Morae®
TechSmith®

Capturing Event Data

Introduction:

Morae Recorder supports the use of plug-ins to capture additional event data beyond Morae's standard RRT information during a recording session. This mechanism can be used to capture a wide variety of information, including physiological measurements of the user's responses, more precise information about the operation of a specific application, or more general data about the operation of the system. A plug-in can be either an in-proc or out-of-proc COM server. Most details of the information capture, including number of files generated and file format, are left to the plug-in. Recorder will basically tell the plug-in when to start and stop recording and where to write its output. The plug-in will be able to provide configuration and calibration dialogs to the user. Recorded data will be loaded into Manager for examination and analysis by a corresponding plug-in as described in {Reading and Displaying Events in Manager}. In addition to recording data for later analysis in Manager, the Recorder plug-in can optionally send captured event data Observers for live display. This mechanism is described in {Supplying Data from a Recorder Plug-in to Observers}.

Morae provides two mechanisms for enabling a plug-in to be recognized and loaded by Recorder.

1. When registering the plug-in's CLSID with Microsoft Windows, it also registers itself as implementing the CATID_EventCaptureSource category (A5DC9083-35B9-411F-8E5D-54F731296D2C). This is the more common mechanism. It works for either in-proc or out-of-proc plug-ins. This allows Recorder to directly load and communicate with the plug-in.

2. For a plug-in that is implemented as a DLL and is loaded into one or more other application on the system, we provide an alternative mechanism to facilitate inter-process communication. Rather than registering itself as implementing a specific category, when the plug-in registers its CLSID, it also registers itself with the IPluginManager interface of our RecorderCaptureHelper application. This can simplify communication if a plug-in is loaded into multiple instances of other applications. The RecorderCaptureHelper can relay important commands from Recorder to all instances of the plug-in that are running and handle additional instances of the plug-in that start up while a recording is in progress. This is described in more detail below.

Interfaces described in this document:

1. IEventCapture – This interface must be implemented by all Recorder plug-ins. Recorder will call this interface to perform all operations necessary for capturing events and saving the data as part of the recording session.

2. IPluginManager – This interface is implemented by Morae's RecorderCaptureHelper. It can be used by plug-ins that are implemented as DLLs and mapped into other applications running on the system to simplify inter-process communication.

Related documents:

1. {Reading and Displaying Events in Manager} explains how data that is captured by a Recorder plug-in can be utilized in Manager.
2. {Supplying Data from a Recorder Plug-in to Observers} explains how captured data can be transferred between Recorder and Observer for live display.

Interfaces:

1. IEventCapture (derives from IUnknown)

This COM interface will be called by Morae Recorder to enable the capture of additional event data during a Morae recording session. An object that implements this interface will be expected to create event file(s) in a given directory. It will be allowed to display optional setup and calibration dialogs to allow the user to configure options and perform calibration as necessary.

This interface can be implemented by either an in-proc or out-of-proc COM server.

Data captured by an event source plug-in can optionally be sent to a corresponding plug-in in Observer for live display of captured data. For more information on this data transfer mechanism, see {Supplying Data from a Recorder Plug-in to Observers}.

Interface ID:

IID_IEventCapture = 78F72AF2-41A0-47F3-B753-6139A9849708

Enumerations:

a. SetupStatus

This is used in the GetSetupStatus method to indicate to Recorder whether or not the plug-in is ready to record. Current valid values are:

i. Setup_Ready – The plug-in is ready to record.
ii. Setup_Incomplete – The plug-in's setup dialog should be called to complete the configuration before the recording can begin.

b. CalibStatus

GetCalibrationStatus should output one of these values:

i. Calib_Ready – The plug-in is ready to record.
ii. Calib_Needed – The plug-in requires calibration before the recording can begin.

c. CalibReq

GetCalibrationRequirements should output one of these values:

i. Calib_Never – The plug-in never requires calibration.
ii. Calib_OneTime – The plug-in requires calibration at least once after it has been installed.
iii. Calib_EachRecording – The plug-in requires calibration prior to the start of each recording.
iv. Calib_EachStartup – The plug-in requires calibration each time Recorder is started.

d. CaptureControl

This is one of the inputs in the ControlCapture method call:

i. Capture_Idle – Recorder is in an idle state.
ii. Capture_Start – The recording session is starting.
iii. Capture_Stop – The recording session is complete.
iv. Capture_Pause – An active recording session has been paused.
v. Capture_Resume – A paused recording has been resumed.

Methods:

a. HRESULT GetName (

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

This gets the name that will be used to identify the plug-in to the user. The nLocaleID will indicate which language should be used.

Arguments:

i. pStr – 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 – LCID that should be used to determine language.

b. HRESULT GetDescription (

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

This gets a description of what the plug-in will capture. If the plug-in has setup options, the description should reflect the options that were set by the user.

Arguments:

i. pStr – 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 – LCID that should be used to determine language.

c. HRESULT HasSetupDialog (

VARIANT_BOOL* pBool) [out]

This is called to determine whether the plug-in is able to display a setup dialog.

Arguments:

i. pBool – Pointer to a VARIANT_BOOL that should be set to VARIANT_TRUE if a setup dialog is available, VARIANT_FALSE if not.

d. HRESULT HasCalibrationDialog (

VARIANT_BOOL* pBool) [out]

This is called to determine whether the plug-in is able to display a calibration dialog.

Arguments:

i. pBool – Pointer to a VARIANT_BOOL that should be set to VARIANT_TRUE if a calibration dialog is available, VARIANT_FALSE if not.

e. HRESULT GetSetupStatus (

long* pStatus) [out]

This method is called before each recording session begins to determine whether the plug-in requires configuration before the recording can start.

Arguments:

i. pStatus – Pointer to a long variable that should be set to one of the values listed for the SetupStatus enumeration above.

f. HRESULT GetCalibrationStatus (

long* pStatus) [out]

This method is called before each recording session begins to determine whether the plug-in requires calibration before the recording can start.

Arguments:

i. pStatus – Pointer to a long variable that should be set to one of the values listed for the CalibStatus enumeration above.

g. HRESULT GetCalibrationRequirements (

unsigned long* pStatus) [out]

This method can be called to determine the frequency with which calibration is required by the plug-in.

Arguments:

i. pStatus – Pointer to an unsigned long variable that should be set to one of the values listed for the CalibReq enumeration above.

h. HRESULT GetCaptureParameters (

SAFEARRAY ppArParam) [out]**

This method allows Recorder to store setup parameters that control how the plug-in is configured to operate. Typically, this will only be important for plug-ins that expose a setup dialog to allow the user to select capture options. If the plug-in does not require storage of setup parameters, it should return S_FALSE and not modify the *ppArParam argument. The datatype of the SAFEARRAY created by the plug-in should be "flat." In other words, it should not consist of pointers. VT_BSTR is an exception to this. A SAFEARRAY of BSTRs is valid. If VT_VARIANT is the datatype, none of the variants in the array can hold pointers. Datatypes that are not supported include VT_UNKNOWN and VT_DISPATCH. (Note that SAFEARRAYs of VT_RECORD datatype must be used with care. In this case, when SetCaptureParameters is called to restore parameters that have been previously saved, no IRecordinfo interface will be available. Some of the Windows SAFEARRAY manipulation functions will not work properly with an array of this type that does not have an IRecordinfo implementation available.)

Plug-ins that operate via the RecorderCaptureHelper and wish to use default COM marshalling must use a SAFEARRAY with VT_UI1 datatype (unsigned char) for capture parameters. In this case COM can use a type library that is installed with Morae to marshal the method calls.

Arguments:

i. ppArParam – Address of a SAFEARRAY pointer. The plug-in must use system calls to allocate resources for this array. Limitations on the data type of this array are described above. Manager will be responsible for de-allocation of this array.

i. HRESULT SetCaptureParameters (

SAFEARRAY* pArParam) [in]

Recorder will use this method to apply setup parameters previously stored from the GetCaptureParameters method call. If this is not applicable, the plug-in should return S_OK and ignore the pArParam argument. The caller will de-allocate this array after this method returns.

Arguments:

i. pArParam – Pointer to a SAFEARRAY. The information in this SAFEARRAY came from a previous call to GetCaptureParameters.

j. HRESULT SetOutputDirectory (

BSTR strDir) [in]

This call will be made before ControlCapture is called to tell the plug-in to start capturing events. The plug-in should create any files that it needs to write in this directory, making sure that it is not overwriting existing files.

Arguments:

i. strDir – BSTR that provides the full path to the directory that the plug-in should place data files into.

k. HRESULT GetOutputFilepaths (

SAFEARRAY parNames) [out]**

This call will be made after ControlCapture is called with an argument of "Capture_Stop" to tell the plug-in to stop capturing events. The plug-in should create a safearray of BSTR values, each of which supplies the full path for a file that the plug-in has created and filled with captured event data.

Arguments:

i. pStrName – Address of a SAFEARRAY pointer. The plug-in will allocate resources for this SAFEARRAY and the BSTR values that it contains. The caller will de-allocate all resources.

l. HRESULT ShowSetupDialog (

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

This call asks the plug-in to display a dialog to allow the user to configure capture options. The varHwndParent argument provides the handle for the parent window for the dialog. If the user cancels the dialog, this method should return S_FALSE. A return value of S_OK indicates that the user wishes to save his new settings.

Arguments:

i. varHwndParent – VARIANT that contains the HWND of Recorder's main window. This should be used as the owner of the dialog that is shown. The plug-in should be able to handle a value that is either signed or unsigned, 32-bit or 64-bit.

ii. localeID – LCID that should be used to determine language.

m. HRESULT ShowCalibrationDialog (

VARIANT varHwndParent,	**[in]**
long localeID)	**[in]**

This call asks the plug-in to display a dialog to allow the user to perform a calibration. The varHwndParent argument provides the handle for the parent window for the dialog. The plug-in is responsible for saving and applying all calibration settings.

Arguments:

i. varHwndParent – VARIANT that contains the HWND of Recorder's main window. This should be used as the owner of the dialog that is shown. The plug-in should be able to handle a value that is either signed or unsigned, 32-bit or 64-bit.
ii. localeID – LCID that should be used to determine language.

n. HRESULT SetCaptureRect (

int nTop,	**[in]**
int nBottom,	**[in]**
int nLeft,	**[in]**
int nRight)	**[in]**

When the computer screen is serving as the primary video source for a recording, this provides the screen coordinates of the screen region that is being recorded. If the screen is not the primary video source, this method will be called with all of the parameters set to 0. The plug-in can (optionally) use this information to restrict the events that it captures and records.

Arguments:

i. nTop – int value that provides the top of the capture rectangle in screen coordinates (pixels).
ii. nBottom – int value that provides the bottom of the capture rectangle in screen coordinates (pixels).
iii. nLeft – int value that provides the left side of the capture rectangle in screen coordinates (pixels).
iv. nRight – int value that provides the right side of the capture rectangle in screen coordinates (pixels).

o. HRESULT ControlCapture (

long nCmd,	**[in]**
unsigned long nTickCount)	**[in]**

This method is called to indicate when a recording starts and stops. The nCmd input will be set to one of the CaptureControl enumeration values. The nTickCount parameter will be set to the tick count (acquired via the Win32 API call GetTickCount()) of the computer when the recording was started, stopped, paused, or resumed. If a series of plug-ins are called, all will get the same nTickCount value for that call. When a recording is started, SetOutputDirectory and SetCaptureRect will be called before ControlCapture is called with nCmd set to Capture_Start. When a recording is stopped, ControlCapture will be called with nCmd set to

Capture_Stop. Then, GetOutputFilepaths will be called. Then, ControlCapture will be called with nCmd set to Capture_Idle. Note that the Capture_Idle command will be sent to all plug-ins when a recording is stopped, whether or not the user chose to include the plug-in in the data to be captured in the recording.

Arguments:

i. nCmd – long value that will be set to one of the CaptureControl enumeration values.
ii. nTickCount – unsigned long that provides the tick count of the system when this operation occurred.

p. HRESULT GetEventReaderInfo (

GUID* puuID,	**[out]**
unsigned long* pnMajorVersion,	**[out]**
unsigned long* pnMinorVersion,	**[out]**
BSTR* pstrHandlerName,	**[out]**
BSTR* pstrURL,	**[out]**
long localeID)	**[in]**

This method is called to get information on the event handler plug-in object that will be used by Manager to read the event file(s) created by the IEventCapture plug-in. *puuID should be set to the CLSID of the component that will read the file(s). The other information will be displayed to the user in an error message if an error occurs when Manager attempts to read the data file. *pstrURL should be set to the URL of a website that the user could go to for more information or to check for an upgrade.

Arguments:

i. puuID – Pointer to a GUID variable that the plug-in should set to the CLSID of the plug-in that Manager will call to read the files created during the recording. This plug-in must implement the IMoraeEventStream interface.
ii. pnMajorVersion – Pointer to an unsigned long variable that the plug-in should set to the major version of the plug-in that will read the data file(s).
iii. pnMinorVersion – Pointer to an unsigned long variable that the plug-in should set to the minor version of the plug-in that will read the data file(s).
iv. pstrHandlerName – Pointer to a BSTR variable that the plug-in should be set to the name of the plug-in that Manager will use to read the data file(s).
v. pstrURL – Pointer to a BSTR variable that should be set to the URL of a website that the user can go to for more information on the plug-in that will read the data file(s).
vi. nLocaleID – LCID that should be used to determine the language that for the pstrHandlerName.

2. IPluginManager (derives from IUnknown)

This interface is designed to simplify development of certain types of plug-ins for Morae Recorder. The use of this interface is optional. Its purpose is to aid in inter-process communication between Morae Recorder and plug-ins that are mapped into other processes on the computer. IPluginManager is implemented by the RecorderCaptureHelper COM server that is installed along with Morae Recorder.

To use this interface, a plug-in should still register its CLSID with the operating system, but it does not register as implementing a category. Instead, when it registers or unregisters, it also instantiates the RecorderCaptureHelper object (by calling CoCreateInstance) and calls RegisterComponent or UnregisterComponent in the IPluginManager interface. Then, each time that the plug-in is loaded into a target process, it again instantiates the RecorderCaptureHelper and calls AddEventSource to notify the RecorderCaptureHelper of its presence. Before unloading, the plug-in should call RemoveEventSource to notify RecorderCaptureHelper that it is leaving. The RecorderCaptureHelper will handle communication and synching issues among all instances of the plug-in and Morae Recorder.

Note: At this time (Morae version 3.3.0), event capture plug-ins that use this mechanism are not able to send live data to Observers.

CLSID:

CLSID_PluginManager = 3F877D30-6802-4377-9863-6E13F3F84D3C

Interface ID:

IID_IPluginManager = 3F4E295C-0935-4ACC-B00A-B37F13DDE85F

Instantiating RecorderCaptureHelper:

To create an instance of RecorderCaptureHelper, call CoCreateInstance with the above values for the CLSID and IID. The context should be CLSCTX_LOCAL_SERVER.

Methods:

a. HRESULT RegisterComponent (

BSTR strName,	**[in]**
REFGUID clsidHelper,	**[in]**
REFGUID clsidRecorder)	**[in]**

This method will register the plug-in as operating via the RecorderCaptureHelper. It provides two unique identifiers – one that matches its CLSID as registered with Windows, the other that will be used in later calls to AddEventSource and RemoveEventSource. Note that this method should be called only when the plug-in is registered. On Vista or later, it must be called with Admin privileges.

Arguments:

i. strName – BSTR that provides a name for the plug-in.

ii. clsidHelper – GUID that is set to the CLSID that the plug-in uses in registering itself with Windows.
iii. clsidRecorder – GUID that the plug-in will use in subsequent calls to AddEventSource and RemoveEventSource.

b. HRESULT UnRegisterComponent (

BSTR strName,	**[in]**
REFGUID clsidHelper,	**[in]**
REFGUID clsidRecorder)	**[in]**

This method will unregister the plug-in as operating via the RecorderCaptureHelper. Arguments are the same as described above for RegisterComponent. Note that this method should be called only when the plug-in is unregistered. On Vista or later, it must be called with Admin privileges.

Arguments:

i. strName – BSTR that provides a name for the plug-in.
ii. clsidHelper – GUID that is set to the CLSID that the plug-in uses in registering itself with Windows.
iii. clsidRecorder – GUID that the plug-in will use in subsequent calls to AddEventSource and RemoveEventSource.

c. HRESULT AddEventSource (

IUnknown* pSource,	**[in]**
unsigned long nProcID,	**[in]**
REFGUID clsidRecorder)	**[in]**

This method should be called by each instance of the plug-in that is mapped into a target process. Note that, depending on the state of Recorder, several methods may be called on the plug-in before this method returns. These methods could include IEventCapture::SetCaptureParameters, IEventCapture::SetOutputDirectory, IEventCapture::SetCaptureRect, and IEventCapture::ControlCapture.

Arguments:

i. pSource – Pointer to the IUnknown interface of the plug-in object that implements IEventCapture.
ii. nProcID – Identifier of the process which the plug-in is loaded into. This can be obtained from the operating system by calling GetCurrentProcessID.
iii. clsidRecorder – GUID that the plug-in supplied when it called RegisterComponent.

d. HRESULT RemoveEventSource (

IUnknown* pSource,	**[in]**
unsigned long nProcID,	**[in]**
REFGUID clsidRecorder)	**[in]**

This method should be called by each instance of the plug-in before it is unmapped from a target process. All arguments are the same as for AddEventSource. Note that if a recording is in progress, IEventSource::GetOutputFilepaths may be called on the plug-in before this method returns to get a list of files this plug-in may have created to save event data.

Arguments:

i. pSource – Pointer to the IUnknown interface of the plug-in object that implements IEventCapture.
ii. nProcID – Identifier of the process which the plug-in is loaded into. This can be obtained from the operating system by calling GetCurrentProcessID.
iii. clsidRecorder – GUID that the plug-in supplied when it called RegisterComponent.