Landmarks API

Landmarks Python extension for Symbian S60 3rd edition and higher

Martin Wibbels
martin.wibbels@novay.nl

Contents:

1 Introduction

The API is split up in a number of classes. The starting point is the LandmarkDatabase. Most other classes either require that class for their creation, or don't make much sense without it (like Landmark instances), or manage/describe them like the DatabaseManager.

2 Constructors/static methods

2.1 landmarks.OpenDefaultDatabase()

Opens the default landmarks database and constructs a LandmarkDatabase instance.

2.2 landmarks.OpenDatabase(string uri)

Opens the landmarks database at the specified uri and constructs a LandmarkDatabase instance.

2.3 landmarks.CreateLandmark()

Creates a new, empty Landmark instance. The landmark can be initialized and then added to the landmark database using AddLandmark.

2.4 landmarks.CreateLandmarkCategory()

Creates a new, empty LandmarkCategory instance. The category can be initialized and then added to the landmark database using AddCategory on the CategoryManager class.

2.5 landmarks.CreateAreaCriteria(double southLat, double northLat, double westLong, double eastLong)

Creates a new Area search criteria. Can be used in searches using StartLandmarkSearch on the LandmarkSearch class.

2.6 landmarks.CreateCatNameCriteria(string namePattern)

Creates a new category name search criteria. Wild-card characters "?" and "*" are supported in the name pattern. Can be used in searches using StartCategorySearch on the LandmarkSearch class.

2.7 landmarks.CreateCategoryCriteria(int itemId, int globalCategory, string categoryName)

Creates a new category search criteria. Specify either a category item id, a global category id or a category name. Specifying a 0 or empty string for the other elements. Can be used in searches using StartLandmarkSearch on the LandmarkSearch class.

2.8 landmarks.CreateCompositeCriteria(int compositionType, [SearchCriteria] criteria);

Creates a new composite search. The compositionType specifies the way in which the criteria should be combined (currently only ECompositionAnd is supported), criteria is a list of search criteria. Note that nested composite criteria are not allowed. Can be used in searches using StartLandmarkSearch on the LandmarkSearch class.

2.9 landmarks.CreateIdListCriteria([int] idList)

Creates a new id list search: all elements that have an id that is in the idList match. This criterion must be combined with other search criteria using CreateCompositeCriteria. It is of no use on its own. If it is not combined with another criterion, StartLandmarkSearch will fail with error code KErrArgument.

2.10 landmarks.CreateNearestCriteria(double latitude, double longitude, double altitude, boolean useCoverageRadius, double distance)

Creates a nearest search criteria. Will search for nearest landmarks and return them in order of distance. UseCoverageRadius specifies if the coverage radius should be used. If coverage radius is used, the distance to the landmark coverage area border is used instead of the distance to the landmark coverage area center. If the source coordinate is inside the landmark coverage area, the distance is considered zero. Can be used in searches using StartLandmarkSearch on the LandmarkSearch class.

2.11 landmarks.CreateTextCriteria(string text, int attributes, [int] positionFieldIds)

Creates a text search criteria. The specified text will be searched in the attributes specified (eg ELandmarkName, EDescription) and/or the specified list of fields. The

position fields ids can be one or more of the EPositionField* constants. Can be used in searches using StartLandmarkSearch on the LandmarkSearch class.

2.12 Landmarks.ShowSelectCategoryDialog(bool showEmptyCategories, int selectedItem)

Shows a dialog that allows the user to select a single category. Set showEmptyCategories to true if categories not assigned to any landmarks should be selectable, false otherwise. SelectedItem contains the initially selected category, or KPosLmNullItemId for no initial selection (selectedItem is currently ignored by Symbian). The result will be a tuple consisting of (in order):

Int	0 if the user cancelled the dialog, non-zero otherwise. If 0 then all other values will be 0 as well
Int	Id of the selected category
LandmarkDatabase	Database that contains the selected category

2.13 landmarks.ShowSelectLandmarkDialog(int selectedItem)

Shows a dialog that allows the user to select a single category. SelectedItem contains the initially selected landmark, or KPosLmNullItemId for no initial selection (selectedItem is currently ignored by Symbian). The result will be a tuple consisting of (in order):

Int	0 if the user cancelled the dialog, non-zero otherwise. If 0 then all other values will be 0 as well
Int	Id of the selected landmark
LandmarkDatabase	Database that contains the selected landmark

2.14 landmarks.ShowSelectMultipleCategoryDialog(bool showEmptyCategories, [int] selectedItems)

Shows a dialog that allows the user to select multiple categories. Set showEmptyCategories to true if categories not assigned to any landmarks should be selectable, false otherwise. SelectedItems contains the list of initially selected categories (empty list for no initial selection). The result will be a tuple consisting of (in order):

Int	0 if the user cancelled the dialog, non-zero otherwise. If 0 then all other values will be 0 as well	
[]	List of tuples consisting of (in order):	
	Int	Id of a selected category
	LandmarkDatabase	Database that contains this selected category

Note that for every unique LandmarkDatabase only one instance is returned.

2.15 landmarks.ShowSelectMultipleLandmarkDialog([int] selectedItems)

Shows a dialog that allows the user to select multiple categories. SelectedItems contains the list of initially selected landmarks (empty list for no initial selection). The result will be a tuple consisting of (in order):

Int	0 if the user cancelled the dialog, non-zero otherwise. If 0 then all other values

	will be 0 as well	
[]	List of tuples consisting of (in order):	
	Int	Id of a selected landmark
	LandmarkDatabase	Database that contains this selected landmark

Note that for every unique LandmarkDatabase only one instance is returned.

2.16 landmarks.ReleaseLandmarkResources()

When using some landmark services, e.g. LandmarkDatabase, resources are allocated globally. To release these resources, ReleaseLandmarkResources must be called , otherwise the client may receive an ALLOC panic.

2.17 landmarks.CreateDatabaseManager()

Create a DatabaseManager instance.

2.18 landmarks.CreateDatabaseInfo(string uri)

Creates a new DatabaseInfo instance with the specified uri. Note that there are strict requirements for the uri.

2.19 landmarks.PackLandmark(Landmark landmark)

Packs the specified landmark into a binary buffer. The result is a 8-bit string (“s”)

2.20 landmarks.PackCategory(LandmarkCategory category)

Packs the specified category into a binary buffer. The result is a 8-bit string (“s”)

2.21 landmarks.UnpackLandmark(string buffer)

Unpacks a landmark from the specified 8-bit string (“s”). The result is a Landmark instance.

2.22 landmarks.UnpackCategory(string buffer)

Unpacks a category from the specified 8-bit string (“s”). The result is a LandmarkCategory instance.

2.23 landmarks.CreateLandmarkEncoder(string mimeType)

Creates an encoder for encoding landmarks into the specified mime-type. Note that mimeType should be an 8-bit string (“s”). The result is a LandmarkEncoder instance. Supported mime types are:

Name	Mime-type	WDP port number
Landmark Collection (XML)	application/vnd.nokia.landmarkcollection+xml	N/A
Landmark (WBXML)	application/vnd.nokia.landmark+wbxml	3969 (0x0F81)

2.24 landmarks.CreateLandmarkParser(string mimeType)

Creates an parser for parsing landmarks from the specified mime-type. Note that mimeType should be an 8-bit string ("s"). The result is a LandmarkParser instance. See landmarks.CreateLandmarkEncoder for supported mime-types.

3 LandmarkDatabase class

The heart of the Landmark API: stores landmarks and categories, allows searches etc.

3.1 LandmarkDatabase::Close()

Closes the database. The instance may not be used after this. Closing the database is not mandatory: it will be closed automatically once Python decides that the LandmarkDatabase is no longer referenced. Note that closing the database also renders any and all dependent CategoryManager and LandmarkSearch instances obsolete.

3.2 LandmarkDatabase::LandmarkIterator(int sortAttribute, int sortOrder)

Creates a ItemIterator instance that iterates over all landmarks in the database. The sortAttribute determines based on which attribute the landmarks should be sorted (ENoAttribute means no sorting), sortOrder can be EAscending or EDescending.

3.3 LandmarkDatabase::CreateSearch()

Creates a LandmarkSearch instance that searches on this database.

3.4 LandmarkDatabase::CreateCategoryManager()

Creates a CategoryManager instance that manages the categories in this database.

3.5 LandmarkDatabase::AddLandmark(Landmark landmark)

Adds the specified landmark to the database. Returns the id assigned to the landmark.

3.6 LandmarkDatabase::UpdateLandmark(Landmark landmark)

Updates the specified landmark to the database.

3.7 LandmarkDatabase::RemoveLandmark(int id)

Removes the landmark with the specified id from the database.

3.8 LandmarkDatabase::RemoveAllLandmarks()

Removes all landmarks from the database. Returns a LmOperation instance. Whilst the operation is not finished the database may be locked.

3.9 LandmarkDatabase::ReadLandmark(int id)

Reads the landmark with the specified id from the database. Returns a Landmark instance.

3.10 LandmarkDatabase::ReadPartialLandmark(int id)

Reads the landmark with the specified id partially from the database. Returns a Landmark instance. Which fields are read is specified using the SetPartialReadParameters method.

3.11 LandmarkDatabase::PartialReadParameters()

Gets the partial read parameters, used in the ReadPartialLandmark method. The result will be a tuple consisting of (in order):

Int	Attributes to be read, one or a combination of attribute flags (ELandmarkName, EDescription, …)
[int]	List of position fields to be read. Each element in the list will be a EPositionField* value.

3.12 LandmarkDatabase::SetPartialReadParameters(int attributes, [int] positionFieldIds)

Sets the partial read parameters, used in the ReadPartialLandmark method. The attributes are the attributes to be read: one or a combination of attribute flags (ELandmarkName, EDescription, …). The positionFieldIds is a list of position fields to be read. Each element in the list will be a EPositionField* value.

3.13 LandmarkDatabase::ShowEditDialog(int landmarkId, int editorAttributes, int editorMode, bool mapAndNavigationAllowed)

Shows a dialog that allows the user to edit the existing landmark with the specified id. The editorAttributes are a combination of ELmk* values, they determine which fields can be viewed/editied. EditorMode is either ELmkEditor or ELmkViewer and determines wheter the dialog will be for viewing or editing. Finally mapAndNavigationAllowed determines whether the Map and Navigation related Menu Options will be enabled or not. Currently this does not seem to work. Returns 0 if the user cancelled the dialog, non-zero otherwise.

3.14 LandmarkDatabase::ShowNewDialog(Landmark landmark, int editorAttributes, int editorMode, bool mapAndNavigationAllowed)

Shows a dialog that allows the user to edit the specified new landmark. The editorAttributes are a combination of ELmk* values, they determine which fields can be viewed/editied. EditorMode is either ELmkEditor or ELmkViewer and determines wheter the dialog will be for viewing or editing. Finally mapAndNavigationAllowed determines whether the Map and Navigation related Menu Options will be enabled or not. Currently this does not seem to work. Returns 0 if the user cancelled the dialog, non-zero otherwise.

3.15 LandmarkDatabase::ExportLandmarks(LandmarkEncoder encoder, [int] landmarkIds, int transferOptions)

Exports the landmarks, whose ids are in the landmarkIds list, from this database using the specified LandmarkEncoder instance. The transferOptions determine how categories are

treated (one of EDefaultOptions, EIncludeCategories, EIncludeGlobalCategoryNames, ESupressCategoryCreation).

3.16 LandmarkDatabase::ImportLandmarkSet(LandmarkParser encoder, [int] landmarkIndexes, int transferOptions)

Imports the landmarks, whose indexes are in the landmarkIndexes list, into this database using the specified LandmarkParser instance. The transferOptions determine how categories are treated (one of EDefaultOptions, EIncludeCategories, EIncludeGlobalCategoryNames, ESupressCategoryCreation). Returns a LmOperation instance. See ImportedLandmarksIterator.

3.17 LandmarkDatabase::ImportLandmarks(LandmarkParser encoder, int transferOptions)

Imports the landmarks into this database using the specified LandmarkParser instance. The transferOptions determine how categories are treated (one of EDefaultOptions, EIncludeCategories, EIncludeGlobalCategoryNames, ESupressCategoryCreation). Returns a LmOperation instance. See ImportedLandmarksIterator.

3.18 LandmarkDatabase::ImportedLandmarksIterator(LmOperation operation)

Gets an iterator through the set of imported landmarks, based on an import operation (LmOperation instance) created using ImportLandmarks or ImportLansmarkSet. Returns an ItemIterator instance.

3.19 LandmarkDatabase::PreparePartialLandmarks([int] landmarkIds)

Prepares a partial read operation for the specified list of landmarks. What fields are read is determined by SetPartialReadParameters. Returns a LmOperation instance. See TakePreparedPartialLandmarks.

3.20 LandmarkDatabase::TakePreparedPartialLandmarks(LmOperation operation)

Retrieves the partially read landmarks from a **completed** read operation initiated using PreparePartialLandmarks. Returns a list if Landmark instances.

4 SearchCriteria class

All search criteria (Area, CatName, Category, Nearest, …) are mapped to instances of this class.

4.1 SearchCriteria::Close()

Closes the criteria. The instance may not be used after this. Closing the criteria is not mandatory: it will be closed automatically once Python decides that the SearchCriteria is no longer referenced.

5 LandmarkSearch class

Represents a search in the landmark database. Created using LandmarkDatabase::CreateSearch. Its name is slightly misleading, as it also allows searching for categories. Its main operations are StartLandmarkSearch for searching landmarks and StartCategorySearch for searching categories.

5.1 LandmarkSearch::Close()

Closes the search. The instance may not be used after this. Closing the search is not mandatory: it will be closed automatically once Python decides that the LandmarkSearch is no longer referenced.

5.2 LandmarkSearch::SetMaxNumOfMatches(int maxNum)

Sets the maximum number of matches found during the search. Use KPosLmMaxNumOfMatchesUnlimited to apply no limit.

5.3 LandmarkSearch::MaxNumOfMatches()

Gets the maximum number of matches found during the search. Will be KPosLmMaxNumOfMatchesUnlimited if no limit is set.

5.4 LandmarkSearch::StartLandmarkSearch(SearchCriteria criteria, int sortAttribute, int sortOrder, boolean searchOnlyPreviousMatches)

Starts a search for landmarks using the specified criteria. The sortAttribute determines based on which attribute the results should be sorted (ENoAttribute means no sorting), sortOrder can be EAscending or EDescending. If searchOnlyPreviousMatches is true then the search will only be performed within the results of previous search. Returns an LmOperation instance.

5.5 LandmarkSearch::StartCategorySearch(SearchCriteria criteria, int categorySortOrder, boolean searchOnlyPreviousMatches)

Starts a search for categories using the specified criteria. The categorySortAttribute determines how the results should be sorted (ECategorySortOrderNone, ECategorySortOrderNameAscending or ECategorySortOrderNameDescending). If searchOnlyPreviousMatches is true then the search will only be performed within the results of previous search. Returns an LmOperation instance.

5.6 LandmarkSearch::NumOfMatches()

Gets the number of matches so far in the search.

5.7 LandmarkSearch::MatchIterator()

Gets an iterator over the matching items, be they Landmark or LandmarkCategory instances (based on whether StartLandmarkSearch or StartCategorySearch was called). Returns an ItemIterator instance.

6 LmOperation class

Long running operations in the Landmarks API return an instance of this class so that the client can run it incrementally and check the progress of the operation. This can be done either using repeated calls to NextStep(), or by calling Execute() once. A long running operation can be cancelled using Cancel(), though the result of the partially executed operation will of course be undefined.

6.1 LmOperation::Close()

Closes the operation. The instance may not be used after this. Closing the operation is not mandatory: it will be closed automatically once Python decides that the LmOperation is no longer referenced. Note: this is identical to Cancel().

6.2 LmOperation::Cancel()

Cancels the operation. The instance may not be used after this. Note: this is identical to Close().

6.3 LmOperation::NextStep()

Incrementally executes the long running operation. The result will be a progress indication [0.0 … 1.0] where 0.0 indicates that the operation has not started and 1.0 indicates that the operation has completed. Note: you should keep calling NextStep() until the result equals 2.0. This may seem strange, but in theory NextStep could return a progress value of 1.0 whilst the operation is not yet finished: the progress value is just an indication, internally a completion flag is used to signal the actual completion of the operation.

6.4 LmOperation::Execute()

Synchronously executes the long running operation. This is equivalent to calling NextStep() in a tight loop until the operation is completed. Note: you cannot mix NextStep() and Execute() usage: either use NextStep() for as long as necessary, or use Execute() once.

7 ItemIterator class

Iterator over items (Landmark or LandmarkCategory instances). The iterator does not return the actual instances but rather their ids. The actual landmark or category can then subsequently be read using LandmarkDatabase::ReadLandmark or CategoryManager::ReadCategory.

7.1 ItemIterator::Close()

Closes the iterator. The instance may not be used after this. Closing the iterator is not mandatory: it will be closed automatically once Python decides that the ItemIterator is no longer referenced.

7.2 ItemIterator::Next()

Moves to the next element in the iterator and returns its id. This is either the id of a landmark or the id of a category, depending on the method that returned the ItemIterator instance.

7.3 ItemIterator::Reset()

Resets the iterator. The next call to Next will return the first item ID in the iterated set.

7.4 ItemIterator::NumOfItems()

Gets the number of items in the iterated set.

7.5 ItemIterator::GetItemIds(int index, int count)

Gets a list of count item ids from the iterated set, starting at index. Note that 0 <= index < NumOfItems and 0 <= (index + count – 1) < NumOfItems, otherwise the operation will throw an exception.

8 Landmark class

Represents an actual landmark as stored in the landmark database.

8.1 Landmark::Close()

Closes the landmark. The instance may not be used after this. Closing the landmark is not mandatory: it will be closed automatically once Python decides that the Landmark is no longer referenced.

8.2 Landmark::IsPartial()

Returns true if the landmark is partial, ie. The result of a ReadPartialLandmark call.

8.3 Landmark::GetLandmarkName()

Gets the name of the landmark, or None if none set

8.4 Landmark::SetLandmarkName(string name)

Sets the name of the landmark

8.5 Landmark::GetLandmarkDescription()

Gets the description of the landmark, or None if none set

8.6 Landmark::SetLandmarkDescription (string description)

Sets the description of the landmark

8.7 Landmark::GetPosition()

Gets the position of the landmark, or None if none set. If set the result will be a tuple consisting of (in order):

Double	Latitude
Double	Longitude
Double	Altitude
Double	Horizontal accuracy
Double	Vertical accuracy

8.8 Landmark::SetPosition(double latitude, double longitude, double altitude, double horizontalAccuracy, double verticalAccuracy)

Sets the position of the landmark

8.9 Landmark::GetCoverageRadius()

Gets the coverage radius of the landmark, or None if none set.

8.10 Landmark::StCoverageRadius(double radius)

Sets the coverage radius of the landmark.

8.11 Landmark::LandmarkId()

Gets the unique id of the landmark. Will be KPosLmNullItemId if the landmark has not been added to the database yet.

8.12 Landmark::AddCategory(int categoryId)

Adds the specified category to the landmark.

8.13 Landmark::RemoveCategory(int categoryId)

Removes the specified category from the landmark.

8.14 Landmark::GetCategories()

Gets a list of the ids of the categories associated with this landmark.

8.15 Landmark::IsPositionFieldAvailable(int positionFieldId)

Checks if the specified position field id (one of the EPositionField* constants) is available for this landmark.

8.16 Landmark::NumOfAvailablePositionFields()

Gets the number of available position fields for this landmark.

8.17 Landmark::FirstPositionFieldId()

Gets the first available position field id. See NextPositionFieldId. Returns EPositionFieldNone if the landmark has no available position fields.

8.18 Landmark::NextPositionFieldId(int previousId)

Gets the next available position field id, based on the previous id. See FirstPositionFieldId. Returns EPositionFieldNone if there are no more fields.

8.19 Landmark::GetPositionField(int positionFieldId)

Gets the value of the specified position field id as a string, or None if the specified field is not available. See IsPositionFieldAvailable and FirstPositionFieldId.

8.20 Landmark::SetPositionField(int positionFieldId, string value)

Sets the value of the specified position field to the specified value. See IsPositionFieldAvailable, FirstPositionFieldId and the EPositionField* constants.

8.21 Landmark::RemovePositionField(int positionFieldId)

Removes the value for the specified position field. After this call IsPositionFieldAvailable for the specified id will return false, and GetPositionField will return None.

8.22 Landmark::GetIcon()

Gets the icon associated with this landmark. The result will be a tuple consisting of (in order):

String	Icon file name
Int	The index of the icon within the icon file
Int	The index of the icon mask within the icon file

8.23 Landmark::SetIcon(string iconFile, int iconIndex, int iconMaskIndex)

Sets the icon associated with this landmark. IconFile is the name of the file that contains the icon, iconIndex is the index of the icon within that file, iconMaskIndex is the index of the icon mask within that file.

8.24 Landmark::RemoveLandmarkAttributes(int attributes)

Removes the values for the specified attributes. Attributes should be one or a combination of attribute flags (ELandmarkName, EDescription, …). For example: after calling RemoveLandmarkAttributes(EDescription) a call to GetLandmarkDescription will return None.

9 CategoryManager class

Manages the categories for a landmark database. Created using LandmarkDatabase::CreateCategoryManager. Categories can be global or local.

9.1 CategoryManager::Close()

Closes the category manager. The instance may not be used after this. Closing the manager is not mandatory: it will be closed automatically once Python decides that the CategoryManager is no longer referenced.

9.2 CategoryManager::CategoryIterator(int categorySortOrder)

Creates a ItemIterator instance that iterates over all categories in the database. The categorySortOrder determines how the categories should be sorted (ECategorySortOrderNone, ECategorySortOrderNameAscending or ECategorySortOrderNameDescending).

9.3 CategoryManager::ReferencedCategoryIterator(int categorySortOrder)

Creates a ItemIterator instance that iterates over all referenced categories in the database. A category is referenced if there are landmarks in the database which contain this category. The categorySortOrder determines how the categories should be sorted (ECategorySortOrderNone, ECategorySortOrderNameAscending or ECategorySortOrderNameDescending).

9.4 CategoryManager::ReadCategory(int id)

Reads the category with the specified id from the database. Returns a LandmarkCategory instance.

9.5 CategoryManager::AddCategory(LandmarkCategory category)

Adds the specified category to the database. Returns the id assigned to the category.

9.6 CategoryManager::UpdateCategory(LandmarkCategory category)

Updates the specified category to the database.

9.7 CategoryManager::RemoveCategory (int id)

Removes the category with the specified id from the database.

9.8 CategoryManager::GetCategory(string name)

Gets the id of the category with the specified name. Returns KPosLmNullItemId if the category was not found.

9.9 CategoryManager::GetGlobalCategory(int globalId)

Gets the local category id for the specified global category id.

9.10 CategoryManager::GlobalCategoryName(int globalId)

Gets name of the specified global category. Will return None if the specified global id is unknown.

9.11 CategoryManager::ResetGlobalCategories()

Global categories usually have a default name and icon, but they can be changed. This function resets the names and icons to the default ones. Returns a LmOperation instance.

9.12 CategoryManager::AddCategoryToLandmarks(int categoryId, [int] landmarkIds)

Adds the specified category to all landmarks in the specified list of landmark ids. Returns a LmOperation instance.

9.13 CategoryManager::RemoveCategoryFromLandmarks(int categoryId, [int] landmarkIds)

Removes the specified category from all landmarks in the specified list of landmark ids. Returns a LmOperation instance.

9.14 CategoryManager::RemoveCategories([int] categoryIds)

Removes the specified categories from the database. Will also remove those categories from all landmarks that contain them. Returns a LmOperation instance.

10 LandmarkCategory class

Represents a category in the database. Managing category instances is done through the CategoryManager. Adding categories to/removing categories from a landmark is done using the Landmark class.

10.1 LandmarkCategory::Close()

Closes the category. The instance may not be used after this. Closing the category is not mandatory: it will be closed automatically once Python decides that the LandmarkCategory is no longer referenced.

10.2 LandmarkCategory::GetIcon()

Gets the icon associated with this category. The result will be a tuple consisting of (in order):

String	Icon file name
Int	The index of the icon within the icon file
Int	The index of the icon mask within the icon file

10.3 LandmarkCategory::SetIcon(string iconFile, int iconIndex, int iconMaskIndex)

Sets the icon associated with this category. IconFile is the name of the file that contains the icon, iconIndex is the index of the icon within that file, iconMaskIndex is the index of the icon mask within that file.

10.4 LandmarkCategory::GetCategoryName()

Gets the name of the category, or None if none set

10.5 LandmarkCategory::SetCategoryName(string name)

Sets the name of the category

10.6 LandmarkCategory::GlobalCategory()

Gets the global category id for this category, if any. Will return KPosLmNullGlobalCategory otherwise.

10.7 LandmarkCategory::CategoryId()

Gets the id of this category. Will be KPosLmNullItemId if the category has not been added to the database yet.

10.8 LandmarkCategory::RemoveCategoryAttributes(int attributes)

Removes the values for the specified attributes. Attributes should be one or a combination of attribute flags (ECategoryName or EIcon). For example: after calling RemoveCategoryAttributes(ECategoryName) a call to GetCategoryName will return None.

11 DatabaseManager class

Manages the landmark databases. Created using landmarks.CreateDatabaseManager. Allows creating/deleting local databases and registering/unregistering remote databases, retrieving/changing settings etc.

11.1 DatabaseManager::Close()

Closes the database manager. The instance may not be used after this. Closing the manager is not mandatory: it will be closed automatically once Python decides that the DatabaseManager is no longer referenced.

11.2 DatabaseManager::DefaultDatabaseUri()

Gets the uri of the default database

11.3 DatabaseManager::SetDefaultDatabaseUri(string uri)

Sets the uri of the default database

11.4 DatabaseManager::ListDatabases(string protocol)

Lists the uris of all databases if protocol is an empty string, or the uris of all databases that have the specified protocol. The result is a list of strings.

11.5 DatabaseManager::ListDatabaseInfos(string protocol)

Lists the DatabaseInfo instances of all databases if protocol is an empty string, or the DatabaseInfo instances of all databases that have the specified protocol in their uri.

11.6 DatabaseManager::RegisterDatabase(DatabaseInfo info)

Registers the specified info, so that it will appear in future ListDatabases and ListDatabaseInfos calls. Note: the database must be a remote database, not a local one.

11.7 DatabaseManager::UnregisterDatabase(string uri)

Unregisters the database with the specified uri, so that it will no longer appear in future ListDatabases and ListDatabaseInfos calls. Note: the database must be a remote database, not a local one.

11.8 DatabaseManager::UnregisterAllDatabases(string protocol)

Unregisters all databases that have the specified protocol in their uri, so that it will no longer appear in future ListDatabases and ListDatabaseInfos calls. Note: only affects remote databases. The specified protocol may not be an empty string.

11.9 DatabaseManager::ModifyDatabaseSettings(string uri, DatabaseSettings settings)

Modifies the settings of the database with the specified uri to the specified settings. The documentation states that you should get the DatabaseInfo, get the DatabaseSettings from there, change those settings and then call this method. (So why the uri?)

11.10 DatabaseManager::GetDatabaseInfo(DatabaseInfo info)

Gets the DatabaseInfo for the database. Create a DatabaseInfo instance with a specific uri, then call this method to retrieve all other DatabaseInfo members.

11.11 DatabaseManager::CreateDatabase(DatabaseInfo info)

Creates the database for the specified DatabaseInfo instance.

11.12 DatabaseManager::DeleteDatabase(string uri)

Deletes the database for the specified uri.

11.13 DatabaseManager::DatabaseExists(string uri)

Returns true if the database for the specified uri exists, false otherwise.

11.14 DatabaseManager::CopyDatabase(string srcUri, string destUri)

Copies the database from the srcUri to the destUri. No database may already exist for the destUri.

12 DatabaseInfo class

Info on a database. A new instance can be created using landmarks.CreateDatabaseInfo, for creating new databases or retrieving the info of a particular database using DatabaseManager::GetDatabaseInfo. Alternatively DatabaseManager::ListDatabaseInfos returns a list of DatabaseInfo instances for existing databases.

12.1 DatabaseInfo::Close()

Closes the database info. The instance may not be used after this. Closing the info is not mandatory: it will be closed automatically once Python decides that the DatabaseInfo is no longer referenced.

12.2 DatabaseInfo::DatabaseUri()

Gets the uri of the database.

12.3 DatabaseInfo::Protocol()

Gets the protocol in the uri of the database.

12.4 DatabaseInfo::IsDefault()

Returns true if this is the default database, false otherwise.

12.5 DatabaseInfo::DatabaseMedia()

Gets which storage media the database resides in. One of the EMedia* constants.

12.6 DatabaseInfo::DatabaseDrive()

Gets the drive letter for the drive where the database resides. Will be 0 if the letter is not set, or not applicable for the uri protocol (for example with remote databases).

12.7 DatabaseInfo::Settings()

Gets the DatabaseSettings for this database.

12.8 DatabaseInfo::Size()

Gets the size of the database in bytes.

13 DatabaseSettings class

Contains the settings of a database, currently only its display name.

13.1 DatabaseSettings::Close()

Closes the database settings. The instance may not be used after this. Closing the settings is not mandatory: it will be closed automatically once Python decides that the DatabaseSettings is no longer referenced.

13.2 DatabaseSettings::IsAttributeSet(int attribute)

Returns true if the specified attribute is set, false otherwise. Currently only EName is supported.

13.3 DatabaseSettings::UnsetAttribute(int attribute)

Unsets the specified attribute. Currently only EName is supported.

13.4 DatabaseSettings::DatabaseName()

Gets the display name for the database.

13.5 DatabaseSettings::SetDatabaseName(string name)

Sets the display name for the database. You must call DatabaseManager::ModifyDatabaseSettings before the change becomes effective.

14 CBufBase class

A dynamic buffer that does not necessarily occupy a single contiguous memory range.

14.1 CBufBase::Close()

Closes the CBufBase. The instance may not be used after this. Closing the CBufBase is not mandatory: it will be closed automatically once Python decides that the CBufBase is no longer referenced.

14.2 CBufBase::Size()

Returns the size of the buffer, in bytes.

14.3 CBufBase::Ptr(int aPos)

Returns an 8-bit string (“s”) which contains all from the specified data byte until the end of the contiguous region containing that byte. Note that 0 <= aPos < Size(). This operation is required because the buffer may occupy multiple memory regions. A way of using this is:

```
Index = 0;
Size = cBufBase.Size();
While (Index < Size):
      S = cBufBase.Ptr(Index);
      print S;
      Index = Index + len(S);

cBufBase.Close();
```

15 LandmarkEncoder class

An encoder for encoding landmarks to a particular mime-type. Created using landmarks.CreateLandmarkEncoder. See SetOutputFile or SetUseOutputBuffer.
The basic protocol for encoding is to

1) define where to write the output to by calling SetUseOutputBuffer or SetOutputFile,
2) optionally add collection data using AddCollectionData
3) add landmark data to encode by using functions in LandmarkEncoder and/or LandmarkDatabase::ExportLandmarks
4) finalize the encoding by calling FinalizeEncoding.

15.1 LandmarkEncoder::Close()

Closes the landmark encoder. The instance may not be used after this. Closing the encoder is not mandatory: it will be closed automatically once Python decides that the LandmarkEncoder is no longer referenced.

15.2 LandmarkEncoder::AddCategoryForLatestLandmark(LandmarkCategory category)

Adds a landmark category for the most recently added landmark.

15.3 LandmarkEncoder::AddLandmark(Landmark landmark)

Adds a landmark to be encoded.

15.4 LandmarkEncoder::AddCollectionData(int dataId, string data)

Adds landmark collection data to be encoded. Landmark collection data is generic information about the landmark collection. It can for instance contain a name for the landmark collection. Predefined collection attributes are defined by the EPosLmCollData* constants but also format specific collection meta data can be defined.

15.5 LandmarkEncoder::FinalizeEncoding()

Finalizes the encoding process. Flushes and closes the output file, or flushes and compresses the output buffer.

15.6 LandmarkEncoder::SetOutputFile(string fileName)

Sets the name of the file to write to. File must not yet exist.

15.7 LandmarkEncoder::SetUseOutputBuffer()

Lets the encoder use an output buffer. Returns a CBufBase instance.

16 LandmarkParser class

A parser for parsing landmarks from data with a particular mime-type. Created using landmarks.CreateLandmarkParser. See SetInputFile or SetInputBuffer.

16.1 LandmarkParser::Close()

Closes the landmark parser. The instance may not be used after this. Closing the parser is not mandatory: it will be closed automatically once Python decides that the LandmarkParser is no longer referenced.

16.2 LandmarkParser::SetInputFile(string fileName)

Sets the name of the file to read from.

16.3 LandmarkParser::SetInputBuffer(string data)

Sets the input data that should be parsed. The data should be an 8-bit string ("s").

16.4 LandmarkParser::ParseContent(boolean buildIndex)

Parses the content specified by SetInputFile or SetInputBuffer.
The client can specify that the parser should build an index while parsing using buildIndex. Building an index uses more memory but it allows unlimited direct access to all parsed data. If the parser does not support indexing, this call will fail.
Returns a LmOperation instance.

16.5 LandmarkParser::NumOfParsedLandmarks()

Gets the number of parsed landmarks.

16.6 LandmarkParser::FirstCollectionDataId()

Gets first id of the available collection data, or 0 if no collection data is available.

16.7 LandmarkParser::NextCollectionDataId(int previousId)

Gets the next id of the available collection data, after the specified previousId, or 0 if no more collection data is available.

16.8 LandmarkParser::CollectionData(int dataId)

Gets the collection data for the specified data id, as a string.

16.9 LandmarkParser::Landmark(int index)

Returns the parsed landmark at the specified index (if buildIndex was true when calling ParseContent), or the last parsed landmark if index = KPosLastParsedLandmark. This does not require an index. Note that LmOperation::NextStep() will parse exactly one landmark.

16.10 LandmarkParser::LandmarkCategory(int categoryId)

Gets the LandmarkCategory instance for a category id associated with a parsed landmark. The categoryId should be one of the ids found with the parsed landmarks.