
Landmarks Database Management API Specification

Version 1.0
July 12, 2005

S E R I E S **60** P L A T F O R M

Legal Notice

Copyright © 2005 Nokia Corporation. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

Contents

1.	Document Control	5
1.1	References.....	5
1.2	Documentation Conventions.....	5
1.3	Abbreviations	5
1.4	Definitions.....	5
2.	Purpose	6
2.1	List of the Required Interfaces	6
2.2	Constraints	6
3.	Technical Specification	7
3.1	Type of the Interface	7
3.2	Interface Class Structure.....	7
3.3	Usage	7
3.3.1	Protocol.....	7
3.3.2	Error handling.....	8
3.3.3	Memory overhead.....	8
3.3.4	Extensions to the API	8
3.4	Example	8
3.4.1	Listing databases and getting database information.....	8
3.4.2	Moving a database	9
3.4.3	Setting a name for a database	9
3.5	Detailed Description	9
3.5.1	CPosLmDatabaseManager	10
3.5.2	HPosLmDatabaseInfo	18
3.5.3	TPosLmDatabaseEventType.....	21
3.5.4	TPosLmDatabaseSettings.....	22
3.5.5	Code architecture	24

Change History

July 12, 2005	V1.0	Initial document release

1. Document Control

1.1 References

[1]	<i>Series 60 Landmarks API Specification Document</i>	
[2]	<i>Plug-in Architecture 6.2 – ECom Architecture Overview</i>	http://www.symbian.com/

1.2 Documentation Conventions

Code is shown with the `Courier New` font.

1.3 Abbreviations

API	Application Programming Interface
-----	-----------------------------------

1.4 Definitions

DBMS	Database Management System – a commonly used term for the Symbian OS database engine.
ECom	Symbian OS framework for plug-in DLLs.
Landmark	A landmark is a named object that contains a location. The location can be defined by various attributes, e.g. WGS 84 coordinates or a textual address.
Landmark database	Persistent storage of a collection of landmarks and landmark categories.

2. Purpose

The purpose of Landmarks Database Management API is to allow clients to manage landmark databases, for example creating new databases or deleting existing databases. The API is used mainly by end-user applications. To access the databases, Landmarks API (see reference document [1]) is used.

2.1 List of the Required Interfaces

Symbian DBMS API is needed to access local landmark databases.

ECom is used by the implementation to make it extensible.

2.2 Constraints

Landmarks Database Management API requires the Platform Security features in Series 60 Platform release 3.0 and Symbian OS 9.1; it cannot be used in earlier versions.

3. Technical Specification

3.1 Type of the Interface

Landmarks Database Management API is a server client API (implementation encapsulating a session interface).

3.2 Interface Class Structure

The main interface to the API is the `CPosLmDatabaseManager` class. The class `HPosLmDatabaseInfo` holds information about a database and `TPosLmDatabaseSettings` is used to read or change the settings for a database. The `TPosLmDatabaseEvent` class is used when notifications about changes are requested.

3.3 Usage

3.3.1 Protocol

To start managing landmark databases, the client needs to create an instance of the `CPosLmDatabaseManager` class.

Database URIs are used to refer to landmarks databases (see reference document [1]). For example, to create a new landmark database use the following code:

```
_LIT(KDatabaseUri, "file://c:myLandmarks.ldb");

CPosLmDatabaseManager* dbManager =
CPosLmDatabaseManager::NewL();
CleanupStack::PushL(dbManager);

HPosLmDatabaseInfo* dbInfo =
HPosLmDatabaseInfo::NewLC(KDatabaseUri);
dbManager->CreateDatabaseL(*dbInfo);

CleanupStack::PopAndDestroy(2, dbManager);
```

The actual implementation used to create the database depends on the type of the URI. Method calls that have a URI parameter are blocking and can take a long time to complete. This should be taken into consideration if the used protocol is slow to perform the task. For local file access, this will be fast.

Some protocols may not support creating and deleting databases. In that case, the client should use `RegisterDatabaseL()` and `UnregisterDatabaseL()` instead. By registering a database, a link is created so that it will show up in `ListDatabasesL()`. Registering is only done for remote databases. It is not supported for local files.

Listening to events is asynchronous. The client should create an active object that calls `NotifyDatabaseEvent`. Canceling event listening is done by calling `CancelNotifyDatabaseEvent`.

3.3.2 Error handling

Landmarks Database Management API uses the standard Symbian error reporting mechanism. In case of a serious error, panics are used. Otherwise, errors are reported through return codes or leaves.

3.3.3 Memory overhead

Depending on the type of the URI, an ECom plug-in will be loaded for the used protocol. Memory usage will depend on the protocol. For local file access, the memory usage will be low.

3.3.4 Extensions to the API

It is possible to create implementations for new protocols using ECom plug-ins, but the API for writing those extensions is not described in this document.

3.4 Example

3.4.1 Listing databases and getting database information

Below are two examples of how to list landmark databases and get information about them using the two different versions of `ListDatabases`. The first example uses less memory at the cost of performance.

```
CPosLmDatabaseManager* dbManager =
CPosLmDatabaseManager::NewL();
CleanupStack::PushL(dbManager);

// Get a list of databases
CDesCArray* dbList = dbManager->ListDatabasesLC();

for (TInt i = 0; i < dbList->Count(); ++i)
{
    TPtrC dbUri = (*dbList)[i];
    HPosLmDatabaseInfo* dbInfo =
HPosLmDatabaseInfo::NewLC(dbUri);
    dbManager->GetDatabaseInfoL(*dbInfo);

    // Get information about the database
    TPtrC uri = dbInfo->DatabaseUri();
    TPtrC protocol = dbInfo->Protocol();
    TBool defaultDb = dbInfo->IsDefault();
    TChar drive = dbInfo->DatabaseDrive();
    TInt size = dbInfo->Size();

    CleanupStack::PopAndDestroy(dbInfo);
}

// Destroy list
CleanupStack::PopAndDestroy(dbList);

CleanupStack::PopAndDestroy(dbManager);
```

The example below will allocate memory for holding information about all the databases at once.

```
CPosLmDatabaseManager* dbManager =
CPosLmDatabaseManager::NewL();
CleanupStack::PushL(dbManager);

// Get a list of database information
```

```

RPointerArray<HPosLmDatabaseInfo> dbInfoList;
dbManager->ListDatabasesL(dbInfoList);

for (TInt i = 0; i < dbInfoList.Count(); ++i)
{
    HPosLmDatabaseInfo* dbInfo = dbInfoList[i];

    // Get settings
    const TPosLmDatabaseSettings& settings = dbInfo->Settings();
    // Check if a display name is set
    if (settings.IsAttributeSet(TPosLmDatabaseSettings::ENAME))
    {
        TPtrC displayName = settings.DatabaseName();
    }
}

// Destroy list
dbInfoList.ResetAndDestroy();

CleanupStack::PopAndDestroy(dbManager);

```

3.4.2 Moving a database

The example below shows how to move a database between two drives:

```

_LIT(KSourceUri, "file://c:myLandmarks.ldb");
_LIT(KDestUri, "file://e:myLandmarks.ldb");

CPosLmDatabaseManager* dbManager =
CPosLmDatabaseManager::NewL();
CleanupStack::PushL(dbManager);

dbManager->CopyDatabaseL(KSourceUri, KDestUri);
dbManager->DeleteDatabaseL(KSourceUri);

CleanupStack::PopAndDestroy(dbManager);

```

3.4.3 Setting a name for a database

The example below shows how to set a friendly name for a database. Typically, applications will display this name instead of the URI.

```

_LIT(KDatabaseUri, "file://c:myLandmarks.ldb");
_LIT(KDisplayname, "My landmarks");

CPosLmDatabaseManager* dbManager =
CPosLmDatabaseManager::NewL();
CleanupStack::PushL(dbManager);
HPosLmDatabaseInfo* dbInfo =
HPosLmDatabaseInfo::NewLC(KDatabaseUri);

TPosLmDatabaseSettings& settings = dbInfo->Settings();
settings.SetDisplayName(KDisplayname);
dbManager->ModifyDatabaseSettingsL(KDatabaseUri, settings);

CleanupStack::PopAndDestroy(dbInfo);
CleanupStack::PopAndDestroy(dbManager);

```

3.5 Detailed Description

The Landmarks Database Management API classes are documented in detail in the following sections.

3.5.1 CPosLmDatabaseManager

This class is used to manage landmark databases.

`CPosLmDatabaseManager` contains functions for listing, registering, unregistering, creating, deleting, copying landmark databases, etc. It also has functions for managing the default landmark database. The client can listen to events related to database management.

A database is local if it resides in the phone or in some device that is mapped to the phone's file system. If a database is not local, it is remote.

The client refers to a database by URI. The URI consists of a protocol specifier and the database location: *protocol://location*. If the client does not specify a protocol, *file://* is assumed.

For local landmark databases, the URI consists of the drive and the database file name, e.g. *c:landmarks.LDB*. The path cannot be specified by the client. The extension of the database file name must be LDB. If a path is included or if the file extension is not LDB, the client receives the error code `KErrArgument`. For local landmark databases, the client receives the error code `KErrBadName` if the file name is invalid and `KErrNotReady` if the drive specified in the URI does not exist.

If the client specifies a local database URI and does not specify the drive letter, e.g. *landmarks.LDB*, the *c:* drive is assumed.

Local databases are created by calling `CreateDatabaseL`. Bookmarks to remote databases are created by calling `RegisterDatabaseL`. After this, they are listed by the database manager.

If `CPosLmDatabaseManager` is used, the client must call the global function `ReleaseLandmarkResources` before terminating in order to release all the used landmark resources. Otherwise, the client may receive an `ALLOC` panic.

3.5.1.1 Constructor

```
static CPosLmDatabaseManager* NewL()
```

Usage:

A two-phased constructor.

This function requires the `ReadUserData` capability.

3.5.1.2 Managing databases

```
void CopyDatabaseL(const TDesC& aSourceUri, const TDesC&
aTargetUri)
```

Usage:

Copies a landmark database to a new location.

Database locations are specified as URIs. The URI construction is described in the class description for `CPosLmDatabaseManager` (see Section 3.5.1.1). The target URI protocol must be the same as the source URI protocol.

This function requires the `ReadUserData` and `WriteUserData` capabilities. If the databases are remote, the `NetworkServices` capability is also needed.

Parameters:

<code>aSourceUri</code>	The URI of the database to copy.
<code>aTargetUri</code>	The URI of the new database location.

Leave codes:

<code>KErrNotSupported</code>	The protocol specified in the URI is not supported.
<code>KErrArgument</code>	The URI is incorrect.
<code>KErrAlreadyExists</code>	There is already a database at the target URI.
<code>KErrInUse</code>	There is a write-lock on the database, e.g. some client is currently modifying the database.
<code>KErrNotFound</code>	There is no database at the source URI.

```
void CreateDatabaseL(HPosLmDatabaseInfo& aDatabaseInfo)
```

Usage:

Creates a landmark database.

This function requires the `ReadUserData` and `WriteUserData` capabilities. If the database is remote, the `NetworkServices` capability is also needed.

Parameters:

<code>aDatabaseUri</code>	The URI of the new database.
---------------------------	------------------------------

Leave codes:

<code>KErrNotSupported</code>	The protocol specified in the URI is not supported.
<code>KErrArgument</code>	The URI is incorrect.
<code>KErrAlreadyExists</code>	There is already a database at this URI.

```
void DeleteDatabaseL(const TDesC& aDatabaseUri)
```

Usage:

Deletes a landmark database.

The database to be deleted is specified by passing a URI to this function. The URI construction is described in the class description for `CPosLmDatabaseManager` (see Section 3.5.1.1).

If the specified database does not exist, the call is ignored.

This function requires the `ReadUserData` and `WriteUserData` capabilities. If the database is remote, the `NetworkServices` capability is also needed.

Parameters:

aDatabaseUri	The URI of the database to delete.
--------------	------------------------------------

Leave codes:

KErrNotSupported	The protocol specified in the URI is not supported.
KErrArgument	The URI is incorrect.
KErrInUse	The database is in use by some client.
KErrAccessDenied	The database is read-only.

```
void ModifyDatabaseSettingsL(const TDesC& aDatabaseUri, const
TPosLmDatabaseSettings& aDatabaseSettings)
```

Usage:

Modifies the settings for a landmark database.

This function requires the `ReadUserData` and `WriteUserData` capabilities.

Parameters:

aDatabaseUri	The URI of the database for which to modify the settings.
aDatabaseSettings	The new settings for the database.

Leave codes:

KErrNotSupported	The protocol specified in the URI is not supported in the platform.
KErrNotFound	The specified database is not found.
KErrArgument	The URI is incorrect.

```
void SetDefaultDatabaseUriL(const TDesC& aDatabaseUri)
```

Usage:

Sets a landmark database as default.

This database is opened the next time `CPosLandmarkDatabase::OpenL` is called without parameters.

To move the current default database to another drive, first use `CopyDatabaseL` to copy the database to the new drive, then use `SetDefaultDatabaseL` to set the new database as default and finally, use `DeleteDatabaseL` if the old default database should be deleted.

Only "file" protocol databases can be set as default.

This function requires the `WriteDeviceData` capability.

Parameters:

<code>aDatabaseUri</code>	The URI of the database to be set as default.
---------------------------	---

Leave codes:

<code>KErrNotSupported</code>	The protocol specified as the protocol in the URI is something else than <i>file://</i> .
<code>KErrArgument</code>	The URI is incorrect.
<code>KErrNotFound</code>	The landmark database to be set as default does not exist.

3.5.1.3 Registering and unregistering

```
void RegisterDatabaseL(HPosLmDatabaseInfo& aDatabaseInfo)
```

Usage:

Registers a landmark database.

The landmark database is returned when listing landmark databases.

For local landmark databases, this function leaves with the error code `KErrNotSupported`. To add a local database, the client must call `CreateDatabaseL`.

The client supplies an information object containing the URI of the database to register. The information object can also contain database settings, e.g. a display name for the database.

This function requires the `ReadUserData` and `WriteUserData` capabilities.

Parameters:

<code>aDatabaseInfo</code>	Information about the landmark database to register.
----------------------------	--

Leave codes:

<code>KErrNotSupported</code>	The protocol specified in the URI is not supported in the platform or the protocol does not allow registering landmark databases.
<code>KErrArgument</code>	The URI is incorrect.
<code>KErrAlreadyExists</code>	The database already exists in the registry.

```
void UnregisterDatabaseL (const TDesC & aDatabaseUri)
```

Usage:

Unregisters a landmark database.

After this, the landmark database is not returned when listing landmark databases.

For local landmark databases, this function leaves with the error code `KErrNotSupported`. To remove a local database, the client must call `DeleteDatabaseL`.

This function requires the `ReadUserData` and `WriteUserData` capabilities.

Parameters:

<code>aDatabaseUri</code>	The URI of the database to register.
---------------------------	--------------------------------------

Leave codes:

<code>KErrNotSupported</code>	The protocol specified in the URI is not supported in the platform or the protocol does not allow unregistering landmark databases.
<code>KErrArgument</code>	The URI is incorrect.

```
void UnregisterAllDatabasesL(const TDesC& aProtocol)
```

Usage:

Unregisters all the landmark databases that are accessed through a certain protocol.

After this, the landmark databases are not returned when listing landmark databases.

For local landmark databases, this function leaves with the error code `KErrNotSupported`. To remove a local database, the client must call `DeleteDatabaseL`.

This function requires the `ReadUserData` and `WriteUserData` capabilities.

Parameters:

<code>aProtocol</code>	The protocol for which to unregister all databases.
------------------------	---

Leave codes:

<code>KErrNotSupported</code>	The protocol is not supported in the platform or the protocol does not allow unregistering landmark databases.
-------------------------------	--

3.5.1.4 Events

```
void NotifyDatabaseEvent(TPosLmDatabaseEvent& aEvent,  
TRequestStatus& aStatus)
```

Usage:

Listens to database management events.

This function is asynchronous and it completes the request status when an event occurs. At this time, the client can read event information from the retrieved event object.

Event listening can be canceled by calling `CancelNotifyDatabaseEvent`.

This function requires the `ReadUserData` capability.

Parameters:

<code>aEvent</code>	Contains the event information when an event occurs.
<code>aStatus</code>	Is completed with <code>KErrNone</code> if an event occurs or with an error code if some error is encountered.

```
TInt CancelNotifyDatabaseEvent()
```

Usage:

Cancels a call to `NotifyDatabaseEvent`.

Returns:

`KErrNone` if the request was successfully cancelled, otherwise a system wide error code.

```
HBufC* DatabaseUriFromLastEventLC()
```

Usage:

Retrieves the database URI associated with the last event.

Some events, e.g. `EPosLmDbDatabaseRegistered` and `EPosLmDbDatabaseUnregistered`, refer to a specific database. The URI of the database can be retrieved through this function.

If the URI should be retrieved, this function must be called before `NotifyDatabaseEvent` is called again to listen to the next event.

This function requires the `ReadUserData` capability.

Returns:

The database URI associated with the event. The client takes ownership of the descriptor object.

Leave codes:

<code>KErrNotFound</code>	There is no database URI associated with the event or the database URI has been discarded because the client has called <code>NotifyDatabaseEvent</code> again.
---------------------------	---

3.5.1.5 Database information

```
TBool DatabaseExistsL(const TDesC& aDatabaseUri)
```

Usage:

Checks if the specified landmark database exists.

The database to be checked is specified by passing a URI to this function. The URI construction is described in the class description for `CPosLmDatabaseManager` in Section 3.5.1.1. The URI must point to a database, which is handled by this database manager, i.e. not a remote location.

This function requires the `ReadUserData` capability. If the database is remote, the `NetworkServices` capability is also needed.

Parameters:

<code>aDatabaseUri</code>	The URI of the database, which should be checked for existence.
---------------------------	---

Returns:

`ETrue` if the database exists, otherwise `EFalse`.

Leave codes:

<code>KErrNotSupported</code>	The protocol specified in the URI is not supported.
<code>KErrArgument</code>	The URI is incorrect.

`HBufC* DefaultDatabaseUriLC()`

Usage:

Retrieves the URI of the default landmark database.

The default landmark database is the one opened when calling `CPosLandmarkDatabase::OpenL` without any parameters.

Only "file"-protocol databases can be set as default.

This function requires the `ReadUserData` capability.

Returns:

The URI of the default landmark database. The client takes ownership of the descriptor object.

`void GetDatabaseInfoL(HPosLmDatabaseInfo& aDatabaseInfo)`

Usage:

Retrieves information about a landmark database.

This function requires the `ReadUserData` capability.

Parameters:

<code>aDatabaseInfo</code>	An information object containing the URI of the landmark database. On return, the object contains information about the landmark database, including any database settings.
----------------------------	---

Leave codes:

KErrNotSupported	The protocol specified in the URI is not supported in the platform.
KErrNotFound	The specified database is not found.
KErrArgument	The URI is incorrect.

```
void ListDatabasesL(RPointerArray<HPosLmDatabaseInfo>&
aDatabaseInfoArray,
const TDesC& aProtocol = KNullDesC)
```

Usage:

Lists information about each landmark database known by the Landmarks system.

The client can specify a protocol string as input to this function. The function then only returns information about the databases, which are accessed through this protocol.

The client specifies an array, which is populated by this function. The client takes ownership of all the information objects in the array.

If no databases are found, an empty array is returned.

This function requires the `ReadUserData` capability.

Parameters:

aDatabaseInfoArray	On return, contains information about the landmark databases. Any objects that are in the array when it is passed to this function are not removed.
aProtocol	The protocol for which database URIs should be listed. If no protocol is specified, i.e. if an empty string is specified, all known database URIs are listed.

Leave codes:

KErrNotSupported	The protocol is not supported in the platform.
------------------	--

```
CDesCArray* ListDatabasesLC (const TDesC & aProtocol =
KNullDesC)
```

Usage:

Lists the URIs to all the landmark databases known by the Landmarks system.

The client can specify a protocol string as input to this function. The function then only returns a list of the databases, which are accessed through this protocol.

The client takes ownership of the returned array.

If no databases are found, an empty array is returned.

This function requires the `ReadUserData` capability.

Parameters:

aProtocol	The protocol for which database URIs should be listed. If no protocol is specified, i.e. if an empty string is specified, all known database URIs are listed.
-----------	---

Returns:

A list of the database URIs.

Leave codes:

KErrNotSupported	The protocol is not supported in the platform.
------------------	--

3.5.2 HPosLmDatabaseInfo

`HPosLmDatabaseInfo` encapsulates information about a landmark database.

3.5.2.1 Constructor

```
static HPosLmDatabaseInfo* NewL(const HPosLmDatabaseInfo&
aDatabaseInfo)
```

Usage:

A two-phased copy constructor.

Parameters:

aDatabaseInfo	The information instance to copy.
---------------	-----------------------------------

Returns:

A new instance of this class.

```
static HPosLmDatabaseInfo* NewL(const TDesC& aDatabaseUri)
```

Usage:

A two-phased constructor.

Parameters:

aDatabaseUri	The URI of the landmark database.
--------------	-----------------------------------

Returns:

A new instance of this class.

```
static HPosLmDatabaseInfo* NewLC(const HPosLmDatabaseInfo&
aDatabaseInfo)
```

Usage:

A two-phased copy constructor.

Parameters:

aDatabaseInfo	The information instance to copy.
---------------	-----------------------------------

Returns:

A new instance of this class.

```
static HPosLmDatabaseInfo* NewLC(const TDesC& aDatabaseUri)
```

Usage:

A two-phased constructor.

Parameters:

aDatabaseUri	The URI of the landmark database.
--------------	-----------------------------------

Returns:

A new instance of this class.

3.5.2.2 Member functions

```
TChar DatabaseDrive() const
```

Usage:

Returns which database drive the database resides in.

Note that this attribute is only set if the `HPosLmDatabaseInfo` instance has been returned from a `CPosLmDatabaseManager` function, e.g.

```
CPosLmDatabaseManager::ListDatabasesL,  
CPosLmDatabaseManager::GetDatabaseInfoL,  
CPosLmDatabaseManager::RegisterDatabaseL,  
CPosLmDatabaseManager::CreateDatabaseL Or  
CPosLmDatabaseManager::ModifyDatabaseSettingsL. If not, this function  
returns 0.
```

If database drive is not applicable for the database, e.g. the database is remote, this function returns 0.

Returns:

The drive letter for the drive where the database resides, or 0 if the letter is not set.

```
TMediaType DatabaseMedia() const
```

Usage:

Returns which storage media the database resides in.

Note that this attribute is only set if the `HPosLmDatabaseInfo` instance has been returned from a `CPosLmDatabaseManager` function, e.g.

```
CPosLmDatabaseManager::ListDatabasesL,  
CPosLmDatabaseManager::GetDatabaseInfoL,  
CPosLmDatabaseManager::RegisterDatabaseL,
```

`CPosLmDatabaseManager::CreateDatabaseL OR
CPosLmDatabaseManager::ModifyDatabaseSettingsL`. If not, this function returns `EMediaUnknown`.

Returns:

The storage media the database resides in.

`TPtrC DatabaseUri() const`

Usage:

Retrieves the database URI.

Returns:

A pointer to the URI descriptor. This pointer is valid until the `HPosLmDatabaseInfo` object is destroyed.

`TBool IsDefault() const`

Usage:

Returns whether the database is the default database or not.

Note that this attribute is only set if the `HPosLmDatabaseInfo` instance has been returned from a `CPosLmDatabaseManager` function, e.g.

`CPosLmDatabaseManager::ListDatabasesL,
CPosLmDatabaseManager::GetDatabaseInfoL,
CPosLmDatabaseManager::RegisterDatabaseL,
CPosLmDatabaseManager::CreateDatabaseL OR
CPosLmDatabaseManager::ModifyDatabaseSettingsL`. If not, this function returns `EFalse`.

Returns:

`ETrue` if the database is the default one, otherwise `EFalse`.

`TPtrC Protocol() const`

Usage:

Returns the protocol part from the set URI.

For example, if the URI is `file://c:/landmarks.LDB`, then the protocol is "file".

If no protocol is specified, an empty descriptor is returned. This implies "file" protocol.

Returns:

A pointer to the protocol descriptor. This pointer is valid until the `HPosLmDatabaseInfo` object is destroyed.

`TPosLmDatabaseSettings& Settings()`

Usage:

Retrieves a reference to the database settings.

The reference can be used to read and write to the database settings.

Returns:

Reference to the database settings

```
const TPosLmDatabaseSettings& Settings() const
```

Usage:

Retrieve a const reference to the database settings.

The const reference can be used to read the database settings.

Returns:

A constant reference to the database settings.

```
TInt Size() const
```

Usage:

Returns the size of this object in bytes.

Returns:

The size of this object.

3.5.3 TPosLmDatabaseEventType

A struct for landmark database events.

```
TPosLmDatabaseEventType iEventType
```

Usage:

The type of the event.

Enumeration values:

Value	Description
EPosLmDbUnknownEvent	Something has happened, but what happened is not specified. This event is used if there are too many events in which case the events are bundled into a single unknown event. If the client is interested in some database management information, it should be reread from the database manager.
EPosLmDbDatabaseRegistered	A landmark database has been registered or created. This event is also generated if a database is copied. The URI of the new database can be retrieved by calling <code>CPosLmDatabaseManager::DatabaseUriFromEventLC</code> .
EPosLmDbDatabaseUnregistered	A landmark database has been unregistered or deleted. The URI of the deleted database can be retrieved by calling <code>CPosLmDatabaseManager::DatabaseUriFromEventLC</code> .

EPosLmDbSettingsModified	Information about a database, e.g. the display name, has been changed. The URI of the database can be retrieved by calling <code>CPosLmDatabaseManager::DatabaseUriFromEventLC</code> .
EPosLmDbMediaRemoved	Media was removed, possibly containing landmark databases. Use <code>ListDatabasesL</code> to list the available databases.
EPosLmDbMediaInserted	Media was inserted, possibly containing landmark databases. Use <code>ListDatabasesL</code> to list the available databases.
EPosLmDbNewDefaultDbLocation	The location of the default database has changed. The URI of the default database can be retrieved by calling <code>CPosLmDatabaseManager::DefaultDatabaseUriLC</code> .

3.5.4 TPosLmDatabaseSettings

`TPosLmDatabaseSettings` encapsulates the attributes that can be set for a landmark database.

The only available attribute is the displayable name.

To set a new display name, create an `HPosLmDatabaseInfo` object containing the URI of the database. Call `SetDatabaseName` on the `TPosLmDatabaseSettings` member in `HPosLmDatabaseInfo` and then pass `HPosLmDatabaseInfo` to `CPosLmDatabaseManager::ModifyDatabaseSettingsL`.

When retrieving settings for a database, `IsAttributeSet` can be used to find out whether a display name is set for the database or not.

```
enum TPosLmDatabaseSettings::TAttribute
```

Enumeration of the attributes that can be set for a landmark database.

Enumeration values:

EName	The displayable name for the landmark database.
-------	---

3.5.4.1 Constructor

```
TPosLmDatabaseSettings()
```

Usage:

The default constructor.

Returns:

A new instance of this class.

3.5.4.2 Member functions

```
TPtrC DatabaseName() const
```

Usage:

Retrieves the displayable name for the database.

If the `EName` attribute is not set, an empty descriptor is returned.

Returns:

A pointer to the name descriptor. This pointer is valid until the `TPosLmDatabaseSettings` instance is destroyed.

```
TBool IsAttributeSet(TAttribute aDbAttribute) const
```

Usage:

Checks if a database attribute is set in this instance.

Parameters:

<code>aDbAttribute</code>	The database attribute to check.
---------------------------	----------------------------------

Returns:

`ETrue` if the attribute is set, otherwise `EFalse`.

```
void SetDatabaseName(const TPosLmDatabaseName& aDatabaseName)
```

Usage:

Sets a displayable name for the database.

If an empty descriptor is set, the database display name will be set to an empty string.

Parameters:

<code>aDatabaseName</code>	The new name for the database.
----------------------------	--------------------------------

```
void UnsetAttribute(TAttribute aDbAttribute)
```

Usage:

Unsets a database attribute.

If an attribute is not set in this instance and the instance is passed as input to a function for modifying the settings for a database, e.g.

`CPosLmDatabaseManager::ModifyDatabaseSettingsL`, the attribute will be removed from the database.

Parameters:

<code>aDbAttribute</code>	The database attribute to unset.
---------------------------	----------------------------------

3.5.5 Code architecture

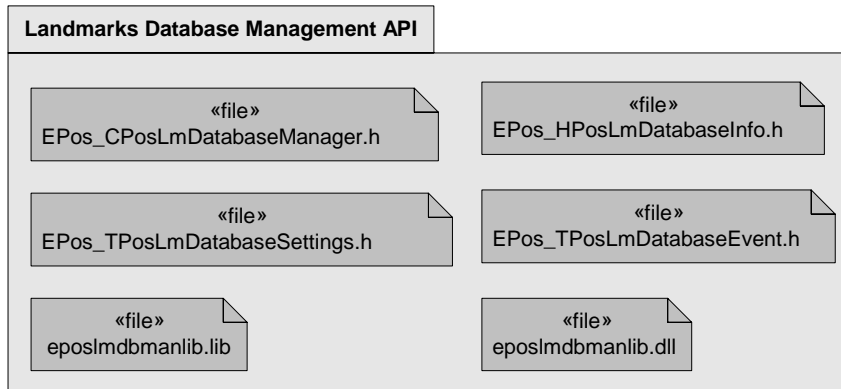


Figure 3.1: Files needed by the Landmarks Database Management API client

The Landmarks Database Management API classes are defined in the header files shown in Figure 3.1. *eposlmbmanlib.LIB* contains link-time information for the API and *eposlmbmanlib.DLL* contains the run-time information.