of all surveys in the study.

Arguments:

i. ppSurveyIDs – Address of a SAFEARRAY pointer that will be set to an array of BSTR values. These strings will be the string form of a GUID. The caller is responsible for freeing all resources.

3. IMoraeProject2 (derives from IMoraeProject)

This interface extends the IMoraeProject interface, adding a couple of methods that provide additional information about the tasks and markers within a study.

Interface ID:

IID_IMoraeProject2 = C2A13A65-7D71-43E9-BE5D-B825DCCD7395

Enumerations:

a. MrkAttrType

This is used in the GetMarkerAttribute method to identify the attribute requested. Current valid values are:

i. MrkAttrColor – Requests the color that the user has designated to represent a marker type.

b. TaskAttrType

This is used in the GetTaskAttribute method to identify the attribute requested. Current valid values are:

i. TaskAttrinstruction – Requests the text of the instruction that the user has configured for a task type.

Methods:

a. HRESULT GetMarkerAttribute (

MrkAttrType ntype,	**[in]**
int nMarkerID,	**[in]**
VARIANT* pVarValue)	**[out]**

This call requests an attribute setting for a given marker type.

Arguments:

i. ntype – member of the MrkAttrType enumeration (described above).
ii. nMarkerID – int that identifies a marker type.
iii. pVarValue – Pointer to a VARIANT value that will be set to the attribute. Resources of this VARIANT must be properly freed. Currently, the only attribute that can be requested with this call is MrkAttrColor. pVarValue will be set to a VT_UI4 value that can be cast to a Windows COLORREF value.

b. HRESULT GetTaskAttribute (

TaskAttrType ntype,	**[in]**
UINT nTaskID,	**[in]**
VARIANT* pVarValue)	**[out]**

This call requests an attribute setting for a given task type.

Arguments:

i. ntype – member of the TaskAttrType enumeration that identifies an attribute.
ii. nTaskID – UINT that identifies a task type.
iii. pVarValue – Pointer to a VARIANT value that will be set to the attribute. Resources of this VARIANT must be properly freed. For example, a SAFEARRAY or BSTR must be freed with the appropriate system call.

4. IMoraeWorkspace (derives from IUnknown)

This interface allows a plug-in to enumerate through the studies that make up a project in the Manager UI. Note that a study exposes the IMoraeProject interface.

Interface ID:

IID_IMoraeWorkspace = 1FD7C206-4A2A-4D15-9877-846F0134C067

Methods:

a. HRESULT EnumProjects (

IEnumMoraeRecording ppIEnum)**	**[out]**

This call requests an enumerator for all of the studies in the project.

Arguments:

i. ppIEnum – Address of an IEnumMoraeRecording pointer that will be set in this call. The caller must call Release on this interface when finished with it. See {Morae Enumeration Interfaces} for details on this interface.