Morae®
TechSmith®

Communication Between Recorder Plug-ins and Observer Plug-ins

Introduction:

This document provides an overview of the interfaces required to transfer data between a Morae Recorder event source plug-in and a Morae Observer draw plug-in. This data transfer mechanism allows real-time display in Morae Observer of data captured by an event source plug-in in Recorder. Note that two-way communication is possible. An Observer plug-in can also send messages to a Recorder plug-in. This facilitates the ability of an Observer plug-in to request set-up or calibration information that may be necessary for proper display of data in Observer.

The four COM interfaces described in this document are involved in communication between Recorder plug-ins and Observer plug-ins:

1. IMoraeRecorderEventSink – This interface is implemented by Recorder. Plug-ins can use this interface to send data to all Observers or to a particular Observer. This interface can also be used to request information from and issue commands to Recorder.

2. IObserverDraw – This interface must be implemented by Observer plug-ins that wish to display event data. The plug-in will be able to receive data from a corresponding Recorder plug-in. The data can be used to alter the main video as it is displayed to the user.

3. IMoraeObserverEventSink – Observer implements this interface to allow its plug-ins to send data to a corresponding Recorder plug-in, to request information, and to issue commands.

4. IMoraeReceiveData – This interface can be implemented by Recorder plug-ins to enable receipt of data from a corresponding Observer plug-in.

Morae uses the COM connection point model to allow event source plug-ins in Recorder to transfer data to Observers. Recorder acts as an event sink, receiving calls on the IMoraeRecorderEventSink interface (defined below). This requires event sources that want to provide data to Observers to implement the standard IConnectionPointContainer and IConnectionPoint interfaces. When Morae Recorder loads a plug-in which implements the IEventCapture interface, it will perform the following steps:

1. Call QueryInterface on the interface to get the IConnectionPointContainer interface.
2. If the QueryInterface call is successful, it will call IConnectionPointContainer::FindConnectionPoint to connect with the IID_IMoraeRecorderEventSink id to get an IConnectionPoint interface. (The IMoraeRecorderEventSink interface is defined below.)
3. If this is successful, Recorder will call IConnectionPoint::Advise to form the connection.

4. Before releasing the IEventSource interface, Recorder will call IConnectionPoint::Unadvise to disconnect.

Likewise, an Observer plug-in can send data to a Recorder plug-in via a callback on Observer's IMoraeObserverEventSink interface. The same connection point model is used on the Observer side to provide this interface to plug-ins that implement the IObserverDraw interface.

Observer plug-ins can receive data from Recorder plug-ins by implementing IObserverDraw. Recorder plug-ins can receive data from Observer plug-ins by implementing IMoraeReceiveData.

Interfaces:

1. IMoraeRecorderEventSink (derives from IUnknown)

Morae Recorder implements this interface to allow event capture plug-ins to transmit data to corresponding Observer plug-ins. Observer plug-ins will receive the data via a call to IObserverDraw::ReceiveDataFromRecorder. This interface also supports methods to request information from Recorder and to issue commands to Recorder.

Interface ID:

IID_IMoraeRecorderEventSink = A9BF754E-5205-461B-BCE6-72833B468086

Enumerations:

a. RecParamType

This is used in the GetRecordingParameter method to specify what information is being requested. Current valid values are:

i. RecParamMainVidFrameRate – Requests the frame rate for the main video source.
ii. RecParamSecVidFrameRate – Requests the frame rate for the secondary video source.
iii. RecParamTaskLoggerID – Requests the ID of the Observer that is serving as the task logger.
iv. RecParamMainWndHandle – Requests the HWND of Recorder's main window.

b. RecCmndType

This is used in the RunRecorderCommand method to identify the command.

i. RecCmndAddViewableWindow – Adds a window to the View menu of Recorder. This allows the user to show or hide the window.

Methods:

a. HRESULT SendBroadcastData (

REFCLSID clsidRecipient,	**[in]**
unsigned long nByteCount,	**[in]**
const byte* pBuffer)	**[in]**

This method broadcasts data to all Observers that are currently connected.

Arguments:

i. clsidRecipient – Ref to a CLSID that uniquely identifies the Observer plug-in that is the intended recipient of the data.
ii. nByteCount – Number of valid bytes pointed to by pBuffer
iii. pBuffer = Pointer to a buffer of data. Morae will make its own copy of the data from this buffer before this method returns. The caller is responsible for managing the memory that pBuffer points to.

b. HRESULT SendTargetedData(

REFCLSID clsidRecipient,	**[in]**
unsigned long nRecipientID,	**[in]**
unsigned long nByteCount,	**[in]**
const byte* pBuffer)	**[in]**

This method sends data to a particular Observer that is currently connected. It allows the Recorder plug-in to respond to a query from a plug-in originating from a particular Observer.

Arguments:

i. clsidRecipient – Ref to a CLSID that uniquely identifies the Observer plug-in that is the intended recipient of the data.
ii. nRecipientID – Identifier of a particular Observer. This ID can come from a call from Recorder to IMoraeReceiveData::ReceiveDataFromObserver (described below).
iii. nByteCount – Number of valid bytes pointed to by pBuffer
iv. pBuffer = Pointer to a buffer of data. Morae will make its own copy of the data from this buffer before this method returns. The caller is responsible for managing the memory that pBuffer points to.

c. HRESULT GetRecordingParameter(

RecParamType nParamType,	**[in]**
VARIANT varSpecifier,	**[in]**
VARIANT* pVarParam)	**[out]**

This method requests information from the Recorder.

Arguments:

i. nParamType – Specifies the type of information requested.
ii. varSpecifier – Provides more detail on the information request. The type of VARIANT expected depends on the value of nParamType.
iii. pVarParam – Pointer to a VARIANT that receives the information requested. The caller is responsible for freeing any memory that was allocated for this variable. For example, if the pVarParam type is set to VT_BSTR, the caller is must call SysFreeString to free the BSTR value.

Supported values for nParamType and the VARIANT types:

i. RecParamMainVidFrameRate
    a) varSpecifier – Type = VT_BOOL: Set the boolVal to VARIANT_TRUE for target frame rate, VARIANT_FALSE for actual frame rate. (Actual frame rate is not supported at this time.)
    b) pVarParam – Will be set to: type = VT_R8, dblVal = frames per second.
ii. RecParamSecVidFrameRate
    a) varSpecifier – Type = VT_BOOL: Set the boolVal to VARIANT_TRUE for target frame rate, VARIANT_FALSE for actual frame rate. (Actual frame rate is not supported at this time.)
    b) pVarParam – Will be set to: type = VT_R8, dblVal = frames per second.
iii. RecParamTaskLoggerID
    a) varSpecifier – Not used
    b) pVarParam – Will be set to: type = VT_UI4, ulVal = ID. This ID corresponds to the nRecipientID used in the IMoraeRecorderEventSink::SendTargetedData method call or the nSenderID used in the IMoraeReceiveData::ReceiveDataFromObserver method.
iv. RecParamMainWndHandle
    a) varSpecifier – Not used
    b) pVarParam – Will be set to: type = VT_UI4, ulVal = HWND of the Recorder main window.

d. HRESULT RunRecorderCommand(

RecCmndType nCmndType,	**[in]**
VARIANT varParam1,	**[in]**
VARIANT varParam2)	**[in]**

This method sends a command to the Recorder.

Arguments:

i. nCmndType – Specifies the type of command.
ii. varParam1 - Provides more detail on the command. The type of VARIANT expected depends on the value of nCmndType.
iii. varParam2 - Provides more detail on the command. The type of VARIANT expected depends on the value of nCmndType.

Supported values for nParamType and the VARIANT types:

i. RecCmndAddViewableWindow
    a) varParam1 – Type = VT_BSTR: Set to the window name that should be displayed in the View menu to allow the user to show or hide the window
    b) varParam2 – Type = VT_UI4, Set ulVal to the HWND of the window that the user will be able to show or hide.

2. IObserverDraw (derives from IUnknown)

This interface allows a plug-in to draw on the main video as it is displayed in Observer. Data can be sent from a Recorder event source plug-in to this Observer plug-in, allowing the real-time display of captured data.

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

A5225649-15EF-45EE-998D-53E123f9A48F

Interface ID:

IID_ IObserverDraw = AFEF491C-8632-4FF2-A168-5ECE36E394A6

Enumerations:

a. CommunicationStatus

This is used in the SetCommunicationStatus method to identify the current status.

i. CommunicationConnected
   This value indicates that the Observer is connected to Morae Recorder.
ii. CommunicationNotConnected
   This value indicates that the Observer is not connected to Morae Recorder.
iii. CommunicationDisconnecting
   This value indicates that the Observer is preparing to disconnect from Morae Recorder.

b. RecordingStatus

This is used in the SetRecordingStatus method to identify the current status.

i. RecordingIdle
Recorder is in an idle state. It has not performed any recording operations since it was launched.

ii. RecordingArmed
Recorder has been armed. It is waiting for an event to trigger the start of the recording process.

iii. RecordingRunning
Recorder is actively recording a session.

iv. RecordingCompleted
A Recording session has been completed.

v. RecordingCanceled
An armed recording was canceled before the start trigger occurred.

vi. RecordingPaused
A recording in progress has entered the paused state. (This is not yet implemented as of version 3.3.)

Methods:

a. HRESULT GetDrawPluginName(

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

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

Arguments:

i. pStr – Pointer to a BSTR. The plug-in must allocate the BSTR and set this pointer. The caller will be responsible for de-allocation of resources.

ii. localeID – Should be used to determine the language.

b. HRESULT GetDrawPluginDescription (

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.

Arguments:

i. pStr – Pointer to a BSTR. The plug-in must allocate the BSTR and set this pointer. The caller will be responsible for de-allocation of resources.

ii. localeID – Should be used to determine the language.

c. HRESULT HasDrawSetupDialog(

VARIANT_BOOL* pBool) [out]

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

Arguments:

i. pBool – Pointer to a VARIANT_BOOL. If the plug-in sets pBool to VARIANT_TRUE, it must be able to show a dialog in the ShowDrawSetupDialog method call. Otherwise, ShowSetupDialog will not be called.

d. HRESULT ShowDrawSetupDialog (

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

This requests that the plug-in display a setup dialog to allow the user to configure draw 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 Observer will assume that the user has modified the draw criteria.

Arguments:

i. varHwndParent – Gives the HWND that the dialog should use as a parent window. The plug-in should be able to accept either a 32-bit or 64-bit number, either signed or unsigned.
ii. localeID – Should be used to determine the language.

e. HRESULT ReceiveDataFromRecorder(

unsigned long nDataBytes, [in]
const byte* pData) [in]

This provides data that has been sent from an event capture plug-in in Recorder.

Arguments:

i. nDataBytes – Size of the pData buffer in bytes.
ii. pData – Pointer to a buffer that contains the data sent from Recorder. This pointer should not be stored because it will not remain valid after this call returns. If the plug-in needs to access this data later, it must allocate and store its own copy of the data.

f. HRESULT ReceiveVideoFrame(

BSTRBLOB* pBitmapInfo,	**[in]**
BSTRBLOB* pBitmapBytes,	**[in,out]**
unsigned hyper rtStart,	**[in]**
unsigned hyper rtEnd)	**[in]**

This call is made to provide a video frame (in the form of a bitmap) to the plug-in for drawing. The pData member of the *pBitmapInfo can be cast to a pointer to a windows BITMAPINFO structure to get 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. These times are given in 100 nanosecond units from the start of the recording.

Arguments:

i. pBitmapInfo – Pointer to a BSTRBLOB structure that contains the information about the bitmap frame. The pData member of the *pBitmapInfo can be cast to a pointer to a windows BITMAPINFO structure to get information about the bitmap
ii. pBitmapBytes – Pointer to a BSTRBLOB structure that contains the bitmap data for the frame. The pData member of the *pBitmapBytes is a pointer to the actual bitmap data that the plug-in is allowed to modify.
iii. rtStart – Start time (from the start of the recording) for the frame given in 100 nanosecond units.
iv. rtEnd – End time (from the start of the recording) for the frame given in 100 nanosecond units.

g. HRESULT GetDrawParameters (

LPSAFEARRAY* ppParams)	**[out]**

Observer will make this call to ask for draw setup information that can be stored and applied to the plug-in in the SetDrawParameters call. The plug-in will allocate resources for the array. The caller will be responsible for de-allocation of resources. The datatype of this array cannot be VT_DISPATCH, VT_UNKNOWN, or VT_VARIANT. All elements of the array must be of the same fixed size. VT_BSTR is an acceptable datatype.

Arguments:

i. ppParams – Pointer to a SAFEARRAY pointer.

h. **HRESULT SetDrawParameters (**
LPSAFEARRAY 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 GetDrawParameters. 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 SAFEARRAY that contains setup information for the plug-in.

i. **HRESULT SetCommunicationStatus(**
CommunicationStatus status) [in]

Called when the communication status of Observer changes.

Arguments:

i. status – One of the values described for the CommunicationStatus enumeration above.

j. **HRESULT SetRecordingStatus(**
RecordingStatus status) [in]

Called when the status of the Recorder changes.

Arguments:

i. status – One of the values described for the RecordingStatus enumeration above.

3. IMoraeObserverEventSink (derives from IUnknown)

Morae Observer exposes this interface to allow an Observer plug-in to send data to a corresponding Recorder plug-in. This can be used, for example, to allow an Observer plug-in to request setup or calibration information when it connects to Recorder with a session already in progress.

Interface ID:

IID_ IMoraeObserverEventSink = EDAEEFD5-593A-4ED4-A080-5E467ACF2CDF

Enumerations:

a. **ObsParamType**

This is used in the GetRecordingParameter method to specify what information is being requested. Current valid values are:

i. ObsParamMainWndHandle – Requests the HWND of Observer's main window.

ii. ObsParamObserverName – Requests the name of the current user in Observer.

b. ObsCmndType

i. ObsCmndAddViewableWindow – Adds a window to the View menu of Observer. This allows the user to show or hide the window.

Methods:

a. HRESULT SendDataToRecorder(

REFCLSID clsidRecipient,	**[in]**
unsigned long nByteCount,	**[in]**
const byte* pBuffer)	**[in]**

Arguments

i. clsidRecipient – CLSID that is used to identify the Recorder plug-in that will receive this data.
ii. nByteCount – Size in bytes of the data in the pBuffer buffer.
iii. pBuffer – Pointer to a buffer of data. Morae will make its own copy of the data from this buffer before this method returns.

b. HRESULT GeObserverParameter(

ObsParamType nParamType,	**[in]**
VARIANT varSpecifier,	**[in]**
VARIANT* pVarParam)	**[out]**

This method requests information from the Observer.

Arguments:

i. nParamType – Specifies the type of information requested.
ii. varSpecifier – Provides more detail on the information request. The type of VARIANT expected depends on the value of nParamType.
iii. pVarParam – Pointer to a VARIANT that receives the information requested. The caller is responsible for freeing any memory that was allocated for this variable. For example, if the pVarParam type is set to VT_BSTR, the caller is must call SysFreeString to free the BSTR value.

Supported values for nParamType and the VARIANT types:

i. ObsParamMainWndHandle
    a) varSpecifier – Not used.
    b) pVarParam – Will be set to: type = VT_UI4, ulVal = HWND of Observer's main window.
ii. ObsParamObserverName
    a) varSpecifier – Not used.
    b) pVarParam – Will be set to: type = VT_BSTR, bstrVal = Name of user logged in to Observer. (This is the name that the Observer used when connecting to the Recorder.

c. HRESULT RunObserverCommand(

ObsCmndType nCmndType,	**[in]**
VARIANT varParam1,	**[in]**
VARIANT varParam2)	**[in]**

This method sends a command to the Recorder.

Arguments:

i. nCmndType – Specifies the type of command.
ii. varParam1 - Provides more detail on the command. The type of VARIANT expected depends on the value of nCmndType.
iii. varParam2 - Provides more detail on the command. The type of VARIANT expected depends on the value of nCmndType.

Supported values for nParamType and the VARIANT types:

i. ObsCmndAddViewableWindow
    a) varParam1 – Type = VT_BSTR: Set to the window name that should be displayed in the View menu to allow the user to show or hide the window.
    b) varParam2 – Type = VT_UI4, Set ulVal to the HWND of the window that the user can control.

4. IMoraeReceiveData (derives from IUnknown)

Morae Recorder plug-ins can implement this optional interface to receive data from corresponding Observer plug-ins. This can be used, for example, to allow an Observer plug-in to request setup or calibration information when it connects to Recorder with a session already in progress.

Interface ID:

IID_ IMoraeReceiveData = 626B4947-543B-49DC-BDEC-96434DBA084E

Methods:

a. HRESULT ReceiveDataFromObserver(

unsigned long nSenderID,	**[in]**
unsigned long nByteCount,	**[in]**
const byte* pBuffer)	**[in]**

Arguments

i. nSenderID – An ID number that can be used to identify an Observer in calls to IMoraeRecorderEventSink::SendTargetedData.
ii. nByteCount – Size in bytes of the data in the pBuffer buffer.
iii. pBuffer – Pointer to a data buffer that contains information sent from an Observer plug-in. This pointer should not be stored because it will not remain valid after this call returns. If the plug-in needs to access this data later, it must allocate and store its own copy of the data.