Data Analysis for Graphing in Manager

Introduction:

One way of visualizing the event data collected during a recording session is to create graphs based on the data. On its Graph tab, Manager allows the user to configure a few different types of graphs. This really means that the user is able to select and configure an analysis tool that prepares a data set for Manager's graphing engine.

Manager's graphical analysis is task-based. In other words, data points are expected to be calculated for each selected task within each selected recording. The user can typically elect to display the graphical results on a per-task or per-recording basis and choose to see summary data or individual values for each task. The analysis tool is able to access all event data that was captured in the selected recordings. It will be given the necessary information on which recordings and which tasks the user has selected for inclusion in the calculations.

When an analysis tool performs an analysis, it provides Manager with an interface to a result set that Manager can use to request the data that will be presented in graphical form. Manager currently presents data in the form of histograms. Individual, mean or distributional data can be displayed.

To provide an analysis tool that Manger can use to prepare graphical displays of the data, a plug-in must be registered as implementing the CATID_ManagerAnalysisTool category (FD7A8CAD-AB03-498D-85FF-CC9CE197FCD1).

The following interfaces are involved in data analysis:

1. IMoraeAnalysisTool – This is the primary interface that must be implemented by the plug-in. Manager uses this interface to get information about the plug-in and to create an analysis object.

2. IMoraeAnalysis – This interface must be implemented by the plug-in object that actually performs an analysis.

3. IMoraeAnalysisResult – This interface must be implemented by the plug-in to provide the results of the analysis to Manager.

4. IMoraeTaskInfo – Manager supplies this interface to the plug-in. The plug-in can use this interface to get information about an individual task that is included in the analysis.

5. IMoraeTaskGroupInfo – Manager supplies this interface to the plug-in. It provides information about a set of tasks that are group together for the analysis.

6. IMoraeRecordingInfo – Manager supplies this interface to the plug-in. It provides basic information about a recording that is included in the analysis.

Related documents:

1. Morae Enumeration Interfaces describes several enumerators that are involved in data analysis.
2. Reading and Displaying Events describes interfaces that are exposed by event stream plug-ins that can be used to access data from individual event streams.
3. Information Supplied by Manager describes the IMoraeProject and IMoraeRecording interfaces which can be used to access the event streams within individual recordings.

Interfaces:

1. IMoraeAnalysisTool (derives from IUnknown)

This interface must be implemented by the plug-in to expose basic information about the analysis tool capabilities and display properties. Manager will also use it to create an analysis object that will be used to actually generate analysis data.

Interface ID:

IID_IMoraeAnalysisTool = 27025EFB-E98C-43f8-9B78-1BC2AFA9829B

Methods:

a. HRESULT GetAnalysisToolName (

```
LCID localeID,          [in]
BSTR* pstr )            [out]
```

This call requests the name of the analysis tool. This will be displayed to the user.

Arguments:

i. localeID – LCID that should be used to determine language.
ii. pstr – Pointer to a BSTR variable that the plug-in should set to the name. The caller will free the string.

b. HRESULT GetAnalysisToolDescription (

```
LCID localeID,          [in]
BSTR* pstr )            [out]
```

This call requests a description of the analysis tool. It is not called at this time (version 3.3).

Arguments:

i. localeID – LCID that should be used to determine language.
ii. pstr – Pointer to a BSTR variable that the plug-in should set to the description. The caller will free the string.

c. **HRESULT GetAnalysisCategory (**
LCID localeID, **[in]**
BSTR* pstr) **[out]**

This call requests the name of the category that the analysis tool fits into. This can be one of Manager's existing analysis tool categories (Application Analysis, Input Device Analysis, Marker Analysis, Survey Analysis, Task Analysis, or Web Page Analysis) or the plug-in can define its own category. The tools are grouped into these categories to organize the choices for the user.

Arguments:

i. localeID – LCID that should be used to determine language.
ii. pstr – Pointer to a BSTR variable that the plug-in should set to the name. The caller will free the string.

d. **HRESULT GetCategoryImage (**
BSTR* pstr) **[out]**

This call requests the path to a file that contains an image that can be used to represent the category that this analysis tool belongs to. This can be a BMP, GIF, JPEG, PNG, TIFF, or EMF file.

Arguments:

i. pstr – Pointer to a BSTR variable that the plug-in should set to the path. The caller will free the string.

e. **HRESULT GetAnalysisImage (**
BSTR* pstr) **[out]**

This call requests the path to a file that contains an image that can be used to represent the analysis that this tool performs. This can be a BMP, GIF, JPEG, PNG, TIFF, or EMF file.

Arguments:

i. pstr – Pointer to a BSTR variable that the plug-in should set to the path. The caller will free the string.

f. **HRESULT SetProject (**
IMoraeProject* pIProject) **[in]**

This call provides the plug-in with an interface to the study that contains the recordings used in the analysis. This interface can be used to access the individual data streams that were captured during the recording. See Information Supplied by Manager for more information in this interface.

Arguments:

i. pIProject – Pointer to an IMoraeProject interface. If the plug-in stores this interface for later use, it should call AddRef on it. This plug-in must call Release on the interface when finished. This may be NULL. In this case, the plug-in should call Release on a stored interface.

g. HRESULT CreateAnalysis (

IMoraeAnalysis ppIAnalysis) [out]**

This call instructs the plug-in to create an analysis object which implements IMoraeAnalysis (described below). The IMoraeAnalysis interface will then be used to actually perform an analysis.

Arguments:

i. ppIAnalysis – Address of an IMoraeAnalysis interface pointer. Manager will call Release on this interface when it is finished with it.

h. HRESULT GetTaskTreatmentParams (

VARIANT_BOOL* pAllowUserTaskSelection, [out]

GUID* puuidGroupingType) [out]

This call determines whether Manager will display the task selection window in the UI to allow the user to select which tasks will be included in the analysis. This will typically be true. Specialized plug-ins may choose to not allow task selection if the analysis is not based on data contained within tasks.

Arguments:

i. pAllowUserTaskSelection – Pointer to a VARIANT_BOOL variable that the plug-in should set to VARIANT_TRUE to allow task selection or VARIANT_FALSE to not allow task selection.

ii. puuidGroupingType – This is not used at this time (version 3.3). The plug-in can set this to a NULL GUID.

2. IMoraeAnalysis (derives from IUnknown)

This interface must be implemented by the plug-in to actually perform the analysis. It includes methods to allow the user to configure an analysis, store settings, specify target recordings and tasks, and run the analysis.

Interface ID:

IID_IMoraeAnalysis = DAE97344-B9D9-4463-85BF-ED19286021BA

Methods:

a. HRESULT GetAnalysisName (

LCID localeID, [in]

BSTR* pstr) [out]

This call requests the name of the analysis that this tool will perform. This will be displayed to the user.

Arguments:

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

ii. pstr – Pointer to a BSTR variable that the plug-in should set to the name. The caller will free the string.

b. **HRESULT GetAnalysisDescription (**

LCID localeID, **[in]**

BSTR* pstr) **[out]**

This call requests the description of the analysis that this tool will perform. If the user is able to configure the analysis via a setup dialog, this string should briefly describe the settings he has selected.

Arguments:

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

ii. pstr – Pointer to a BSTR variable that the plug-in should set to the description. The caller will free the string.

c. **HRESULT HasConfigurationDialog (**

VARIANT_BOOL* pBool) **[out]**

This call asks whether the object has a configuration dialog that allows the user to set parameters.

Arguments:

i. pBool – Pointer to a VARIANT_BOOL variable. The plug-in should set to this to VARIANT_TRUE if it has a dialog, VARIANT_FALSE otherwise.

d. **HRESULT ShowAnalysisDialog (**

LCID localeID, **[in]**

VARIANT varWndParent) **[in]**

This call asks the plug-in to display a configuration dialog. If the user makes any changes, it should return S_OK. If the return value is S_FALSE, Manager will assume that no changes were made.

Arguments:

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

ii. varWndParent – VARIANT that contains the HWND of Manager's main window. This should be the owner of the dialog that is displayed. The plug-in should be able to handle either a VT_I4 or VT_I8 data type.

e. **HRESULT GetAnalysisCriteria (**

SAFEARRAY ppAr)** **[out]**

This call requests the configuration parameters that the user has configured. Typically, this will only be important for plug-ins that expose a setup dialog to allow the user to select analysis 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 "flat". In other words, it should not consist of pointers. VT_BSTR is an exception to this. A SAFEARRAY of BSTRs is valid. Datatypes that are not supported include VT_VARIANT, 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.)

Arguments:

i. ppAr – Address of a SAFEARRAY pointer. The plug-in must allocate and populate this array. The caller will free the resources.

f. **HRESULT SetAnalysisCriteria (**

```
                SAFEARRAY* pAr )                                [out]
```

This call sets the configuration of the analysis. The SAFEARRAY data would have come from a previous call to GetAnalysisCriteria.

Arguments:

i. pAr – Pointer to a SAFEARRAY that contains the criteria used to configure the analysis.

g. **HRESULT RunAnalysis (**

```
                IMoraeAnalysisResult** ppIAnalysisResult,       [out]
                IEnumMoraeTaskGroupInfo** ppTaskGrpInfo,        [out]
                IEnumMoraeRecordingInfo** ppRecInfo )           [out]
```

This call asks the plug-in to perform an analysis. Before this call is made, the SetAnalysisTargets method (described below) will be called to set up the recording and task information that the plug-in will use. Typically, the ppTaskGrpInfo and ppRecInfo outputs will simply be set to the inputs from that method. In unusual cases, the plug-in could output pointers to objects that have different recording and task information.

Arguments:

i. ppIAnalysisResult – Address of an IMoraeAnalysisResult interface pointer. This interface pointer will allow Manager to request the results of the analysis.

ii. ppTaskGrpInfo – Address of an IEnumMoraeTaskGroupInfo interface pointer. This is a typical enumeration interface (described in Morae Enumeration Interfaces) that will allow Manager to enumerate through the task groupings that went into the analysis. Typically, it will be set to the same pointer that was input in the SetAnalysisTargets call.

iii. ppRecInfo – Address of an IEnumMoraeRecordingInfo interface pointer. This is a typical enumeration interface (described in Morae Enumeration Interfaces) that will allow Manager to enumerate through the recordings that were involved in the analysis. Typically, it will be set to the same pointer that was input in the SetAnalysisTargets call.

h. HRESULT GetToolTypeUUID (

GUID* puuid) **[out]**

This call requests a unique identifier of the analysis tool. It is not used at this time.

Arguments:

i. puuid – Pointer to a GUID that should be set to a unique ID. This ID should be the same for any analysis object created by this analysis tool.

i. HRESULT SetAnalysisTargets (

IEnumMoraeTaskGroupInfo* pTaskGrpInfo, **[in]**

IEnumMoraeRecordingInfo* pRecInfo) **[in]**

This call sets the recordings and task groups that are used in the analysis.

Arguments:

i. pTaskGrpInfo – Pointer to an IEnumMoraeTaskGroupInfo interface. This is a typical enumeration interface (described in Morae Enumeration Interfaces) that can be used to step through the groups of tasks that should be used in performing the analysis.

ii. pRecInfo – Pointer to an IEnumMoraeRecordingInfo interface. This is a typical enumeration interface (described in Morae Enumeration Interfaces) that can be used to step through the recordings that should be used in performing the analysis.

3. IMoraeAnalysisResult (derives from IUnknown)

This COM interface is implemented by the plug-in to provide the results of the analysis. Manager will use this interface to determine the type of dataset that is available (scalar, array, or array with stats) and what types of graphical display are appropriate (details or means, arranged by task or by recording). It will then request the data itself.

Interface ID:

IID_IMoraeAnalysisResult = 90DC0AAE-E45D-4880-AA6A-A6F2E603524E

Enumerations:

a. AnalysisResultType

This describes the type of data that is available as a result of the analysis. Current valid values are:

i. AnalysisScalarStats – A scalar result was calculated for each task. Statistical information (mean, range, and standard deviation) are available.

ii. AnalysisArrayDescr – An array of data was calculated for each task. Each task should have a results array that has the same number of elements. There should be a description string that corresponds to each element.

iii. AnalysisArrayDescrStats – An array of statistical information (mean, range, and standard deviation) was calculated for each task. Each task should have a results array that has the same number of elements. There should be a description string that corresponds to each element in the array.

b. AnalysisDataType

This enumeration is used to specify the type of data that Manager is requesting in the call to GetGroupDataValue or GetRecordingDataValue. Possible values are:

i. AnalysisMean – Used for AnalysisScalarStats type data. Requests the mean of task values for a group of tasks or recordings.
ii. AnalysisMedian – Used for AnalysisScalarStats type data. Requests the median of task values for a group of tasks or recordings.
iii. AnalysisMin – Used for AnalysisScalarStats type data. Requests the min of task values for a group of tasks or recordings.
iv. AnalysisMax – Used for AnalysisScalarStats type data. Requests the max of task values for a group of tasks or recordings.
v. AnalysisStdDev – Used for AnalysisScalarStats type data. Requests the standard deviation of task values for a group of tasks or recordings.
vi. AnalysisStdErrMean – Used for AnalysisScalarStats type data. Requests the standard error of the mean of task values for a group of tasks or recordings.
vii. AnalysisTotal – Used for AnalysisScalarStats type data. Requests the sum of task values for a group of tasks or recordings.
viii. AnalysisArrayValues – Used for AnalysisArrayDescr type data. Requests the array of values for a group of tasks or recordings.
ix. AnalysisArrayStr – This value is no longer in use.
x. AnalysisArrayMeans – Used for AnalysisArrayDescrStats type data. Requests the array of mean values for a group of tasks or recordings.
xi. AnalysisArrayMedians – Used for AnalysisArrayDescrStats type data. Requests the array of median values for a group of tasks or recordings.
xii. AnalysisArrayMins – Used for AnalysisArrayDescrStats type data. Requests the array of min values for a group of tasks or recordings.
xiii. AnalysisArrayMaxs – Used for AnalysisArrayDescrStats type data. Requests the array of max values for a group of tasks or recordings.
xiv. AnalysisArraySteDevs – Used for AnalysisArrayDescrStats type data. Requests the array of standard deviation values for a group of tasks or recordings.
xv. AnalysisArrayStdErr – Used for AnalysisArrayDescrStats type data. Requests the array of standard error of the mean values for a group of tasks or recordings.

Methods:

a. **HRESULT GetResultName (**

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

This call requests the name of the measurement that is being calculated. This will be used to create the default label of the Y-axis for the graph.

Arguments:

i. localeID – LCID that should be used to determine language.
ii. pstr – Pointer to a BSTR variable that the plug-in should set to the name. The caller will free the string.

b. **HRESULT GetTaskGroupsLabel (**

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

This call requests the description of the analysis measurement. It will be used to create the default title for the graph.

Arguments:

i. localeID – LCID that should be used to determine language.
ii. pstr – Pointer to a BSTR variable that the plug-in should set to the description. The caller will free the string.

c. **HRESULT GetUnits (**

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

This call requests the units of the analysis measurement. It will be used to create the default label of the Y-axis for the graph.

Arguments:

i. localeID – LCID that should be used to determine language.
ii. pstr – Pointer to a BSTR variable that the plug-in should set to the units. The caller will free the string.

d. **HRESULT AllowDetailsDisplay (**

VARIANT_BOOL* pByTask,	**[out]**
VARIANT_BOOL* bByRecording)	**[out]**

This call asks whether individual measurement values (as opposed to mean values) for each task are available grouped by task or grouped by recording.

Arguments:

i. pByTask – Pointer to a VARIANT_BOOL value that should be set to VARIANT_TRUE if individual measurements grouped by task are available.
ii. pByRecording – Pointer to a VARIANT_BOOL value that should be set to VARIANT_TRUE if individual measurements grouped by recording are available.

e. **HRESULT AllowStatisticsDisplay (**

VARIANT_BOOL* pByTask,	**[out]**
VARIANT_BOOL* bByRecording)	**[out]**

This call asks whether summary statistics of measurement values for each task are available grouped by task or grouped by recording.

Arguments:

i. pByTask – Pointer to a VARIANT_BOOL value that should be set to VARIANT_TRUE if mean values grouped by task are available.
ii. pByRecording – Pointer to a VARIANT_BOOL value that should be set to VARIANT_TRUE if mean values grouped by recording are available.

f. **HRESULT GetTaskData (**

int nTaskID,	**[in]**
VARIANT * bVarData)	**[out]**

This call asks for data from an individual task.

Arguments:

i. nTaskID –Int value that identifies an instance of task. This ID value matches a task ID that is available from the IEnumMoraeTaskGroupInfo interface output by the IMoraeAnalysis::RunAnalysis method described above.
ii. pBVarData – Pointer to a VARIANT value that should be set to the data value for the task. If the result type for the analysis is AnalysisScalarStats, the vt of this VARIANT should be VT_R8. If the result type is AnalysisArrayDescr or AnalysisArrayDescrStats, the vt should be VT_R8 | VT_ARRAY, indicating that the VARIANT contains a SAFEARRAY of data. The caller will free resources.

g. **HRESULT GetResultType (**

AnalysisResultType* pType)	**[out]**

This call asks for the result type of the result set. This determines whether Manager expects results to be scalar or array data.

Arguments:

i. pType – Pointer to an AnalysisResultType enumerated value that should be set to one of the values listed above for this data type.

h. **HRESULT GetDataDescriptions (**

SAFEARRAY ppArrayDescriptions)**	**[out]**

This call requests descriptions of the data for AnalysisArrayDescr or AnalysisArrayDescrStats types of data sets. It will be used to create sub-labels that identify each element of the array data on the X-axis of the graph.

Arguments:

i. ppArrayDescriptions – Address of a SAFEARRAY pointer that should be set to a SAFEARRAY that the plug-in allocates to hold BSTRs of the descriptions. The caller will free all resources.

i. **HRESULT GetNumValuesPerTask (**

ULONG* pCount)	**[out]**

This call asks for the number of data values that were calculated for each task. For AnalysisScalarStats data sets, this should be set to 1. For array data sets, this should be set to the number of values that are in the result set for every task.

Arguments:

i. pCount – Pointer to a ULONG value that should be set to the number of results available for each task.

j. **HRESULT GetGroupDataValue (**

int nGroupID,	**[in]**
AnalysisDataType nDataType,	**[in]**
VARIANT* pVarData)	**[out]**

This call requests data for a group of tasks. Task groups consist of tasks of the same type from multiple recordings. The type of data that is requested will depend on the result type that was output by the GetResultType method.

Arguments:

i. nGroupID – Int value that identifies a task group. This ID value matches a task group ID that is available from the IEnumMoraeTaskGroupInfo interface output by the IMoraeAnalysis::RunAnalysis method described above.
ii. nDataType – A member of the AnalysisDataType enumerated type described above. The description of this enumerated type indicates which data types are applicable to each result type.
iii. pVarType – Pointer to a VARIANT value that should be filled with the data. The type of this VARIANT should be either VT_R8 or VT_R8 | VT_ARRAY, depending on whether the data type requested is an array or not. The caller will free all resources.

k. **HRESULT GetRecordingDataValue (**

REFIID uuidRec,	**[in]**
AnalysisDataType nDataType,	**[in]**
VARIANT* pVarData)	**[out]**

This call requests summary statistics of measurements for all tasks within a given recording.

Arguments:

i. uuidRec – GUID that identifies one of the recordings that was analyzed.

ii. nDataType – A member of the AnalysisDataType enumerated type described above. The description of this enumerated type indicates which data types are applicable to each result type.

iii. pVarType – Pointer to a VARIANT value that should be filled with the data. The type of this VARIANT should be either VT_R8 or VT_R8 | VT_ARRAY, depending on whether the data type requested is an array or not. The caller will free all resources.

4. IMoraeTaskInfo (derives from IUnknown)

This interface is supplied by Manager to provide information about an individual task instance that was selected by the user to be included in the analysis calculation. In the IMoraeAnalysis::RunAnalysis method, Manager supplies an IEnumMoraeTaskGroupInfo interface pointer that allows the plug-in to enumerate through a collection of IMoraeTaskGoupInfo interfaces (described below). From an IMoraeTaskGroupInfo interface, the plug-in can request an IEnumMoraeTaskInfo enumerator that allows the plug-in to enumerate through each individual IMoraeTaskInfo interface that makes up the group.

Interface ID:

IID_ IMoraeTaskInfo = 71C0DA56-C7A2-4b32-9F8F-DF94AAC4C005

Methods:

a. HRESULT GetTaskName (
BSTR* pstr) **[out]**

This call requests the name of the task.

Arguments:

i. pstr – Pointer to a BSTR that will be set to the name. The caller must free this string.

b. HRESULT GetTaskDescription (
BSTR* pstr) **[out]**

This call requests the description of the task.

Arguments:

i. pstr – Pointer to a BSTR that will be set to the description. The caller must free this string.

c. HRESULT GetParentRecordingUUID (
GUID* puuid) **[out]**

This call requests the unique ID of the recording that owns the task. This can be used to identify the IMoraeRecordingInfo or IMoraeRecording interface pointer that can provide information about the parent recording.

Arguments:

i. puuid – Pointer to a GUID that will be set to the unique ID of the parent recording.

d. **HRESULT GetTaskID (**

int* pID)	**[out]**

This call requests the ID of the task. This ID number applies only to the current analysis calculation. Manager will be able to use this ID in calls to IMoraeAnalysisResult::GetTaskData to identify a particular task instance.

Arguments:

i. pID – Pointer to an int that will be set to the ID that identifies this task instance.

e. **HRESULT GetGroupID (**

int* pID)	**[out]**

This call requests the ID of the task group that this task belongs to. A task group consists of a collection of tasks of the same type found in different recordings. This ID number applies only to the current analysis calculation. Manager will be able to use this ID in calls to IMoraeAnalysisResult::GetGroupDataValue to identify a particular task group.

Arguments:

i. pID – Pointer to an int that will be set to the ID that identifies this task group.

f. **HRESULT GetSegmentTimes (**

int nIndex,	**[in]**
VARIANT* pVarStart,	**[out]**
VARIANT* pVarEnd)	**[out]**

This call can be used to get the start and stop times for an individual segment that makes up a task.

Arguments:

i. nIndex – This is the 0-based index into the array of segments that makes up a task instance. (Note that this differs from most other index values in that it starts at 0 rather than 1.)

ii. pVarStart – Pointer to a VARIANT that will be set to a type of VT_I8. Its llVal will be set to the start time of the segment relative to the start of the recording given in 100 nanosecond units.

iii. pVarEnd – Pointer to a VARIANT that will be set to a type of VT_I8. Its llVal will be set to the end time of the segment relative to the start of the recording given in 100 nanosecond units.

g. **HRESULT GetSegmentCount (**
int* pCount) **[out]**

This call requests the number of time segments that make up this task instance.

Arguments:

i. pCount – Pointer to an int that will be set to the segment count.

h. **HRESULT GetTaskType (**
UINT* pType) **[out]**

This call requests the type of the task. This identifier is the same as the task type used in several methods in the IMoraeProject interface (described in Information Supplied by Manager) to specify a task type.

Arguments:

i. pType – Pointer to a UINT value that will be set to the task type.

5. IMoraeTaskGroupInfo (derives from IUnknown)

This interface is supplied by Manager to provide information about a group of tasks included in the analysis calculation. The group will consist of instances of a particular task type that are found in the recordings that have been selected for the analysis. In the IMoraeAnalysis::RunAnalysis method, Manager supplies an IEnumMoraeTaskGroupInfo interface pointer that allows the plug-in to enumerate through a collection of IMoraeTaskGoupInfo interfaces.

Interface ID:

IID_IMoraeTaskGroupInfo = 0A08F046-D423-4039-9BD2-3DB5F03EBE79

Methods:

a. **HRESULT GetTaskGroupName (**
BSTR* pstr) **[out]**

This call requests the name of the task group.

Arguments:

i. pstr – Pointer to a BSTR that will be set to the name. The caller must free this string.

b. **HRESULT GetTaskGroupDescription (**
BSTR* pstr) **[out]**

This call requests the description of the task group.

Arguments:

ii. pstr – Pointer to a BSTR that will be set to the description. The caller must free this string.

c. **HRESULT GetTaskGroupID (**
int* pID) **[out]**

This call requests the ID of the task group. This ID number applies only to the current analysis calculation. Manager will be able to use this ID in calls to IMoraeAnalysisResult::GetGroupDataValue to identify a task group.

Arguments:

i. pID – Pointer to an int that will be set to the ID that identifies this task group.

d. **HRESULT EnumTasks (**
IEnumMoraeTaskInfo ppIEnum)** **[out]**

This call requests an enumerator that will allow the caller to get information about each individual task instance that makes up this group. IEnumMoraeTaskInfo is described in Morae Enumeration Interfaces. It allows enumeration through a collection of objects that implement IMoraeTaskInfo.

Arguments:

i. ppIEnum – Pointer to an IEnumMoraeTaskInfo interface pointer that allows enumeration through a collection of IMoraeTaskInfo objects. The caller must call Release on this interface when it is finished with it.

e. **HRESULT GetTaskCount (**
int* pCount) **[out]**

This call requests the number of task instances that make up this task group.

Arguments:

i. pCount – Pointer to an int value that will be set to the task count.

6. IMoraeRecordingInfo (derives from IUnknown)

This interface is supplied by Manager to provide information about an individual recording that was selected by the user to be included in the analysis calculation. In the IMoraeAnalysis::RunAnalysis method, Manager supplies an IEnumMoraeRecordingInfo interface pointer that allows the plug-in to enumerate through a collection of IMoraeRecordingInfo interfaces.

Interface ID:

IID_IMoraeRecordingInfo = 5EEF1577-E745-4158-BF08-822B4E8D6395

Methods:

a. **HRESULT GetRecordingName (**
BSTR* pstr) **[out]**

This call requests the name of the recording.

Arguments:

i. pstr – Pointer to a BSTR that will be set to the name. The caller must free this string.

b. **HRESULT GetRecordingDescription (**

BSTR* pstr) **[out]**

This call requests the description of the recording.

Arguments:

i. pstr – Pointer to a BSTR that will be set to the description. The caller must free this string.

c. **HRESULT GetRecordingUUID (**

GUID* puuid) **[out]**

This call requests the unique ID of the recording. This can be used to identify the IMoraeRecordingInfo or IMoraeRecording interface pointer that can provide information about the parent recording.

Arguments:

i. puuid – Pointer to a GUID that will be set to the unique ID of the parent recording.