Morae®
TechSmith®

Using Recorder as a COM Server

Introduction:

Morae Recorder supports an out-of-process COM server, which gives a programmer access to many of its powerful recording features. This functionality can be accessed through any programming environment that supports COM. These include Visual Basic, Visual C/C++, Visual Studio.NET, Delphi, and C++ Builder. To facilitate the development of COM clients in languages, such as Visual Basic, that can use late binding to COM servers, the interfaces described here are derived from IDispatch.

To create an instance of Morae Recorder's COM server, clients should use this CLSID:

CLSID_MoraeRecordControl = A6CCC24C-894D-4E03-A193-8A8FC34AC076

When Recorder's main UI is running, no COM client can create an interface to it. Conversely, when a COM client is connected to Recorder, the main UI cannot be run. The COM client can choose to show a live preview window of what will be recorded. This is often useful when using a camera as a video source. The client can also ask Recorder to display a configuration dialog for each audio and video component to allow the user to set the details of how these devices will work. At this time, the COM client cannot individually set all of the study and recording parameters that govern the recording process. It can, however, command Recorder to load a configuration file which contains all of the study and recording details. The configuration file can be created either in Morae Manager or Morae Recorder via the normal user interface.

The typical steps in using Morae Recorder to perform a recording are:

1. Create an instance of the Recorder object.
2. Wait for the RecorderArmed event notification indicating that recorder is ready.
3. Optionally load a configuration file or set capture options.
4. Optionally select and preview video sources.
5. Optionally arm Recorder for more precise control of timing.
6. Optionally enable task logging via the COM interface if desired.
7. Begin recording.
8. Optionally log markers and tasks.
9. Stop recording.
10. Wait for notification that Recorder has finished writing the output and is ready to process commands.
11. Release the COM interface or start a new recording.

The COM interfaces that are implemented by Recorder are composed of methods and properties. Some development languages (such as Visual Basic) allow direct access to properties. Others (such as C++) require the use of accessor methods to retrieve or manipulate properties. In this case, properties can be retrieved by calling "get_<property name>" or, if not read-only, manipulated by calling "put_<property name>".

The Recorder COM server implements event notification by the standard COM connection point mechanism. The Recorder object implements IConnectionPointContainer. Clients can request callbacks to be made on either an IDispatch interface or the IMoraeRecordControlEvents2 interface.

Interfaces described in this document:

1. IMoraeRecordControl – This interface provides the basic capabilities needed to perform a recording. This includes starting and stopping the recording, setting markers, setting the output path, and turning options on or off.
2. IMoraeRecordControl2 – This interface allows more advanced control of Recorder, including selection of a configuration file, task logging, and audio or video device selection and preview.
3. IMoraeRecordControlEvents – This callback interface allows the caller to receive notification of basic recording events such as the beginning or end of the recording, and the writing of the output file.
4. IMoraeRecordControlEvents2 – This callback interface allows the caller to receive additional event notification from Recorder.
5. ICameraOptions – This interface allows turning camera capture on or off.
6. IAudioOptions – This interface allows turning audio recording on or off.
7. IKeystrokeOptions – This interface allows turning keystroke capture on or off.
8. ITextOptions – This interface allows turning text capture on or off.
9. IMouseOptions – This interface allows turning cursor capture on or off.
10. IStudyConfig – This interface allows read-only access to the study configuration information that is currently loaded.

Interfaces:

1. IMoraeRecordControl (derives from IDispatch)

The IMoraeRecordControl interface is used to control the basic aspects of a Morae Recording session. All properties must be set prior to running a recording. If an error is generated, a "_com_error" event will be fired. The IMoraeRecordControl interface contains several sub-interfaces that expose options for the various recording streams (camera, text, mouse, etc).

Interface ID:

IID_IMoraeRecordControl = 1F90953A-2243-4D5E-A28E-84ACF86872E2

Enumerations:

a. moraeError

This enumerated value is used both by this interface and the IMoraeRecordControlEvents interface (described below) to identify any error that occurs. Current valid values are listed in the Appendix.

Methods:

a. HRESULT StartRecording ()

This method commands Recorder to start the recording. Note that this operation can take a few seconds. The client can exert more precise control of the timing of the start if IMoraeRecordControl2::PrepareToRecord is called first. In that case, the StartRecording call takes much less time. When the recording has started, the IMoraeRecordControlEvents::StartedRecording event fires.

b. HRESULT StopRecording ()

This method commands Recorder to stop the recording. When the recording has stopped, the IMoraeRecordControlEvents::StoppedRecording event fires.

c. HRESULT InsertMarker (

BSTR bstrName,	**[in]**
long lType,	**[in]**
BSTR bstrAnnotation)	**[in]**

This method creates a marker at the current time in the recording.

Arguments:

i. bstrName – BSTR that supplies the name of the marker.
ii. lType – long value that identifies the marker type. Valid values range from 0 through 26.
iii. bstrAnnotation – BSTR that supplies the annotation note for the marker.

Properties:

a. RecordingFilePath

Data type = String. (BSTR in native languages, string in .NET languages)
This string is the full path to the desired final output file (extension = rdg) for the recording. It cannot be modified while a recording is in progress. Note that the actual output file name may differ from the one specified by this path to avoid overwriting an existing file. After a recording is completed, the client can call IMoraeRecordControl2::GetRecordingPath to determine the actual output path.

b. RemoteViewersAllowed

Data type = long.
Set this value to 1 to allow Observers to view the recording live. Set it to 0 to prevent Observers from viewing the recording. This cannot be modified while a recording is in progress.

c. CameraOptions

Data type = ICameraOptions interface pointer.
Read-only.
Clients can use this interface (described below) to enable or disable the camera capture option. This interface should not be used while a recording is in progress. Note that the camera option can also be manipulated through the IMoraeRecordControl2::SelectHardwareDevice method (described below).

d. AudioOptions

Data type = IAudioOptions interface pointer.
Read-only.
Clients can use this interface (described below) to enable or disable the audio capture option. This interface should not be used while a recording is in progress. Note that the audio option can also be manipulated through the IMoraeRecordControl2::SelectHardwareDevice method (described below).

e. KeystrokeOptions

Data type = IKeystrokeOptions interface pointer.
Read-only.
Clients can use this interface (described below) to enable or disable the capture of keystrokes during the recording. This interface should not be used while a recording is in progress.

f. TextOptions

Data type = ITextOptions interface pointer.
Read-only.
Clients can use this interface (described below) to enable or disable the capture of text during the recording. This interface should not be used while a recording is in progress.

g. MouseOptions

Data type = IMouseOptions interface pointer.
Read-only.
Clients can use this interface (described below) to enable or disable the capture of cursor activity during the recording. This interface should not be used while a recording is in progress.

h. IsRecording

Data type = VARIANT_BOOL.
Read-only.
Clients can check this property to determine whether a recording is in progress.

i. LastMoraeError

Data type = moraeError enumerated type.
Read-only.
Clients can check this property to determine the most recent error that has occurred.

2. IMoraeRecordControl2 (derives from IMoraeRecordControl)

The IMoraeRecordControl2 provides additional control over the recording process, adding support for hardware device manipulation and selection of a configuration file. All properties must be set prior to running a recording. The IMoraeRecordControl2 interface also provides access to the current IStudyConfig interface. This interface (described below) allows read-only access to the study configuration parameters.

Interface ID:

IID_IMoraeRecordControl2 = 34CBA1C4-A47B-45b1-A518-D1FB4FBC2DDB

Enumerations:

a. HardwareType

This enumerated value is used in this interface and in IMoraeRecordControlEvents2 to identify an audio or video source setting. Possible values are:

i. hwType_MainVid – Refers to the source for the primary video for the recording. The primary video source can either be the screen or a camera.
ii. hwType_Pip – Refers to the source for the secondary video for the recording. This can only be a camera.
iii. hwType_Audio – Refers to the source for the audio input stream.

b. TaskLogger

This enumerated value is used to specify whether the COM client will handle task logging. Values are:

i. taskLogger_COM – This value indicates that the COM client will be the task logger.
ii. taskLogger_Config – This value indicates that the task logging mechanism specified by the current recorder configuration will be in effect.

c. TaskAction

This enumerated value is used to specify the action to be taken in the LogTask method. Values are:

i. taskAction_Start – Starts the task.
ii. taskAction_Stop – Stops the task.
iii. taskAction_Pause – Pauses the current task.
iv. taskAction_Resume – Resumes a paused task.
v. taskAction_SetNote – Sets the value of the note for a task.
vi. taskAction_Cancel – Cancels the task.
vii. taskAction_SetType – Changes the task type.
viii. taskAction_SetScore – Sets the score for the task.

d. TaskType

This has a single enumerated value to specify an undefined task type. Its value is:

i. Task_Undefined – This is the only valid task type that is not defined in the current study configuration.

Methods:

a. HRESULT GetRecordingPath (

BSTR* pbstrPath) [out]

This method can be called after a recording has been completed to ask Recorder for the full path to the output file. This may differ from the path that was set in the IMoraeRecordControl::RecordingFilePath property to avoid overwriting an existing file.

Arguments:

i. pbstrPath – Address of a BSTR value that will be set to the full path of the output file. The caller is responsible for freeing this string.

b. HRESULT SetPreviewOwner (

VARIANT varHwnd) [in]

If the client wishes to display the video preview window, this method should be called before DisplayPreview is called to specify the owner of the modeless dialog window that will be displayed. It provides the HWND of the window owner.

Arguments:

i. varHwnd – VARIANT that contains the HWND of the owner window. The variant type can be VT_I4, VT_UI4, VT_I8, or VT_UI8.

c. HRESULT DisplayPreview (

VARIANT_BOOL bDisplay, [in]
int nLeft, [in]
int nTop) [in]

This method shows or hides the video preview window. The preview window can be useful for confirming that any cameras in use are configured and working properly. Before this call is made, the owner of the preview window should be set by calling SetPreviewOwner. After the owner is set, DisplayPreview can be called any number of times to show or hide the preview window.

Arguments:

i. bDisplay – Set to VARIANT_TRUE to show the window or VARIANT_FALSE to hide it.
ii. nLeft – int value that specifies the location of the left side of the window in screen coordinates (pixels).
iii. nTop – int value that specifies the location of the top of the window in screen coordinates (pixels).

d. HRESULT GetHardwareDeviceList (

HardwareType type,	**[in]**
SAFEARRAY parNames)**	**[out]**

This method requests the names of all devices that are available for a particular audio or video input.

Arguments:

i. type – A member of the HardwareType enumeration defined above to specify the type of input devices requested.
ii. parNames – Address of a SAFEARAY pointer that should be set to a SAFEARRAY of BSTR values which provides the name of each available device of the given type. Note that if the type is hwType_MainVid, the possible screen capture options will be included in this list. The client is responsible for freeing all resources.

e. HRESULT SelectHardwareDevice (

HardwareType type,	**[in]**
BSTR bstrName)	**[in]**

This method selects a device for a given input type.

Arguments:

i. type – A member of the HardwareType enumeration defined above to specify the type of input that is being set.
ii. bstrName –BSTR value which identifies the device. This name should be one of the names that was output in the GetHardwareDeviceList method call.

f. HRESULT GetSelectedDevice (

HardwareType type,	**[in]**
BSTR* bstrName)	**[out]**

This method asks which device is currently selected for a given input type.

Arguments:

i. type – A member of the HardwareType enumeration defined above to specify the type of input.
ii. bstrName – Address of a BSTR value which will be set to the device name. The caller must free this string.

g. HRESULT DoDeviceSetupDlg (

HardwareType type,	**[in]**
VARIANT varHwnd)	**[in]**

This method commands Recorder to display a dialog that allows the user to configure the settings for the current device of the given type. This will be a modal dialog, so this method will not return until the user has dismissed the dialog. Currently, no configuration dialog is available for the audio input device.

Arguments:

i. type – A member of the HardwareType enumeration defined above to specify the type of input.
ii. varHwnd – VARIANT that provides the HWND of the owner of the dialog window. The variant type can be VT_I4, VT_UI4, VT_I8, or VT_UI8.

h. HRESULT PrepareToRecord ()

This method commands Recorder to prepare to record. This method can be called prior to a call to IMoraeRecordControl::StartRecording to exert more precise control over the timing of the initiation of the recording.

i. HRESULT CreateTaskInstance (

UINT nTaskType,	**[in]**
VARIANT_BOOL bStartTask,	**[in]**
UINT* pTaskId)	**[out]**

This method commands Recorder to create an instance of a task and optionally logs the task as starting immediately. This can be called repeatedly for the same task type. In this case, each call will output the same TaskId value. This method will return an error if Recorder is not recording or if an attempt is made to start a new task when a different task is already in progress.

Arguments:

i. nTaskType – A UINT value that specifies the type of task to be created. This value can be either the Task_Undefined value of the TaskType enumeration or it one of the task ID values defined for the study configuration that is currently in use. These ID values are available as the StudyTaskIDs property found in the StudyConfiguration property.
ii. bStartTask – VARIANT_BOOL that can be set to VARIANT_TRUE to indicate that the task should be logged as starting immediately or VARIANT_FALSE to indicate that the task will be started later.
iii. pTaskId – Pointer to a UINT value that will receive an identifier that can be used in subsequent calls to LogTask.

j. HRESULT LogTask (

TaskAction action,	**[in]**
UINT nTaskID,	**[in]**
VARIANT varData)	**[in]**

This method commands Recorder to take some action on a task that was created by calling CreateTaskInstance. The possible actions are listed in the TaskAction enumeration. After a task has been started, it must be stopped before another task can be started. A task can be started and stopped multiple times. In that case, the task will consist of multiple time segments. (Morae does not support multiple separate instances of a given task within a recording.) A task must be started before its score or note can be set. (The task does not need to be currently in progress to allow its score or note to be set.)

Arguments:

i. action – A value from the TaskAction enumeration that indicates what action to take for this task. Description of each action:
    a. taskAction_Start – Starts logging of a segment of a task. A task can consist of multiple segments, but only one instance of a given task can exist within a recording. Note that if CreateTaskInstance was called with bStartTask set to VARIANT_TRUE, this command is not needed.
    b. taskAction_Stop – Stops logging of a segment of a task. After this is called, a different task (or another segment of the same task) can be started. After a segment has been stopped, it cannot be canceled or have its type changed. This effectively commits a segment of a task.
    c. taskAction_Pause – Pauses the logging of a task segment. A sequence of taskAction_Pause and taskAction_Resume calls can be used to create multiple segments of a task.
    d. taskAction_Resume – Resumes a task that had been paused. This will create a new segment of the task.
    e. taskAction_SetNote – Sets the note for a task. The task may be in progress or a segment of the task must have been previously committed with a taskAction_Stop command for this call to be successful.
    f. taskAction_Cancel – A cancel operation can be performed any time after a taskAction_Start call, but before the task segment had been committed with a taskAction_Stop command.
    g. taskAction_SetType – This command can be used to change the type of a task that is in progress. It cannot be used on a segment that has been committed with a taskAction_Stop command. It will not affect any task segments that have been previously committed.
    h. taskAction_SetScore – This command sets the score attribute of the task. The task may be in progress or a segment of the task must have been previously committed with a taskAction_Stop command for this call to be successful.
ii. nTaskID – UINT that was output from the CreateTaskInstance method call which identifies which task is to be logged.
iii. varData – VARIANT data type. Its use depends on the value of the “action” argument. For taskAction_SetNote, this VARIANT should be a VT_BSTR type. For taskAction_SetType, this VARIANT should be a VT_UI4 type. For taskAction_SetScore, this VARIANT should be a VT_I4 type. Allowable values for type and score can be found from the StudyConfiguration property described below.

Properties:

a. **ConfigFile**

Data type = String. (BSTR in native languages, string in .NET languages)
Write-only.
This string is the full path to the configuration file that contains the study and recording configuration settings that Recorder should use. The configuration file can be created in either Morae Manager or Morae Recorder.

b. **TaskLoggerSetting**

Data type = TaskLogger enumerated type.
This setting is not yet implemented.

c. **StudyConfiguration**

Data type = IStudyConfig interface pointer.
This interface (described below) can be used to request information about the current study configuration.

3. IMoraeRecordControlEvents (derives from IUnknown)

The COM client can implement this interface to receive notification of events that occur in Recorder. Alternatively, the client can receive event notification by implementing the IDispatch interface. With either interface, the client uses the COM connection point model to request event notification.

Interface ID:

IID_IMoraeRecordControlEvents = EBBF49CE-3430-42da-A82F-6E4AE48AF041

Enumerations:

a. **moraeProgressOutputState**

This enumerated value is used in the OutputProgress method to identify the current state of Recorder.

i. mopsWrite – Indicates that Recorder is writing the output file.
ii. mopsVerify – Indicates that Recorder is verifying the output file.

Methods:

a. **HRESULT RecorderArmed ()**

This method is called when Recorder reaches an idle state and is ready to receive commands.

b. **HRESULT StartedRecording ()**

This method is called when Recorder has begun recording.

c. **HRESULT StoppedRecording ()**

This method is called when a recording session has ended. It does indicate that the session has been saved.

d. **HRESULT OutputProgress (**

moraeOutputProgressState state,	**[in]**
long nPercent)	**[in]**

This method is called to indicate progress in saving or verifying the final output file.

Arguments:

i. state – A member of the moraeOutputProgressState enumeration defined above to specify the operation that is in progress.
ii. nPercent – long value that ranges from 0 – 100 to indicate the percentage of the operation that has been completed.

e. **HRESULT OutputWritten ()**

This method is called when Recorder has completed writing the output file from a recording session.

f. **HRESULT RecordingError (**

moraeError error)	**[in]**

This method is called when Recorder has encountered an error.

4. IMoraeRecordControlEvents2 (derives from IMoraeRecordControlEvents)

The COM client can implement this interface to receive additional notification of events that are not covered by IMoraeRecordControlEvents. Alternatively, the client can receive event notification by implementing the IDispatch interface. With either interface, the client uses the COM connection point model to request event notification.

Interface ID:

IID_IMoraeRecordControlEvents2 = 23E30B16-030C-4e53-8C93-D525CA8DE1BE

Enumerations:

a. **TriggerType**

This enumerated value is used in the WaitingForTrigger method to identify the type of trigger that will cause Recorder to start the session.

i. trigType_COM – Indicates that Recorder is waiting for a StartRecording command from the COM client. Recorder enters this state after a call to IMoraeRecordControl2::PrepareToRecord.

ii. trigType_Timed – Indicates that Recorder is will start at a particular time as defined by its redording configuration.
iii. trigType_App – Indicates that Recorder is waiting for an application to be launched or closed.
iv. trigType_Click – Indicates that Recorder is waiting for a mouse click.
v. trigType_Observer – Indicates that Recorder is waiting for a signal from an Observer.
vi. trigType_AutoPilot – Indicates that the session will start when the participant triggers it through the AutoPilot interface.
vii. trigType_Keystroke – Indicates that a keystroke will start the session.

Methods:

a. HRESULT PreviewWindowMoved (

VARIANT_BOOL bVisible,	**[in]**
int nLeft,	**[in]**
int nTop)	**[in]**

This method is called when the video preview window is showing, and the user has either moved or closed it. This allows the COM client to store the location of the preview window if desired.

Arguments:

i. bVisible – VARIANT_BOOL that indicates whether the preview window is currently visible.
ii. nLeft – int that gives the location of the left side of the preview window in screen coordinates (pixels).
iii. nTop – int that gives the location of the top of the preview window in screen coordinates (pixels).

b. HRESULT WaitingForTrigger (

TriggerType type)	**[in]**

This method is called when the Recorder is waiting for an external trigger to begin a session. Recorder can enter this state in response to a call to IMoraeRecordControl2::PrepareToRecord. It can also enter a waiting state in response to a call to IMoraeRecordControl::StartRecording if the current recording configuration calls for it to begin the session in response to a trigger.

Argument:

i. type –Member of the TriggerType enumeration described above.

c. HRESULT SelectedDeviceChanged (

HardwareType type) [in]

This method is called the device selected for an audio or video input has changed. This can be in response to removal of a device from the system or selection of a camera for one video input that had been previously selected for the other video input.

Argument:

i. type –Member of the HardwareType enumeration described above.

d. HRESULT DeviceListChanged (

HardwareType type) [in]

This method is called when the list of available hardware devices for a given audio or video source has changed. This indicates that a device has been added to or removed from the system. The client should call IMoraeRecordControl2::GetHardwareDeviceList to determine which devices are available for selection.

Argument:

i. type –Member of the HardwareType enumeration described above.

5. ICameraOptions (derives from IDispatch)

This interface allows turning the option of recording a camera as a secondary video source on or off. It is a property of the IMoraeRecordControl interface.

Interface ID:

IID_CameraOptions = 45015727-32AE-4C91-A86E-D03014506A6F

Properties:

a. Enable

Data type = VARIANT_BOOL

A COM client can set this property to enable or disable use of a camera as a secondary video source. Note that this property may be modified by setting the IMoraeRecordControl2::ConfigFile property.

6. IAudioOptions (derives from IDispatch)

This interface allows turning the option of recording an audio source on or off. It is a property of the IMoraeRecordControl interface.

Interface ID:

IID_AudioOptions = 23AAE18E-8E25-4721-9594-AEA75693F342

Properties:

a. Enable

Data type = VARIANT_BOOL

A COM client can set this property to enable or disable use of an audio source. Note that this property may be modified by setting the IMoraeRecordControl2::ConfigFile property.

7. IKeystrokeOptions (derives from IDispatch)

This interface allows turning the option of keystroke capture on or off. It is a property of the IMoraeRecordControl interface.

Interface ID:

IID_KeystrokeOptions = 12DDF399-0290-49C1-B9B2-A7F1C29605A2

Properties:

a. Enable

Data type = VARIANT_BOOL

A COM client can set this property to enable or disable keystroke capture. Note that this property may be modified by setting the IMoraeRecordControl2::ConfigFile property.

8. ITextOptions (derives from IDispatch)

This interface allows turning the option of recording screen text on or off. It is a property of the IMoraeRecordControl interface.

Interface ID:

IID_TextOptions = 192EC51B-743D-45ED-993E-1448C16A3045

Properties:

a. Enable

Data type = VARIANT_BOOL

A COM client can set this property to enable or disable screen text capture. Note that this property may be modified by setting the IMoraeRecordControl2::ConfigFile property.

9. IMouseOptions (derives from IDispatch)

This interface allows turning the option of recording cursor activity on or off. It is a property of the IMoraeRecordControl interface.

Interface ID:

IID_MouseOptions = 1D2C82CB-FC6F-4505-A4BF-A6C04B341B70

Properties:

b. Enable

Data type = VARIANT_BOOL

A COM client can set this property to enable or disable recording cursor activity. Note that this property may be modified by setting the IMoraeRecordControl2::ConfigFile property.

10. IStudyConfig (derives from IDispatch)

The COM client can use this interface to request information about the current study configuration that will be used to record a session. It is a property of the IMoraeRecordControl2 interface.

Interface ID:

IID_IStudyConfig = 0C0A5144-0695-4C77-8FE2-BD46369C9B63

Methods:

a. HRESULT GetStudyMarkerStrings (

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

This method requests the type and definition strings for a particular marker.

Arguments:

i. localeID – LCID value that indicates the language that the pstrType string will be in.
ii. nMarkerID – int value that identifies the marker type. This value should be in the range 0 – 26.
iii. pstrType – Address of a BSTR variable that will be set to the marker type string. The caller must free this string.
iv. pstrDefn – Address of a BSTR variable that will be set to the marker definition. Note that this is a user-defined string, so the LCID parameter does not apply to it. The caller must free this string.

b. HRESULT GetStudyMarkerScores (

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

This method requests the valid score values that can be set for a given marker.

Arguments:

i. nMarkerID – int value that identifies the marker type. This value should be in the range 0 – 26.
ii. ppArrayScores – Address of a SAFEARRAY pointer. This will be set to contain an array of VT_I4 values. The caller must free this array.

c. HRESULT GetStudyMarkerScoreString (

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

This method requests the string that the user has defined for a particular marker score value.

Arguments:

i. nMarkerID – int value that identifies the marker type. This value should be in the range 0 – 26.
ii. nScore – int value that identifies the score.
iii. pstrDefn – Address of a BSTR variable that will be set to the string that describes the marker score. Note that this is a user-defined string, so the LCID parameter does not apply to it. The caller must free this string

d. HRESULT GetStudyTaskDefinition (

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

This method requests the name and definition strings for a particular task. Note that these are user-defined strings, so no language identifier is necessary.

Arguments:

i. nTaskID – UINT value that identifies the task type. This value should be one of the values in the StudyTaskIDs property.
ii. pstrName – Address of a BSTR variable that will be set to the task name. The caller must free this string.
iii. pstrDefn – Address of a BSTR variable that will be set to the task definition. The caller must free this string.

e. HRESULT GetStudyTaskScores (

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

This method requests the valid score values that can be set for a given task.

Arguments:

i. nTaskID – UINT value that identifies the task type. This value should be one of the values in the StudyTaskIDs property.
ii. ppArrayScores – Address of a SAFEARRAY pointer. This will be set to contain an array of VT_I4 values. The caller must free this array.

f. HRESULT GetStudyTaskScoreString (

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

This method requests the string that the user has defined for a particular marker score value.

Arguments:

i. nTaskID – UINT value that identifies the task type. This value should be one of the values in the StudyTaskIDs property.
ii. nScore – int value that identifies the score.
iii. pstrDefn – Address of a BSTR variable that will be set to the string that describes the marker score. Note that this is a user-defined string, so the LCID parameter does not apply to it. The caller must free this string

g. HRESULT GetStudySurveyName (

BSTR struuidSurvey,	**[in]**
BSTR* pstrName)	**[out]**

This method requests the name of a survey.

Arguments:

i. struuidSurvey – BSTR that is the string representation of the GUID value that uniquely identifies a survey that has been configured for the study. This string should be one of the strings found in the StudySurveyIDs property.
ii. pstrName – Address of a BSTR variable that will be set to the survey name. The caller must free this string.

Properties:

a. **StudyTaskIDs**

   Data type = SAFEARRAY of UINT (VT_UI4) values.

   Read-only.

   This property contains an array of the ID values of all of the tasks that have been defined for the current study.

b. **StudySurveyIDs**

   Data type = SAFEARRAY of BSTR (VT_BSTR) values.

   Read-only.

   This property contains an array of the ID values of all of the surveys that have been defined for the current study. These ID values are GUIDs in string form.

Appendix: moraeError Enumerations

The following lists all the possible error codes that LastMoraeError could be set to by a COM method.

merrNone	No error has occurred
merrInvalidTempFolder	The temporary recoding storage folder is invalid
merrOutputFileExists	The output RDG file already exists
merrTempAccessDenied	Recorder cannot write to the temporary storage folder
merrMemError	No more memory could be allocated
merrEmptyTempFolder	No temporary storage folder has been specified
merrEmptyOutputFilename	No output RDG file name has been specified
merrUnkownError	An unknown error has occurred
merrInvalidFilename	The file name given was invalid
merrInvalidStartTime	The automatic start time is invalid
merrCreateScreenCapture	Screen recorder could not be initialized
merrSetScreenInput	Failed to set the screen recorder input
merrSetScreenOutput	Failed to set the screen recorder output
merrSetScreenOutFile	Failed to set the screen recorder output file
merrSetScreenSaveDlg	Failed to set the screen recorder save dialog
merrSetScreenTempDir	Failed to set the screen recorder temporary storage directory
merrSetAudioDevice	Failed to initialize the audio input device
merrSetAudioFormat	Failed to set the audio stream format
merrSetAviInterleave	Failed to set the AVI interleave property
merrSetFrameCallback	Failed to set the frame capture callback
merrSetSdkStateCallback	Failed to set the SDK state callback
merrSetScreenFrameRate	Failed to set the screen AVI frame rate
merrSetScreenKeyframe	Failed to set the screen AVI keyframe
merrSetScreenDataRate	Failed to set the screen AVI data rate
merrSetAviQuality	Failed to set the AVI quality property
merrSetScreenVideoCodec	Failed to set the screen video codec
merrSetScreenVideoCodecState	Failed to set the screen video codec state
merrSetScreenHideCapRect	Failed to hide the screen video capture rectangle
merrNoScreenConfigInfo	There was no screen configuration information

merrSetAudioFileOption	Failed to set audio file options
merrStartScreenCapture	The screen capture failed to start
merrStopTimerSetup	An error occurred while setting up the stop timer
merrCameraStart	Failed to start the camera capture
merrStopTimeInvalid	The automatic stop time is invalid
merrSetScreenStartPaused	Failed to set the screen recorder to pause when started
merrScreenResume	Failed to resume the screen recorder
merrEmptyOutputFolder	No output folder for the RDG has been specified
merrOutputAccessDenied	Recorder does not have permission to write to the output folder
merrInvalidOutputFolder	The folder to put the output RDG file in is invalid
merrInvalidOutputFileName	The name of the output RDG file is invalid
merrCameraInitialization	The camera could not be initialized
merrEmptyTempFileName	The temporary file name was empty
merrTempFileExists	The temporary file already exists
merrStartTimerSetup	An error occurred while setting up the stop timer
merrSdkAllocation	Memory allocation in the SDK failed
merrSdkRendering	An error occurred while the SDK was trying to render something
merrSdkFile	A file error occurred while manipulating a file in the SDK
merrSdkAudioDevice	The SDK could not initialize the audio device
merrSdkAudioCodec	The SDK could not set the audio codec
merrSdkVideoCodec	The SDK could not set the video codec
merrSetSdkErrorCallback	Failed to set the error callback for the SDK
merrPrestartTimerSetup	Failed to set the prestart timer
merrReleaseCaptureHooks	Failed to release capture hooks
merrSetCaptureHooks	Failed to set capture hooks
merrCreateFrameThread	Failed to create the frame thread
merrCantFindScreenVideoFile	The screen recorder failed to create the screen video file
merrSetCursor	Failed to set the cursor
merrSetLayeredWndCapture	Failed to initialize layered window capture
merrMainWndDoesNotExist	Fatal Error: Recorder's main window does not exist
merrRecordingInProgress	The operation could not be completed because a recording is in progress

merrNoRecordingInProgress	The operation could not be completed because no recording is in progress
merrRecordingStartFatalFailure	Recorder encountered a fatal error while attempting to start the recording
merrRecordingStopFatalFailure	Recorder encountered a fatal error while attempting to stop the recording
merrInsertMarkerFatalFailure	Recorder encountered a fatal error while attempting to insert a marker
merrInterfacesNotInitialized	Internal interfaces that the COM object uses were not initialized correctly