



# UltraLite® C and C++ Programming

Copyright © 2010 iAnywhere Solutions, Inc. Portions copyright © 2010 Sybase, Inc. All rights reserved.

This documentation is provided AS IS, without warranty or liability of any kind (unless provided by a separate written agreement between you and iAnywhere).

You may use, print, reproduce, and distribute this documentation (in whole or in part) subject to the following conditions: 1) you must retain this and all other proprietary notices, on all copies of the documentation or portions thereof, 2) you may not modify the documentation, 3) you may not do anything to indicate that you or anyone other than iAnywhere is the author or source of the documentation.

iAnywhere®, Sybase®, and the marks listed at <http://www.sybase.com/detail?id=1011207> are trademarks of Sybase, Inc. or its subsidiaries. ® indicates registration in the United States of America.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

---

---

# Contents

<b>About this book .....</b>	<b>xi</b>
<b>About the SQL Anywhere documentation .....</b>	<b>xi</b>
<b>UltraLite for C/C++ developers .....</b>	<b>1</b>
<b>Developing embedded SQL applications .....</b>	<b>1</b>
<b>System requirements and supported platforms .....</b>	<b>2</b>
<b>UltraLite C/C++ API architecture .....</b>	<b>2</b>
<b>Creating databases .....</b>	<b>3</b>
<b>Application development .....</b>	<b>5</b>
<b>Developing applications using the UltraLite C++ API .....</b>	<b>7</b>
Quick start guide to UltraLite C++ application development .....	7
iPhone and Mac OS X considerations .....	7
Connecting to a database .....	8
Closing an UltraLite database connection .....	10
Accessing data using SQL statements .....	10
Accessing data using ULTable class .....	17
Managing transactions .....	22
Accessing schema information .....	23
Handling errors .....	24
Authenticating users .....	24
Encrypting data .....	25
Synchronizing data .....	26
Compiling and linking your application .....	26
<b>Developing applications using embedded SQL .....</b>	<b>29</b>
Example of embedded SQL .....	29
Initializing the SQL Communications Area .....	31
Connecting to a database .....	32
Using host variables .....	33
Fetching data .....	43
Authenticating users .....	47

Encrypting data .....	48
Adding synchronization to your application .....	49
Building embedded SQL applications .....	56
<b>Developing UltraLite applications for Windows Mobile .....</b>	<b>59</b>
Choosing how to link the runtime library .....	59
Building the CustDB sample application .....	59
Storing persistent data .....	60
Deploying Windows Mobile applications .....	61
Deploying applications that use ActiveSync .....	61
Assigning class names for applications .....	61
Synchronization on Windows Mobile .....	63
<b>API reference .....</b>	<b>67</b>
<b>UltraLite C/C++ common API reference .....</b>	<b>69</b>
Macros and compiler directives for UltraLite C/C++ applications .....	69
UL_RS_STATE enumeration .....	70
ul_column_sql_type enumeration .....	70
ul_column_storage_type enumeration .....	72
ul_database_option_id enumeration .....	73
ul_database_property_id enumeration .....	74
ul_error_action enumeration .....	76
ul_sync_state enumeration .....	76
ul_validate_status_id enumeration .....	78
ul_binary structure .....	80
ul_error_info structure .....	80
ul_stream_error structure .....	81
ul_sync_info structure .....	81
ul_sync_info_w2 structure .....	83
ul_sync_result structure .....	86
ul_sync_stats structure .....	87
ul_sync_status structure .....	87
ul_validate_data structure .....	88
ULVF_DATABASE variable .....	89
ULVF_EXPRESS variable .....	89
ULVF_FULL_VALIDATE variable .....	89

ULVF_IDX_HASH variable .....	90
ULVF_IDX_REDUNDANT variable .....	90
ULVF_INDEX variable .....	90
ULVF_TABLE variable .....	90
UL_AS_SYNCHRONIZE variable .....	90
UL_SYNC_ALL variable .....	91
UL_SYNC_ALL_PUBS variable .....	91
UL_SYNC_STATUS_FLAG_IS_BLOCKING variable .....	91
UL_VALID_IS_ERROR variable .....	91
UL_VALID_IS_INFO variable .....	92
<b>UltraLite C/C++ API reference .....</b>	<b>93</b>
ULConnection class .....	93
ULDatabaseManager class .....	139
ULDatabaseSchema class .....	151
ULError class .....	155
ULIndexSchema class .....	161
ULPreparedStatement class .....	167
ULResultSet class .....	179
ULResultSetSchema class .....	230
ULTable class .....	235
ULTableSchema class .....	243
ul_column_default_type enumeration .....	254
ul_column_name_type enumeration .....	255
ul_index_flag enumeration .....	255
ul_table_sync_type enumeration .....	256
UL_BLOB_CONTINUE variable .....	256
ul_index_iter_start variable .....	257
ul_publication_iter_start variable .....	257
ul_table_iter_start variable .....	257
<b>UltraLite Embedded SQL API reference .....</b>	<b>259</b>
db_fini method .....	259
db_init method .....	259
MLFTEnableEccE2ee method .....	260
MLFTEnableEccE2eeW method .....	260
MLFTEnableEccEncryption method .....	261

MLFTEnableEccEncryptionW method .....	261
MLFTEnableRsaE2ee method .....	262
MLFTEnableRsaE2eeW method .....	262
MLFTEnableRsaEncryption method .....	262
MLFTEnableRsaEncryptionW method .....	263
MLFTEnableRsaFipsE2ee method .....	263
MLFTEnableRsaFipsE2eeW method .....	264
MLFTEnableRsaFipsEncryption method .....	264
MLFTEnableRsaFipsEncryptionW method .....	264
MLFTEnableZlibCompression method .....	265
MLFTEnableZlibCompressionW method .....	265
MLFileDownload method .....	266
MLFileDownloadW method .....	266
MLFileUpload method .....	267
MLFileUploadW method .....	267
MLFiniFileTransferInfo method .....	268
MLFiniFileTransferInfoW method .....	268
MLInitFileTransferInfo method .....	269
MLInitFileTransferInfoW method .....	269
ULCancelGetNotification method .....	270
ULCancelGetNotificationW method .....	270
ULChangeEncryptionKey method .....	271
ULChangeEncryptionKeyW method .....	271
ULCheckpoint method .....	272
ULCountUploadRows method .....	272
ULCountUploadRowsW method .....	273
ULCreateDatabase method .....	274
ULCreateDatabaseW method .....	275
ULCreateNotificationQueue method .....	276
ULCreateNotificationQueueW method .....	277
ULDeclareEvent method .....	278
ULDeclareEventW method .....	278
ULDeleteAllRows method .....	279
ULDestroyNotificationQueue method .....	279
ULDestroyNotificationQueueW method .....	280

---

ULECCLibraryVersion method .....	280
ULEnableAesDBEncryption method .....	281
ULEnableAesFipsDBEncryption method .....	281
ULEnableEccE2ee method .....	282
ULEnableEccSyncEncryption method .....	282
ULEnableHttpSynchronization method .....	283
ULEnableRsaE2ee method .....	283
ULEnableRsaFipsE2ee method .....	283
ULEnableRsaFipsSyncEncryption method .....	283
ULEnableRsaSyncEncryption method .....	284
ULEnableTcpipSynchronization method .....	284
ULEnableZlibSyncCompression method .....	285
ULErrorInfoInitFromSqlca method .....	285
ULErrorInfoParameterAt method .....	286
ULErrorInfoParameterCount method .....	286
ULErrorInfoString method .....	286
ULErrorInfoURL method .....	287
ULGetDatabaseID method .....	287
ULGetDatabaseProperty method .....	288
ULGetDatabasePropertyW method .....	288
ULGetErrorParameter method .....	289
ULGetErrorParameterCount method .....	289
ULGetErrorParameterW method .....	290
ULGetIdentity method .....	291
ULGetLastDownloadTime method .....	291
ULGetLastDownloadTimeW method .....	292
ULGetNotification method .....	292
ULGetNotificationParameter method .....	293
ULGetNotificationParameterW method .....	294
ULGetNotificationW method .....	295
ULGetSyncResult method .....	295
ULGlobalAutoincUsage method .....	296
ULGrantConnectTo method .....	296
ULGrantConnectToW method .....	297
ULInitSyncInfo method .....	298

ULInitSyncInfoW method .....	298
ULIsSynchronizeMessage method .....	298
ULLibraryVersion method .....	299
ULRSALibraryVersion method .....	299
ULRegisterForEvent method .....	300
ULRegisterForEventW method .....	300
ULResetLastDownloadTime method .....	302
ULResetLastDownloadTimeW method .....	302
ULRevokeConnectFrom method .....	303
ULRevokeConnectFromW method .....	303
ULRollbackPartialDownload method .....	304
ULSendNotification method .....	304
ULSendNotificationW method .....	305
ULSetDatabaseID method .....	306
ULSetDatabaseOptionString method .....	306
ULSetDatabaseOptionStringW method .....	306
ULSetDatabaseOptionULong method .....	307
ULSetErrorCallback method .....	307
ULSetErrorCallbackW method .....	308
ULSetSyncInfo method .....	308
ULSetSyncInfoW method .....	309
ULSetSynchronizationCallback method .....	310
ULSetSynchronizationCallbackEx method .....	310
ULSignalSynclsComplete method .....	311
ULStartSynchronizationDelete method .....	311
ULStaticFini method .....	311
ULStaticInit method .....	311
ULStopSynchronizationDelete method .....	312
ULSynchronize method .....	312
ULSynchronizeFromProfile method .....	312
ULSynchronizeFromProfileW method .....	313
ULSynchronizeW method .....	314
ULTriggerEvent method .....	314
ULTriggerEventW method .....	315
ULTruncateTable method .....	315



---

ULValidateDatabase method .....	316
ULValidateDatabaseTableName method .....	317
ULValidateDatabaseTableNameW method .....	318
ULValidateDatabaseW method .....	319
ml_file_transfer_info structure .....	320
ml_file_transfer_info_w structure .....	322
ml_file_transfer_status structure .....	325
mlft_stream_error structure .....	326
mlft_stream_error_w structure .....	326
MLFT_STATUS_FLAG_IS_BLOCKING variable .....	327
<b>Tutorial: Build an application using the C++ API .....</b>	<b>329</b>
Lesson 1: Create database and connect to database .....	329
Lesson 2: Insert data into the database .....	333
Lesson 3: Select and list rows from the table .....	334
Lesson 4: Add synchronization to your application .....	335
Code listing for tutorial .....	336
<b>Tutorial: Build an iPhone application using the C++ API .....</b>	<b>339</b>
Compiling the UltraLite iPhone library .....	339
Creating an UltraLite application on iPhone .....	340
Lesson 1: Creating a new iPhone application project .....	340
Lesson 2: Adding a database to the application .....	342
Lesson 3: Adding data to the database .....	345
Lesson 4: Displaying data from the database .....	349
Lesson 5: Deleting data from the database .....	351
Lesson 6: Adding synchronization .....	352
Lesson 7: Adding a progress display .....	357
Conclusion .....	363
<b>Index .....</b>	<b>365</b>

---

---

# About this book

This book describes UltraLite C and C++ programming interfaces. With UltraLite, you can develop and deploy database applications to handheld, mobile, or embedded devices.

## About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in four formats:

- **DocCommentXchange** DocCommentXchange is a community for accessing and discussing SQL Anywhere documentation on the web.

To access the documentation, go to <http://dcx.sybase.com>.

- **HTML Help** On Windows platforms, the HTML Help contains the complete SQL Anywhere documentation, including the books and the context-sensitive help for SQL Anywhere tools.

To access the documentation, choose **Start » Programs » SQL Anywhere 12 » Documentation » HTML Help (English)**.

- **Eclipse** On Unix platforms, the complete Help is provided in Eclipse format. To access the documentation, run *sadoc* from the *bin32* or *bin64* directory of your SQL Anywhere installation.

- **PDF** The complete set of SQL Anywhere books is provided as a set of Portable Document Format (PDF) files. You must have a PDF reader to view information.

To access the PDF documentation on Windows operating systems, choose **Start » Programs » SQL Anywhere 12 » Documentation » PDF (English)**.

To access the PDF documentation on Unix operating systems, use a web browser to open */documentation/en/pdf/index.html* under the SQL Anywhere installation directory.

## Documentation conventions

This section lists the conventions used in this documentation.

### Operating systems

SQL Anywhere runs on a variety of platforms. Typically, the behavior of the software is the same on all platforms, but there are variations or limitations. These are commonly based on the underlying operating system (Windows, Unix), and seldom on the particular variant (IBM AIX, Windows Mobile) or version.

To simplify references to operating systems, the documentation groups the supported operating systems as follows:

- **Windows** The Microsoft Windows family includes platforms that are used primarily on server, desktop, and laptop computers, as well as platforms used on mobile devices. Unless otherwise specified, when the documentation refers to Windows, it refers to all supported Windows-based platforms, including Windows Mobile.

Windows Mobile is based on the Windows CE operating system, which is also used to build a variety of platforms other than Windows Mobile. Unless otherwise specified, when the documentation refers to Windows Mobile, it refers to all supported platforms built using Windows CE.

- **Unix** Unless otherwise specified, when the documentation refers to Unix, it refers to all supported Unix-based platforms, including Linux and Mac OS X.

For the complete list of platforms supported by SQL Anywhere, see [“Supported platforms” \[SQL Anywhere 12 - Introduction\]](#).

## Directory and file names

Usually references to directory and file names are similar on all supported platforms, with simple transformations between the various forms. In these cases, Windows conventions are used. Where the details are more complex, the documentation shows all relevant forms.

These are the conventions used to simplify the documentation of directory and file names:

- **Uppercase and lowercase directory names** On Windows and Unix, directory and file names may contain uppercase and lowercase letters. When directories and files are created, the file system preserves letter case.

On Windows, references to directories and files are *not* case sensitive. Mixed case directory and file names are common, but it is common to refer to them using all lowercase letters. The SQL Anywhere installation contains directories such as *Bin32* and *Documentation*.

On Unix, references to directories and files *are* case sensitive. Mixed case directory and file names are not common. Most use all lowercase letters. The SQL Anywhere installation contains directories such as *bin32* and *documentation*.

The documentation uses the Windows forms of directory names. You can usually convert a mixed case directory name to lowercase for the equivalent directory name on Unix.

- **Slashes separating directory and file names** The documentation uses backslashes as the directory separator. For example, the PDF form of the documentation is found in *install-dir\Documentation\en\PDF* (Windows form).

On Unix, replace the backslash with the forward slash. The PDF documentation is found in *install-dir/documentation/en/pdf*.

- **Executable files** The documentation shows executable file names using Windows conventions, with a suffix such as *.exe* or *.bat*. On Unix, executable file names have no suffix.

For example, on Windows, the network database server is *dsrv12.exe*. On Unix, it is *dsrv12*.

- **install-dir** During the installation process, you choose where to install SQL Anywhere. The environment variable `SQLANY12` is created and refers to this location. The documentation refers to this location as *install-dir*.

For example, the documentation may refer to the file *install-dir/readme.txt*. On Windows, this is equivalent to `%SQLANY12%\readme.txt`. On Unix, this is equivalent to `$(SQLANY12)/readme.txt` or `$(SQLANY12)/readme.txt`.

For more information about the default location of *install-dir*, see [“SQLANY12 environment variable” \[SQL Anywhere Server - Database Administration\]](#).

- **samples-dir** During the installation process, you choose where to install the samples included with SQL Anywhere. The environment variable `SQLANYSAMP12` is created and refers to this location. The documentation refers to this location as *samples-dir*.

To open a Windows Explorer window in *samples-dir*, choose **Start » Programs » SQL Anywhere 12 » Sample Applications And Projects**.

For more information about the default location of *samples-dir*, see [“SQLANYSAMP12 environment variable” \[SQL Anywhere Server - Database Administration\]](#).

## Command prompts and command shell syntax

Most operating systems provide one or more methods of entering commands and parameters using a command shell or command prompt. Windows command prompts include Command Prompt (DOS prompt) and 4NT. Unix command shells include Korn shell and bash. Each shell has features that extend its capabilities beyond simple commands. These features are driven by special characters. The special characters and features vary from one shell to another. Incorrect use of these special characters often results in syntax errors or unexpected behavior.

The documentation provides command line examples in a generic form. If these examples contain characters that the shell considers special, the command may require modification for the specific shell. The modifications are beyond the scope of this documentation, but generally, use quotes around the parameters containing those characters or use an escape character before the special characters.

These are some examples of command line syntax that may vary between platforms:

- **Parentheses and curly braces** Some command line options require a parameter that accepts detailed value specifications in a list. The list is usually enclosed with parentheses or curly braces. The documentation uses parentheses. For example:

```
-x tcpip(host=127.0.0.1)
```

Where parentheses cause syntax problems, substitute curly braces:

```
-x tcpip{host=127.0.0.1}
```

If both forms result in syntax problems, the entire parameter should be enclosed in quotes as required by the shell:

```
-x "tcpip(host=127.0.0.1)"
```

- **Semicolons** On Unix, semicolons should be enclosed in quotes.
- **Quotes** If you must specify quotes in a parameter value, the quotes may conflict with the traditional use of quotes to enclose the parameter. For example, to specify an encryption key whose value contains double-quotes, you might have to enclose the key in quotes and then escape the embedded quote:

```
-ek "my \"secret\" key"
```

In many shells, the value of the key would be my "secret" key.

- **Environment variables** The documentation refers to setting environment variables. In Windows shells, environment variables are specified using the syntax `%ENVVVAR%`. In Unix shells, environment variables are specified using the syntax `$ENVVVAR` or `${ENVVVAR}`.

## Contacting the documentation team

We would like to receive your opinions, suggestions, and feedback on this Help.

You can leave comments directly on help topics using DocCommentXchange. DocCommentXchange (DCX) is a community for accessing and discussing SQL Anywhere documentation. Use DocCommentXchange to:

- View documentation
- Check for clarifications users have made to sections of documentation
- Provide suggestions and corrections to improve documentation for all users in future releases

Go to <http://dcx.sybase.com>.

## Finding out more and requesting technical support

### Newsgroups

If you have questions or need help, you can post messages to the Sybase iAnywhere newsgroups listed below.

When you write to one of these newsgroups, always provide details about your problem, including the build number of your version of SQL Anywhere. You can find this information by running the following command: **dbeng12 -v**.

The newsgroups are located on the *forums.sybase.com* news server.

The newsgroups include the following:

- [sybase.public.sqlanywhere.general](#)
- [sybase.public.sqlanywhere.linux](#)
- [sybase.public.sqlanywhere.mobilink](#)
- [sybase.public.sqlanywhere.product\\_futures\\_discussion](#)
- [sybase.public.sqlanywhere.replication](#)
- [sybase.public.sqlanywhere.ultralite](#)
- [ianywhere.public.sqlanywhere.qanywhere](#)

For web development issues, see <http://groups.google.com/group/sql-anywhere-web-development>.

#### **Newsgroup disclaimer**

iAnywhere Solutions has no obligation to provide solutions, information, or ideas on its newsgroups, nor is iAnywhere Solutions obliged to provide anything other than a systems operator to monitor the service and ensure its operation and availability.

iAnywhere Technical Advisors, and other staff, assist on the newsgroup service when they have time. They offer their help on a volunteer basis and may not be available regularly to provide solutions and information. Their ability to help is based on their workload.

## **Developer Centers**

The **SQL Anywhere Tech Corner** gives developers easy access to product technical documentation. You can browse technical white papers, FAQs, tech notes, downloads, techcasts and more to find answers to your questions as well as solutions to many common issues. See <http://www.sybase.com/developer/library/sql-anywhere-techcorner>.

The following table contains a list of the developer centers available for use on the SQL Anywhere Tech Corner:

<b>Name</b>	<b>URL</b>	<b>Description</b>
<b>SQL Anywhere .NET Developer Center</b>	<a href="http://www.sybase.com/developer/library/sql-anywhere-techcorner/microsoft-net">www.sybase.com/developer/library/sql-anywhere-techcorner/microsoft-net</a>	Get started and get answers to specific questions regarding SQL Anywhere and .NET development.
<b>PHP Developer Center</b>	<a href="http://www.sybase.com/developer/library/sql-anywhere-techcorner/php">www.sybase.com/developer/library/sql-anywhere-techcorner/php</a>	An introduction to using the PHP (PHP Hypertext Preprocessor) scripting language to query your SQL Anywhere database.

Name	URL	Description
<b>SQL Anywhere Windows Mobile Developer Center</b>	<a href="http://www.sybase.com/developer/library/sql-anywhere-techcorner/windows-mobile">www.sybase.com/developer/library/sql-anywhere-techcorner/windows-mobile</a>	Get started and get answers to specific questions regarding SQL Anywhere and Windows Mobile development.



---

# UltraLite for C/C++ developers

The C/C++ interfaces provide the following benefits for UltraLite developers:

- A small, high-performance database store with native synchronization.
- The power, efficiency, and flexibility of the C or C++ language.
- The ability to deploy applications on Windows Mobile, Windows desktop platforms, Linux desktop, embedded Linux, and iPhone.

All UltraLite C/C++ interfaces utilize the same UltraLite run time engine. The APIs each provide access to the same underlying functionality.

For more information about the features of UltraLite databases, see [“Creating and configuring UltraLite databases” \[UltraLite - Database Management and Reference\]](#).

## Developing embedded SQL applications

When developing embedded SQL applications, you mix SQL statements with standard C or C++ source code. To develop embedded SQL applications you should be familiar with the C or C++ programming language.

The development process for embedded SQL applications is as follows:

1. Design your UltraLite database.
2. Write your source code in an embedded SQL source file, which typically has extension *.sql*.

When you need data access in your source code, use the SQL statement you want to execute, prefixed by the EXEC SQL keywords. For example:

```
EXEC SQL BEGIN DECLARE SECTION
    int cost
    char pname[31];
EXEC SQL END DECLARE SECTION

EXEC SQL SELECT price, prod_name
    INTO :cost, :pname
    FROM ULProduct
    WHERE prod_id= :pid;
```

3. Preprocess the *.sql* files.

SQL Anywhere includes a SQL preprocessor (sqlpp), which reads the *.sql* files and generates *.cpp* files. These files hold function calls to the UltraLite runtime library.

4. Compile your *.cpp* files.

5. Link the *.cpp* files.

You must link the files with the UltraLite runtime library.

For more information about embedded SQL development, see [“Building embedded SQL applications” on page 56](#).

## System requirements and supported platforms

### Development platforms

To develop applications using UltraLite C++, you require the following:

- A Microsoft Windows, Linux, or Mac desktop as a development platform.
- A supported Microsoft or GNU C/C++ compiler.

### Target platforms

UltraLite C/C++ supports the following target platforms:

- Windows Mobile 5.0 or later
- Windows XP or later
- Linux
- Embedded Linux
- iPhone 2.0 or later
- Mac

## UltraLite C/C++ API architecture

The UltraLite C/C++ API architecture is defined in the *ulcpp.h* header file. The following list describes some of the commonly used objects:

- **ULDatabaseManager** Provides methods for managing database connections, such as `CreateDatabase` and `OpenConnection`.
- **ULConnection** Represents a connection to an UltraLite database. You can create one or more `ULConnection` objects.
- **ULTable** Provides direct access to tables in the database.

- **ULPreparedStatement, ULResultSet, and ULResultSetSchema** Create Dynamic SQL statements, make queries and execute INSERT, UPDATE, and DELETE statements, and attain programmatic control over database result sets.

For more information about accessing the API reference, see [“UltraLite C/C++ API reference” on page 93](#).

## Creating databases

An UltraLite database can be created using any of the following methods:

- The **Create Database Wizard** in Sybase Central.
- The command line utility ulinit or ulload, from a SQL Anywhere database reference or an XML schema file.
- Calling the CreateDatabase method.

Using Sybase Central, a database can be interactively created with appropriate definitions for the desired tables and other schema-related items.

Applications that create a database by calling the ULCreateDatabase method also require you to execute either the SQL CREATE statements to create tables and index definitions, or the ALTER DATABASE statement.

---

# Application development

This section provides development notes for the UltraLite C/C++ API.



---

# Developing applications using the UltraLite C++ API

## Quick start guide to UltraLite C++ application development

The following procedure is generally used when creating an application using the UltraLite C++ API:

1. Initialize a `ULDatabaseManager` object. See [“ULDatabaseManager class” on page 139](#).
2. (Optional) Enable features in the UltraLite runtime library. See [“Compiling and linking your application” on page 26](#).
3. Use an UltraLite database. You can open a connection to an existing database, create a new one, drop an existing database, or validate that an existing database has no file corruption.
4. Finalize the `ULDatabaseManager` object.

The `ULDatabaseManager` object should only be initialized once in your application and then finalized when your application is terminating. All methods on the `ULDatabaseManager` class are static. Use the `ULError` class to get error information throughout your UltraLite application.

## iPhone and Mac OS X considerations

### Development environment

The development environment for iPhone and Mac OS X is Xcode.

### Build settings

To reference the UltraLite header files and library it is convenient to create a user-defined build setting set to the location of the SQL Anywhere installation directory. For example, set `SQLANY_ROOT` to `/Applications/SQLAnywhere12`. To create this setting, open the project editor's **Build** pane and select **Add User-Defined Setting** and enter the name and value.

### Include files

To find the UltraLite include files, add `$(SQLANY_ROOT)/sdk/include` to the **User Header Search Paths** (`USER_HEADER_SEARCH_PATHS`) build setting.

## Unsupported MobiLink client network protocol options

UltraLite for iPhone and/or Mac OS X does not support the following MobiLink client network protocol options:

- `certificate_company`
- `certificate_unit`
- `client_port`
- `identity`
- `identity_password`
- `network_leave_open`
- `network_name`

## Encryption

Both Mac and iPhone UltraLite clients require that the public keys used for end-to-end encryption (i.e. provided to the `e2ee_public_key` stream parameter) must be encapsulated in a PEM encoded X509 certificate (as opposed to a PEM public key file). See “[e2ee\\_public\\_key](#)” [*MobiLink - Client Administration*].

The following encryption standards are not supported:

- ECC encryption (only RSA)
- FIPS approved encryption

## Debugging iPhone applications

The Xcode debugger (GDB) has support for stepping through and breaking on `longjmp()` calls. Applications typically do not use `longjmp`, but the UltraLite runtime library does internally (sometimes, when an error is signalled, for instance). This may cause problems when tracing through application code and stepping over UltraLite calls. If you step over an UltraLite call and get an error from the debugger: Restart the program, set a breakpoint after the problematic line and, instead of stepping over the problematic line, use the **Continue** command - this will have the same effect because the debugger will stop at the following breakpoint, but should avoid problems related to `longjmp` calls. The most likely place to encounter this is when using **OpenConnection()** to open an existing database or determine that the database doesn't exist (an error is signalled when the database doesn't exist).

## See also

- “[Tutorial: Build an iPhone application using the C++ API](#)” on page 339
- “[MobiLink client network protocol options](#)” [*MobiLink - Client Administration*]
- “[Securing UltraLite databases](#)” [*UltraLite - Database Management and Reference*]

# Connecting to a database

UltraLite applications must connect to the database before performing operations on its data. This section describes how to connect to an UltraLite database.



The `ULDatabaseManager` class is used to open a connection to a database. The `ULDatabaseManager` class returns a non-null `ULConnection` object when a connection is established. Use the `ULConnection` object to perform the following tasks:

- Commit or roll back transactions.
- Synchronize data with a MobiLink server.
- Access tables in the database.
- Work with SQL statements.
- Handle errors in your application.

For more information about the `ULConnection` class, see [“ULConnection class” on page 93](#).

### iPhone database file

Ensure you specify a writable path for the database file. Use the `NSSearchPathForDirectoriesInDomains` function to query the `NSDocumentDirectory`, for example.

You can find sample code in the `samples-dir\UltraLite\CustDB\` directory.

### To connect to an UltraLite database

1. Initialize the `ULDatabaseManager` object and enable features in UltraLite using the following code:

```
if( !ULDatabaseManager::Init() ) {
    return 0;
}
ULDatabaseManager::EnableAesDBEncryption();

// Use ULDatabaseManager.Fini() when terminating the app.
```

For more information about the `ULDatabaseManager` class, see [“ULDatabaseManager class” on page 139](#).

2. Open a connection to an existing database or create a new database if the specified database file does not exist using the following code:

```
ULConnection * conn;
ULError ulerr;

conn =
ULDatabaseManager::OpenConnection( "dbf=sample.udb;dbkey=aBcD1234",
&ulerr );
if( conn == NULL ) {
    if( ulerr.GetSQLCode() == SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
        conn =
ULDatabaseManager::CreateDatabase( "dbf=sample.udb;dbkey=aBcD1234",
&ulerr );
        if( conn == NULL ) {
            // write code that uses ulerr to determine what happened
            return 0;
        }
        // add code to create the schema for your database
    } else {
```

```
        // write code that uses ulerr to determine what happened
        return 0;
    }
}
assert( conn != NULL );
```

In this step, you declare a `ULError` object that contains error information in case the connection is not successful. For more information about the `ULError` class, see [“ULError class” on page 155](#).

### Multi-threaded applications

Each connection and all objects created from it should be used by a single thread. If an application requires multiple threads accessing the UltraLite database, each thread requires a separate connection.

## Closing an UltraLite database connection

It is important to release resources when they are no longer being used; otherwise, an UltraLite database file remains in use as long as an application has a connection to the database.

### To close an UltraLite database connection

1. Call the `Close` method to release resources.

Use the following code when the application no longer requires a connection to the database:

```
if( conn != NULL ) {
    conn->Close( &ulerr );
}
```

2. Call the `Fini` method to finalize the `ULDatabaseManager` object.

Use the following code when closing the application.

```
ULDatabaseManager.Fini();
```

### See also

- [“Close method” on page 98](#)
- [“Fini method” on page 146](#)

## Accessing data using SQL statements

UltraLite applications can access table data by executing SQL statements or using the `ULTable` class. This section describes data access using SQL statements.

This section explains how to perform the following tasks using SQL:

- Inserting, deleting, and updating rows.
- Retrieving rows to a result set.

- Scrolling through the rows of a result set.

This section does not describe the SQL language. For more information about the SQL language, see [“UltraLite SQL statements” \[UltraLite - Database Management and Reference\]](#).

For more information about using the ULTable class, see [“Accessing data using ULTable class” on page 17](#).

## Data manipulation: Insert, Delete, and Update

With UltraLite, you can perform SQL data manipulation by using the ExecuteStatement method, a member of the ULPreparedStatement class. See [“ULPreparedStatement class” on page 167](#).

### Referencing parameters in prepared statements

UltraLite indicates query parameters using the ? character. For any INSERT, UPDATE, or DELETE statement, each ? is referenced according to its ordinal position in the prepared statement. For example, the first ? is referred to as parameter 1, and the second as parameter 2.

### To insert a row

1. Declare a ULPreparedStatement using the following code:

```
ULPreparedStatement * prepStmt;
```

2. Prepare a SQL statement for execution.

The following code prepares an INSERT statement for execution:

```
prepStmt = conn->PrepareStatement("INSERT INTO MyTable(MyColumn1) VALUES  
(?)");
```

3. Check for errors when preparing the statement.

For example, the following code is useful when checking for SQL syntax errors:

```
if( prepStmt == NULL ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error  
    return;  
}
```

4. Set values to replace ? characters in the prepared statement.

The following code sets ? characters to "some value" while error checking. For example, an error is caught when the parameter ordinal is out of range for the number of parameters in the prepared statement.

```
if( !prepStmt->SetParameterString(1, "some value") ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error
```

```
    return;  
}
```

5. Execute the prepared statement, inserting the data into the database.

The following code checks for errors that could occur after executing the statement. For example, an error is returned if a duplicate index value is found in a unique index.

```
bool success;  
success = prepStmt->ExecuteStatement();  
if( !success ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error  
} else {  
    // Use the following line if you are interested in the number of rows  
    // inserted ...  
    ul_u_long rowsInserted = prepStmt->GetRowsAffectedCount();  
}
```

6. Clean up the prepared statement resources.

The following code releases the resources used by the prepared statement object. This object should no longer be accessed after the Close method is called.

```
prepStmt->Close();
```

7. Commit the data to the database.

The following code saves the data to the database and prevents data loss. The data from step 5 is lost if the device application terminates unexpectedly before the application completes a commit call.

```
conn->Commit();
```

### To delete a row

1. Declare a ULPreparedStatement using the following code:

```
ULPreparedStatement * prepStmt;
```

2. Prepare a SQL statement for execution.

The following code prepares a DELETE statement for execution:

```
prepStmt = conn->PrepareStatement("DELETE FROM MyTable(MyColumn1) VALUES  
(?)");
```

3. Check for errors when preparing the statement.

For example, the following code is useful when checking for SQL syntax errors:

```
if( prepStmt == NULL ) {  
    const ULError * ulerr;  
    ulerr = conn->GetLastError();  
    // write code to handle the error  
    return;  
}
```

4. Set values to replace ? characters in the prepared statement.

The following code sets ? characters to 7 while error checking. For example, an error is caught when the parameter ordinal is out of range for the number of parameters in the prepared statement.

```
ul_s_long value_to_delete = 7;
if( !prepStmt->SetParameterInt(1, value_to_delete) ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error.
    return;
}
```

5. Execute the prepared statement, deleting the data from the database.

The following code checks for errors that could occur after executing the statement. For example, an error is returned if you try deleting a row that has a foreign key referenced to it.

```
bool success;
success = prepStmt->ExecuteStatement();
if( !success ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
} else {
    // Use the following line if you are interested in the number of rows
    deleted ...
    ul_u_long rowsDeleted = prepStmt->GetRowsAffectedCount();
}
```

6. Clean up the prepared statement resources.

The following code releases the resources used by the prepared statement object. This object should no longer be accessed after the Close method is called.

```
prepStmt->Close();
```

7. Commit the data to the database.

The following code saves the data to the database and prevents data loss. The data from step 5 is lost if the device application terminates unexpectedly before the application completes a commit call.

```
conn->Commit();
```

## To update a row

1. Declare a ULPreparedStatement using the following code:

```
ULPreparedStatement * prepStmt;
```

2. Prepare a SQL statement for execution.

The following code prepares an UPDATE statement for execution:

```
prepStmt = conn->PrepareStatement("UPDATE MyTable SET MyColumn = ? WHERE
MyColumn = ?");
```

3. Check for errors when preparing the statement.

For example, the following code is useful when checking for SQL syntax errors:

```
if( prepStmt == NULL ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
    return;
}
```

4. Set values to replace ? characters in the prepared statement.

The following code sets ? characters to integer values while error checking. For example, an error is caught when the parameter ordinal is out of range for the number of parameters in the prepared statement.

```
bool success;
success = prepStmt->SetParameterInt( 1, 25 );
if( success ) {
    success = prepStmt->SetParameterInt( 2, -1 );
}
if( !success ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
    return;
}
```

5. Execute the prepared statement, updating the data in the database.

The following code checks for errors that could occur after executing the statement. For example, an error is returned if a duplicate index value is found in a unique index.

```
success = prepStmt->ExecuteStatement();
if( !success ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
} else {
    // if you are interested in the number of rows updated ...
    ul_ulong rowsUpdated = prepStmt->GetRowsAffectedCount();
}
```

6. Clean up the prepared statement resources.

The following code releases the resources used by the prepared statement object. This object should no longer be accessed after the Close method is called.

```
prepStmt->Close();
```

7. Commit the data to the database.

The following code saves the data to the database and prevents data loss. The data from step 5 is lost if the device application terminates unexpectedly before the application completes a commit call.

```
conn->Commit();
```

## Data retrieval: SELECT

The SELECT statement allows you to retrieve information from the database. When you execute a SELECT statement, the PreparedStatement.ExecuteQuery method returns a ResultSet object. See [“ExecuteQuery method” on page 171](#).

### To execute a SELECT statement

1. Declare the required variables using the following code:

```
ULPreparedStatement * prepStmt;
ULResultSet * resultSet;
```

2. Prepare a SQL statement for execution.

The following code prepares a SELECT statement for execution:

```
prepStmt = conn->PrepareStatement("SELECT MyColumn1 FROM MyTable");
```

3. Check for errors when preparing the statement.

For example, the following code is useful when checking for SQL syntax errors:

```
if( prepStmt == NULL ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
    return;
}
```

4. Execute the SQL and return a result set object that can be used to move the results of the query.

```
resultSet = prepStmt->ExecuteQuery();
if( resultSet == NULL ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
    prepStmt->Close();
    return;
}
```

5. Traverse the rows by calling the Next method. Store the result as a string and store them in a buffer.

The Next method moves to the next row of the result set. The ULResultSet object is positioned on a row if the call returns true; otherwise, if the call returns false, all the rows have been traversed.

```
while( resultSet->Next() ) {
    char buffer[ 100 ];
    resultSet->GetString( 1, buffer, 100 );
    printf( "MyColumn = %s\n", buffer );
}
```

6. Clean up the prepared statement and result set object resources.

The prepared statement object should not be accessed after the Close method is called.

```
resultSet->Close();  
prepStmt->Close();
```

## Navigating SQL result sets

You can navigate through a result set using methods associated with the `ULResultSet` class. For more information about the `ULResultSet` class, see [“ULResultSet class” on page 179](#).

The result set class provides you with the following methods to navigate a result set:

- **AfterLast** Position immediately after the last row.
- **BeforeFirst** Position immediately before the first row.
- **First** Move to the first row.
- **Last** Move to the last row.
- **Next** Move to the next row.
- **Previous** Move to the previous row.
- **Relative(offset)** Move a specified number of rows relative to the current row, as specified by the signed offset value. Positive offset values move forward in the result set, relative to the current pointer position in the result set. Negative offset values move backward in the result set. An offset value of zero does not move the current location, but allows you to repopulate the row buffer.

## Result set schema description

The `GetResultSetSchema` method allows you to retrieve schema information about a result set, such as column names, total number of columns, column scales, column sizes, and column SQL types.

### Example

The following example demonstrates how to use the `GetResultSetSchema` method to display schema information in a command prompt:

```
const char * name;  
int column_count;  
const ULResultSetSchema & rss = prepStmt->GetResultSetSchema();  
int column_count = rss.GetColumnCount();  
for( int i = 1; i < column_count; i++ ) {  
    name = rss.GetColumnName( i );  
    printf( "id = %d, name = %s\n", i, name );  
}
```

In this example, required variables are declared and the `ULResultSetSchema` object is assigned. It is possible to get a `ULResultSetSchema` object from the result set object itself, but this example demonstrates how the schema is available after the statement is prepared and before the query is executed. The number of rows in the result set are counted, and the name of each column is displayed.



**See also**

- [“GetResultSetSchema method” on page 174](#)

## Accessing data using ULTable class

UltraLite applications can access table data by executing SQL statements or using the ULTable class. This section describes data access using the ULTable class.

This section explains how to perform the following tasks using the ULTable class:

- Scrolling through the rows of a table.
- Accessing the values of the current row.
- Using find and lookup methods to locate rows in a table.
- Inserting, deleting, and updating rows.

For more information about accessing data by executing SQL statements, see [“Accessing data using SQL statements” on page 10](#).

## Navigating the rows of a table

The UltraLite C++ API provides you with several methods to navigate a table to perform a wide range of navigation tasks. For more information about the ULTable class, see [“ULTable class” on page 235](#).

The ULTable object provides you with the following methods to navigate a table:

- **AfterLast** Position immediately after the last row.
- **BeforeFirst** Position immediately before the first row.
- **First** Move to the first row.
- **Last** Move to the last row.
- **Next** Move to the next row.
- **Previous** Move to the previous row.
- **Relative(offset)** Move a specified number of rows relative to the current row, as specified by the signed offset value. Positive offset values move forward in the result set, relative to the current pointer position in the result set. Negative offset values move backward in the result set. An offset value of zero does not move the current location, but allows you to repopulate the row buffer.

## Example

The following example opens the table named MyTable and displays the value of the column named MyColumn for each row:

```
char buffer[ 100 ];
ul_column_num column_id;
ULTable * tbl = conn->OpenTable( "MyTable" );
if( tbl == NULL ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
    return;
}
column_id = tbl->GetTableSchema().GetColumnID( "MyColumn" );
if( column_id == 0 ) {
    // the column "MyColumn" likely does not exist. Handle the error.
    tbl->Close();
    return;
}
while( tbl->Next() ) {
    tbl->GetString( column_id, buffer, 100 );
    printf( "%s\n", buffer );
}
tbl->Close();
```

You expose the rows of the table to the application when you open the ULTable object. By default, the rows are ordered by primary key value but you can specify an index when opening a table to access the rows in a particular order.

## Example

The following example moves to the first row of the MyTable table as ordered by the ix\_col index:

```
ULTable * tbl = conn->OpenTable( "MyTable", "ix_col" );
```

## See also

- [“ULTable class” on page 235](#)

## UltraLite modes

The UltraLite mode determines how values in the buffer are used. You can set the UltraLite mode to one of the following:

- **Insert mode** Data in the buffer is added to the table as a new row when the insert method is called.
- **Update mode** Data in the buffer replaces the current row when the update method is called.
- **Find mode** Locates a row whose value exactly matches the data in the buffer when one of the find methods is called.
- **Lookup mode** Locates a row whose value matches or is greater than the data in the buffer when one of the lookup methods is called.

The mode is set by calling the corresponding method to set the mode. For example, `InsertBegin`, `BeginUpdate`, `FindBegin`, and so on.

## Accessing the current row

A `ULTable` object is always located at one of the following positions:

- Before the first row of the table.
- On a row of the table.
- After the last row of the table.

If the `ULTable` object is positioned on a row, you can use one of a set of methods appropriate for the data type to retrieve or modify the value of the columns in that row.

### Retrieving column values

The `ULTable` object provides a set of methods for retrieving column values. These methods take the column name or ID as the argument.

The following example demonstrates two ways to get an age value out of an open table, assuming that age is the first column of the table:

```
ul_s_long age1 = tbl->GetInt( 1 );
ul_s_long age2 = tbl->GetInt( "age" );
assert( age1 == age2 );
```

Using the column ID version of value retrieval has performance benefits when values are retrieved in a loop.

### Modifying column values

In addition to the methods for retrieving values, there are methods for setting values. These methods take the column name or ID and the value as arguments.

For example demonstrates two ways to set a string value for a row with string columns of `lname` and `fname`, assuming that `lname` is the first column in the table.

```
tbl->SetString( 1, last_name );
tbl->SetString( "fname", first_name );
```

By setting column values, you do not directly alter the data in the database. You can assign values to the columns, even if you are before the first row or after the last row of the table. Do not attempt to access data when the current row is undefined. For example, attempting to fetch the column value in the following example is incorrect:

```
// This code is incorrect
tbl->BeforeFirst();
tbl = tbl.GetInt( cust_id );
```

### Casting values

The method you choose should match the data type you want to assign. UltraLite automatically casts database data types where they are compatible, so that you can use the `GetString` method to fetch an

integer value into a string variable, and so on. See “[Converting data types explicitly](#)” [*UltraLite - Database Management and Reference*].

## Searching rows

UltraLite has different modes of operation for working with data. You can use two of these modes, find and lookup, for searching. The ULTable object has methods corresponding to these modes for locating particular rows in a table.

### Note

The columns you search with Find and Lookup methods must be in the index that is used to open the table.

- **Find methods** Move to the first row that exactly matches specified search values, under the sort order specified when the ULTable object was opened. If the search values cannot be found, the application is positioned before the first or after the last row.
- **Lookup methods** Move to the first row that matches or is greater than a specified search value, under the sort order specified when the ULTable object was opened.

### Example

This example uses a table named MyTable that was created using the following SQL statements:

```
CREATE TABLE MyTable( id int primary key, lname char(100), fname char(100) )
CREATE INDEX ix_lname ON MyTable ( lname )
```

The following code displays all the fname column contents where lname column is "Smith":

```
ULTable * tbl = conn->OpenTable( "MyTable", "ix_lname" );
char buffer[ 100 ];
bool found;
tbl->FindBegin(); // enter "Find mode"
tbl->SetString( "lname", "Smith" ); // set pointer row buffer to "Smith"
found = tbl->FindFirst();
while( found ) {
    tbl->GetString( 3, buffer, 100 );
    printf( "%s\n", buffer );
    found = tbl->FindNext();
}
tbl->Close();
```

In this example, the tbl variable is set to open MyTable using the ix\_lname index so that rows are returned in same order as the lname column. ULTable objects use the values in the row buffer when they execute a find. This buffer is specified as "Smith", as defined by the SetString method. The FindFirst method indicates that traversal should begin at the first row that has lname set to "Smith"; the pointer is positioned after the last row of the table if there are no rows where lname is set to "Smith". The fname is retrieved by the GetString method because The fname has a column ID of 3. The results are then displayed, and the resources are freed.

### See also

- “[ULTable class](#)” on page 235

## Updating rows

The following procedure updates a row in a table.

### To update a row

1. Move to the row you want to update.

You can move to a row by scrolling through the table or by searching the table using find and lookup methods.

2. Enter update mode.

For example, the following instruction enters update mode on table tbl.

```
tbl->BeginUpdate();
```

3. Set the new values for the row to be updated. For example, the following instruction sets the id column in the buffer to 3.

```
tbl->SetInt("id", 3);
```

4. Execute the Update.

```
tbl->Update();
```

#### Caution

When using the Find and Update methods, your pointer may not be in the expected position after updating a column that is involved in the search criteria. In some instances, it is recommended that you use a SQL statement when updating several rows.

After the update operation, the current row is the row that has been updated.

The UltraLite C++ API does not commit changes to the database until use the Commit method. See [“Managing transactions” on page 22](#).

#### Caution

Do not update the primary key of a row: delete the row and add a new row instead.

## Inserting rows

The steps to insert a row are very similar to those for updating rows, except that there is no need to locate a row in the table before carrying out the insert operation.

If you do not set a value for one of the columns, and that column has a default, the default value is used. If the column has no default, one of the following entries is used:

- For nullable columns, NULL.
- For numeric columns that disallow NULL, zero.
- For character columns that disallow NULL, an empty string.
- To explicitly set a value to NULL, use the SetNull method.

### Example

The following code demonstrates new row insertion:

```
ULTable * tbl = conn->OpenTable("MyTable");
bool success;
tbl->InsertBegin(); // enter "Insert mode"
tbl->SetInt("id", 3);
tbl->SetString("lname", "Smith");
tbl->SetString("fname", "Mary");
success = tbl->Insert();
conn->Commit();
tbl->Close();
```

In this example, the tbl variable is set to open MyTable. The values for each column are set in the current row buffer; columns can be referenced name or ID. The Insert method causes the temporary row buffer values to be inserted into the database. The results are then committed and displayed. Resources are freed with the Close method.

## Deleting rows

The steps to delete a row are simpler than inserting or updating rows.

The following procedure deletes a row.

### To delete a row

1. Move to the row you want to delete.
2. Execute the Delete method.

```
tbl->Delete();
```

## Managing transactions

The UltraLite C++ API does not support AutoCommit mode. Transactions are started implicitly by the first statement to modify the database, and must be explicitly committed or rolled back.

### To commit a transaction

- Execute a conn->Commit statement, where conn is a valid ULConnection pointer.

See [“Commit method” on page 98](#).

### To roll back a transaction

- Execute a `conn->Rollback` statement, where `conn` is a valid `ULConnection` pointer.

See [“Rollback method” on page 126](#).

For more information about transaction management in UltraLite, see [“UltraLite transaction processing” \[UltraLite - Database Management and Reference\]](#).

## Accessing schema information

You can programmatically retrieve result set or database structure descriptions. These descriptions are known as schema information, and this information is available through the UltraLite C API schema classes.

### Note

You cannot modify the schema using the UltraLite C API. You can only retrieve the schema information.

You can access the following schema objects and information:

- **ULResultSetSchema** Describes a query or data in a table. It exposes the identifier, name, and type information of each column, and the number of columns in the table. `ULResultSetSchema` classes can be retrieved from the following classes:
  - `ULPreparedStatement`
  - `ULResultSet`
  - `ULTable`
- **ULDatabaseSchema** Exposes the number and names of the tables and publications in the database, and the global properties such as the format of dates and times. `ULDatabaseSchema` classes can be retrieved from `ULConnection` classes.
- **ULTableSchema** Exposes information about the column and index configurations. The column information in the `ULTableSchema` class complements the information available from the `ULResultSetSchema` class. For example, you can determine whether columns have default values or permit null values. `ULTableSchema` classes can be retrieved from `ULTable` classes.
- **ULIndexSchema** Returns information about the column in the index. `ULIndexSchema` classes can be retrieved from `ULTableSchema` classes.

The `ULResultSetSchema` class is returned as a constant reference unlike the `ULDatabaseSchema`, `ULTableSchema` and `ULIndexSchema` classes, which are returned as pointers. You cannot close a class that returns a constant reference but you must close classes that are returned as pointers.

The following code demonstrates proper and improper use of schema class closure:

```
// This code demonstrates proper use of the ULResultSetSchema class:  
const ULResultSetSchema & rss = prepStmt->GetResultSetSchema();
```

```
ccount = prepStmt->GetSchema().GetColumnCount();

// This code demonstrates proper use of the ULDatabaseSchema class:
ULDatabaseSchema * dbs = conn->GetResultSetSchema();
tcount = dbs->GetTableCount();
dbs->Close(); // This line is required.

// This code demonstrates improper use of the ULDatabaseSchema class
// because the object needs to be closed using the Close method:
tcount = conn->GetResultSetSchema()->GetTableCount();
```

### See also

- [“ULPreparedStatement class” on page 167](#)
- [“ULResultSet class” on page 179](#)
- [“ULTable class” on page 235](#)
- [“ULConnection class” on page 93](#)

## Handling errors

The UltraLite C++ API includes a `ULError` object that should be used to retrieve error information. Several methods in the API return a boolean value, indicating whether the method call was successful. In some instances, null is returned when an error occurs. The `ULConnection` object contains a `GetLastError` method, which returns a `ULError` object.

Use `SQLCode` to diagnose an error. In addition to the `SQLCode`, you can use the `GetParameterCount` and `GetParameter` methods to determine whether additional parameters exist to provide additional information about the error.

In addition to explicit error handling, UltraLite supports an error callback function. If you register a callback function, UltraLite calls the function whenever an UltraLite error occurs. The callback function does not control application flow, but does enable you to be notified of all errors. Use of a callback function is particularly helpful during application development and debugging. For more information about using the callback function, see [“Tutorial: Build an application using the C++ API” on page 329](#).

For a sample callback function, see [“ULSetErrorCallback method” on page 307](#).

For a list of error codes thrown by the UltraLite C++ API, see [“SQL Anywhere error messages sorted by Sybase error code” \[Error Messages\]](#).

## Authenticating users

UltraLite databases can define up to four user IDs. UltraLite databases are created with a default user ID and password of **DBA** and **sql**, respectively. All connections to an UltraLite database must supply a user ID and password. Password changes and user ID additions and deletions can be performed once a connection is established.

You cannot directly change a user ID. You can add a user ID and delete an existing user ID.



**Note**

You can use SQL statements to authenticate users as an alternative to using the UltraLite C API. See [“CREATE USER statement” \[SQL Anywhere Server - SQL Reference\]](#).

**To add a user or change a password for an existing user**

1. Connect to the database as an existing user.
2. Grant the user connection authority with the desired password using the `conn->GrantConnectTo` method.

This procedure is the same whether you are adding a new user or changing the password of an existing user. See [“GrantConnectTo method” on page 118](#).

**To delete an existing user**

1. Connect to the database as an existing user.
2. Revoke the user's connection authority using the `conn->RevokeConnectFrom` method.

See [“RevokeConnectFrom method” on page 126](#).

## Encrypting data

You can choose to either encrypt or obfuscate an UltraLite database using the UltraLite C++ API. Encryption provides very secure representation of the data in the database whereas obfuscation provides a simplistic level of security that is intended to prevent casual observation of the contents of the database.

For background information, see [“Choosing database creation parameters for UltraLite” \[UltraLite - Database Management and Reference\]](#).

### Encryption

To create a database with encryption, specify an encryption key by specifying the `key=` connection parameter in the connection string. When you call the `CreateDatabase` method, the database is created and encrypted with the specified key. See [“UltraLite DBKEY connection parameter” \[UltraLite - Database Management and Reference\]](#).

Use the `ChangeEncryptionKey` method to change the encryption key for an existing database. See [“ChangeEncryptionKey method” on page 96](#).

After the database is encrypted, all connections to the database must specify the correct encryption key. Otherwise, the connection fails.

### Obfuscation

Specify `obfuscate=1` as a database creation parameter to obfuscate the database. See [“Securing UltraLite databases” \[UltraLite - Database Management and Reference\]](#).

## Synchronizing data

UltraLite applications can synchronize data with a central database. Synchronization requires the MobiLink synchronization software included with SQL Anywhere.

The UltraLite C++ API supports TCP/IP, TLS, HTTP, and HTTPS synchronization. Synchronization is initiated by the UltraLite application. The methods and properties of the connection object can be used to control synchronization.

For more information about synchronization, see “UltraLite clients” [[UltraLite - Database Management and Reference](#)].

For more information about the `ul_sync_info` structure used for synchronization, see “[ul\\_sync\\_info structure](#)” on page 81.

For more information about synchronization parameters, see “[Synchronization parameters for UltraLite](#)” [[UltraLite - Database Management and Reference](#)].

## Compiling and linking your application

### Runtime libraries for Windows Mobile

The Windows Mobile libraries are in the `install-dir\UltraLite\ce\arm.50\Lib` directory.

The following static and dynamic libraries are provided for Windows Mobile:

- **ulbase.lib** A library containing extra functions that can not be provided in a separate dynamic link library (DLL). You must always link to this library.
- **ulrt.lib** A static library version of the in-process UltraLite runtime. You do not need to deploy additional files if you link to this library.
- **ulrtc.lib** A static library for use with the UltraLite engine for multi-process access to an UltraLite database. You need to deploy `uleng12.exe` with your application if you link to this library. See “UltraLite Engine utility (uleng12)” [[UltraLite - Database Management and Reference](#)].
- **ulimp.lib** An import library for `ulrt12.dll`, the DLL version of the in-process runtime. You need to deploy `ulrt12.dll` with your application if you link to this library.

Specify the following compilation option when linking against this library:

```
/DUL_USE_DLL
```

- **ulrsa.lib** A static library that provides RSA security
- **ulecc.lib** A static library that provides ECC security
- **ulfips.lib** A static library that provides RSA FIPS 140-2 security

## Runtime libraries for Windows desktops

The Windows libraries are in the *install-dir\UltraLite\Windows\x86\Lib\vs8* and *install-dir\UltraLite\Windows\x64\Lib\vs8* directories.

The following libraries are included:

- **ulbase.lib** A library containing extra functions that can not be provided in a separate dynamic link library (DLL). You must always link to this library.
- **ulrt.lib** A static library version of the in-process UltraLite runtime. You do not need to deploy additional files if you link to this library.
- **ulrtc.lib** A static library for use with the UltraLite engine for multi-process access to an UltraLite database. You need to deploy *uleng12.exe* with your application if you link to this library. See “UltraLite Engine utility (uleng12)” [[UltraLite - Database Management and Reference](#)].
- **ulimp.lib** An import library for *ulrt12.dll*, the DLL version of the in-process runtime. You need to deploy *ulrt12.dll* with your application if you link to this library.

Specify the following compilation option when linking against this library:

```
/DUL_USE_DLL
```

- **ulrsa.lib** A static library that provides RSA security
- **ulecc.lib** A static library that provides ECC security
- **ulfips.lib** A static library that provides RSA FIPS 140-2 security

## Runtime libraries for Linux

The Linux libraries are in the *install-dir/ultralite/linux/x86/586/lib* directory.

The following libraries are provided for Linux:

- **libulbase.a** A static ulbase library. You must always link to this library.
- **libulrt.a** A static library version of the in-process UltraLite runtime. You do not need to deploy additional files if you link to this library.
- **libulrtc.a** A static client library for use with the UltraLite engine for multi-process access to an UltraLite database. You need to deploy *install-dir/ultralite/linux/x86/586/uleng12* with your application if you link to this library.
- **libulrsa.a** A static library that provides RSA security
- **libulecc.a** A static library that provides ECC security

The linker/compiler options for *libulrt.a* are:

```
-L<install-dir>/ultralite/linux/x86/586/lib -lulrt -|ulbase
```

and for the engine:

```
-L<install-dir>/ultralite/linux/x86/586/lib -lulrtc -lulbase
```

The headers command line switch is:

```
-I<install-dir>/sdk/include
```

### Runtime libraries for iPhone

UltraLite runtimes must be built after installation. Follow the instructions provided in *install-dir/ultralite/iphone/readme.txt*

To link to the UltraLite runtime library, either:

Control-click the **Frameworks** group and select **Add -> Existing Files...**, then navigate to the *install-dir/ultralite/iphone* directory and select *libulrt.a*.

### OR

Add the following to the **Other Linker Flags** (*OTHER\_LDFLAGS*) build setting:

```
-L$(SQLANY_ROOT)/ultralite/iphone  
-lulrt
```

where *SQLANY\_ROOT* is a custom build setting set to the SQL Anywhere installation directory.

In addition, the *CFNetwork.framework* and *Security.framework* frameworks are required. To add these, control-click the **Frameworks** group and select **Add -> Existing Frameworks...** and select from the list.

### Runtime libraries for Mac OS X

To link to the UltraLite runtime library, Control-click the **Frameworks** group and select **Add -> Existing Files...**, then navigate to the *install-dir/ultralite/macosx/x86\_64* directory and select *libulrt.a* and also *libulbase.a*.

In addition, the *CoreFoundation.framework*, *CoreServices.framework*, and *Security.framework* frameworks are required. To add these, Control-click the **Frameworks** group and select **Add -> Existing Frameworks...** and select from the list.

---

# Developing applications using embedded SQL

This section describes how to write database access code for embedded SQL UltraLite applications.

For an overview of the UltraLite C/C++ development process, see [“UltraLite for C/C++ developers” on page 1](#).

For embedded SQL reference information, see [“UltraLite Embedded SQL API reference” on page 259](#).

For more information about the SQL preprocessor, see [“SQL preprocessor for UltraLite utility \(sqlpp\)” \[UltraLite - Database Management and Reference\]](#).

## Example of embedded SQL

Embedded SQL is an environment that is a combination of C/C++ program code and pseudo-code. The pseudo-code that can be interspersed with traditional C/C++ code is a subset of SQL statements. A preprocessor converts the embedded SQL statements into function calls that are part of the actual code that is compiled to create the application.

Following is a very simple example of an embedded SQL program. It illustrates updating an UltraLite database record by changing the surname of employee 195.

```
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
main( )
{
    db_init( &sqlca );
    EXEC SQL WHENEVER SQLERROR GOTO error;
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";
    EXEC SQL UPDATE employee
        SET emp_lname = 'Johnson'
        WHERE emp_id = 195;
    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
    return( 0 );
error:
    printf( "update unsuccessful: sqlcode = %ld\n",
        sqlca.sqlcode );
    return( -1 );
}
```

Although this example is too simplistic to be useful, it illustrates the following aspects common to all embedded SQL applications:

- Each SQL statement is prefixed with the keywords EXEC SQL.
- Each SQL statement ends with a semicolon.
- Some embedded SQL statements are not part of standard SQL. The INCLUDE SQLCA statement is one example.

- In addition to SQL statements, embedded SQL also provides library functions to perform some specific tasks. The functions `db_init` and `db_fini` are two examples of library function calls.

### Initialization

The above sample code illustrates initialization statements that must be included before working with the data in an UltraLite database:

1. Define the SQL Communications Area (SQLCA), using the following command:

```
EXEC SQL INCLUDE SQLCA;
```

This definition must be the first embedded SQL statement, so a natural place for it is the end of the include list.

If you have multiple `.sql` files in your application, each file must have this line.

2. The first database action must be a call to an embedded SQL library function named `db_init`. This function initializes the UltraLite runtime library. Only embedded SQL definition statements can be executed before this call.

See “[db\\_init method](#)” on page 259.

3. You must use the SQL `CONNECT` statement to connect to the UltraLite database.

### Preparing to exit

The above sample code demonstrates the sequence of calls required when preparing to exit:

1. Commit or rollback any outstanding changes.
2. Disconnect from the database.
3. End your SQL work with a call to a library function named `db_fini`.

When you exit, any uncommitted database changes are automatically rolled back.

### Error handling

There is virtually no interaction between the SQL and C code in this example. The C code only controls the flow of the program. The `WHENEVER` statement is used for error checking. The error action, `GOTO` in this example, is executed whenever any SQL statement causes an error.

## Structure of embedded SQL programs

All embedded SQL statements start with the words `EXEC SQL` and end with a semicolon. Normal C-language comments are allowed in the middle of embedded SQL statements.

Every C program using embedded SQL must contain the following statement before any other embedded SQL statements in the source file.

```
EXEC SQL INCLUDE SQLCA;
```

The first embedded SQL executable statement in the program must be a SQL CONNECT statement. The CONNECT statement supplies connection parameters that are used to establish a connection to the UltraLite database.

Some embedded SQL commands do not generate any executable C code, or do not involve communication with the database. Only these commands are allowed before the CONNECT statement. Most notable are the INCLUDE statement and the WHENEVER statement for specifying error processing.

## Initializing the SQL Communications Area

The SQL Communications Area (SQLCA) is an area of memory that is used for communicating statistics and errors from the application to the database and back to the application. The SQLCA is used as a handle for the application-to-database communication link. It is passed explicitly to all database library functions that communicate with the database. It is implicitly passed in all embedded SQL statements.

UltraLite defines a SQLCA global variable for you in the generated code. The preprocessor generates an external reference for the global SQLCA variable. The external reference is named `sqlca` and is of type SQLCA. The actual global variable is declared in the imports library.

The SQLCA type is defined in the header file `install-dir\SDK\Include\sqlca.h`.

After declaring the SQLCA (`EXEC SQL INCLUDE SQLCA;`), but before your application can carry out any operations on a database, you must initialize the communications area by calling `db_init` and passing it the SQLCA:

```
db_init( &sqlca );
```

### SQLCA provides error codes

You reference the SQLCA to test for a particular error code. The `sqlcode` field contains an error code when a database request causes an error. Macros are defined for referencing the `sqlcode` field and some other fields in the `sqlca`.

## SQLCA fields

The SQLCA contains the following fields:

- **sqlcaid** An 8-byte character field that contains the string SQLCA as an identification of the SQLCA structure. This field helps in debugging when you are looking at memory contents.
- **sqlcabc** A long integer that contains the length in bytes of the SQLCA structure.
- **sqlcode** A long integer that contains an error code when the database detects an error on a request. Definitions for the error codes are in the header file `install-dir\SDK\Include\sqlerr.h`. The error code is 0 (zero) for a successful operation, a positive value for a warning, and a negative value for an error.

You can access this field directly using the `SQLCODE` macro.

For a list of error codes, see [“SQL Anywhere error messages”](#) [*Error Messages*].

- **sqlerrml** The length of the information in the sqlerrmc field.

UltraLite applications do not use this field.

- **sqlerrmc** May contain one or more character strings to be inserted into an error message. Some error messages contain a placeholder string (*%I*) which is replaced with the text in this field.

UltraLite applications do not use this field.

- **sqlerrp** Reserved.
- **sqlerrd** A utility array of long integers.
- **sqlwarn** Reserved.

UltraLite applications do not use this field.

- **sqlstate** The SQLSTATE status value.

UltraLite applications do not use this field.

## Connecting to a database

To connect to an UltraLite database from an embedded SQL application, include the EXEC SQL CONNECT statement in your code after initializing the SQLCA.

The CONNECT statement has the following form:

### EXEC SQL CONNECT USING

```
'uid=user-name;pwd=password;dbf=database-filename';
```

The connection string (enclosed in single quotes) may include additional database connection parameters.

For more information about database connection parameters, see [“UltraLite connection parameters”](#) [*UltraLite - Database Management and Reference*].

For more information about the CONNECT statement, see [“CONNECT statement \[ESQL\] \[Interactive SQL\]”](#) [*SQL Anywhere Server - SQL Reference*].

## Managing multiple connections

If you want more than one database connection in your application, you can either use multiple SQLCAs or you can use a single SQLCA to manage the connections.



## To use multiple SQLCAs

### Managing multiple SQLCAs

1. Each SQLCA used in your program must be initialized with a call to **db\_init** and cleaned up at the end with a call to **db\_fini**.

See [“db\\_init method” on page 259](#).

2. The embedded SQL statement SET SQLCA is used to tell the SQL preprocessor to use a specific SQLCA for database requests. Usually, a statement such as the following is used at the top of your program or in a header file to set the SQLCA reference to point at task specific data:

```
EXEC SQL SET SQLCA 'task_data->sqlca';
```

This statement does not generate any code and does not affect performance. It changes the state within the preprocessor so that any reference to the SQLCA will use the given string.

For more information about creating SQLCAs, see [“SET SQLCA statement \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#).

## To use a single SQLCA

As an alternative to using multiple SQLCAs, you can use a single SQLCA to manage more than one connection to a database.

Each SQLCA has a single active or current connection, but that connection can be changed. Before executing a command, use the SET CONNECTION statement to specify the connection on which the command should be executed.

See [“SET CONNECTION statement \[Interactive SQL\] \[ESQL\]” \[SQL Anywhere Server - SQL Reference\]](#).

# Using host variables

Embedded SQL applications use host variables to communicate values to and from the database. Host variables are C variables that are identified to the SQL preprocessor in a declaration section.

## Declaring host variables

Define host variables by placing them within a declaration section. Host variables are declared by surrounding the normal C variable declarations with BEGIN DECLARE SECTION and END DECLARE SECTION statements.

Whenever you use a host variable in a SQL statement, you must prefix the variable name with a colon (:) so the SQL preprocessor knows you are referring to a (declared) host variable and distinguish it from other identifiers allowed in the statement.

You can use host variables in place of value constants in any SQL statement. When the database server executes the command, the value of the host variable is read from or written to each host variable. Host variables cannot be used in place of table or column names.

The SQL preprocessor does not scan C language code except inside a declaration section. Initializers for variables are allowed inside a declaration section, while **typedef** types and structures are not permitted.

The following sample code illustrates the use of host variables with an INSERT command. The variables are filled in by the program and then inserted into the database:

```
/* Declare fields for personal data. */
EXEC SQL BEGIN DECLARE SECTION;
    long employee_number = 0;
    char employee_name[50];
    char employee_initials[8];
    char employee_phone[15];
EXEC SQL END DECLARE SECTION;
/* Fill variables with appropriate values. */
/* Insert a row in the database. */
EXEC SQL INSERT INTO Employee
    VALUES (:employee_number, :employee_name,
            :employee_initials, :employee_phone );
```

## Data types in embedded SQL

To transfer information between a program and the database server, every data item must have a data type. You can create a host variable with any one of the supported types.

Only a limited number of C data types are supported as host variables. Also, certain host variable types do not have a corresponding C type.

Macros defined in the *sqlca.h* header file can be used to declare a host variable of type VARCHAR, FIXCHAR, BINARY, DECIMAL, or SQLDATETIME. These macros are used as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
    DECL_VARCHAR( 10 ) v_varchar;
    DECL_FIXCHAR( 10 ) v_fixchar;
    DECL_BINARY( 4000 ) v_binary;
    DECL_DECIMAL( 10, 2 ) v_packed_decimal;
    DECL_DATETIME v_datetime;
EXEC SQL END DECLARE SECTION;
```

The preprocessor recognizes these macros within a declaration section and treats the variable as the appropriate type.

The following data types are supported by the embedded SQL programming interface:

- **16-bit signed integer**

```
short int I;
unsigned short int I;
```

- **32-bit signed integer**

```
long int l;
unsigned long int l;
```

- **4-byte floating-point number**

```
float f;
```

- **8-byte floating-point number**

```
double d;
```

- **Packed decimal number**

```
DECL_DECIMAL(p,s)
typedef struct TYPE_DECIMAL {
    char array[1];
} TYPE_DECIMAL;
```

- **Null terminated, blank-padded character string**

```
char a[n]; /* n > 1 */
char *a; /* n = 2049 */
```

Because the C-language array must also hold the NULL terminator, a char a[n] data type maps to a CHAR(n - 1) SQL data type, which can hold -1 characters.

#### Pointers to char, WCHAR, and TCHAR

The SQL preprocessor assumes that a **pointer to char** points to a character array of size 2049 bytes and that this array can safely hold 2048 characters, plus the NULL terminator. In other words, a char\* data type maps to a CHAR(2048) SQL type. If that is not the case, your application may corrupt memory.

If you are using a 16-bit compiler, requiring 2049 bytes can make the program stack overflow. Instead, use a declared array, even as a parameter to a function, to let the SQL preprocessor know the size of the array. WCHAR and TCHAR behave similarly to char.

- **NULL terminated UNICODE or wide character string** Each character occupies two bytes of space and so may contain UNICODE characters.

```
WCHAR a[n]; /* n > 1 */
```

- **NULL terminated system-dependent character string** A TCHAR is equivalent to a WCHAR for systems that use UNICODE (for example, Windows Mobile) for their character set; otherwise, a TCHAR is equivalent to a char. The TCHAR data type is designed to support character strings in either kind of system automatically.

```
TCHAR a[n]; /* n > 1 */
```

- **Fixed-length blank padded character string**

```
char a; /* n = 1 */
DECL_FIXCHAR(n) a; /* n >= 1 */
```

- **Variable-length character string with a two-byte length field** When supplying information to the database server, you must set the length field. When fetching information from the database server, the server sets the length field (not padded).

```
DECL_VARCHAR(n) a; /* n >= 1 */
typedef struct VARCHAR {
    a_sql_ulen len;
    TCHAR array[1];
} VARCHAR;
```

- **Variable-length binary data with a two-byte length field** When supplying information to the database server, you must set the length field. When fetching information from the database server, the server sets the length field.

```
DECL_BINARY(n) a; /* n >= 1 */
typedef struct BINARY {
    a_sql_ulen len;
    unsigned char array[1];
} BINARY;
```

- **SQLDATETIME structure with fields for each part of a timestamp**

```
DECL_DATETIME a;
typedef struct SQLDATETIME {
    unsigned short year; /* for example: 1999 */
    unsigned char month; /* 0-11 */
    unsigned char day_of_week; /* 0-6, 0 = Sunday */
    unsigned short day_of_year; /* 0-365 */
    unsigned char day; /* 1-31 */
    unsigned char hour; /* 0-23 */
    unsigned char minute; /* 0-59 */
    unsigned char second; /* 0-59 */
    unsigned long microsecond; /* 0-999999 */
} SQLDATETIME;
```

The SQLDATETIME structure is used to retrieve fields of the DATE, TIME, and TIMESTAMP type (or anything that can be converted to one of these). Often, applications have their own formats and date manipulation code. Fetching data in this structure makes it easier for you to manipulate this data. Note that DATE, TIME, and TIMESTAMP fields can also be fetched and updated with any character type.

If you use a SQLDATETIME structure to enter a date, time, or timestamp into the database, the day\_of\_year and day\_of\_week members are ignored.

For more information about the date\_format, time\_format, timestamp\_format, and date\_order database options, see “Database options” [[SQL Anywhere Server - Database Administration](#)].

- **DT\_LONGVARCHAR** Long varying length character data. The macro defines a structure, as follows:

```
#define DECL_LONGVARCHAR( size ) \
    struct { a_sql_uint32    array_len;    \
            a_sql_uint32    stored_len;   \
            a_sql_uint32    untrunc_len;  \
            char             array[size+1];\
    }
```

The DECL\_LONGVARCHAR struct may be used with more than 32KB of data. Data may be fetched all at once, or in pieces using the GET DATA statement. Data may be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement. The data is not null terminated.

- **DT\_LONGBINARY** Long binary data. The macro defines a structure, as follows:

```
#define DECL_LONGBINARY( size ) \
    struct { a_sql_uint32    array_len;    \
            a_sql_uint32    stored_len;   \
            a_sql_uint32    untrunc_len;  \
            char             array[size];  \
    }
```

The DECL\_LONGBINARY struct may be used with more than 32KB of data. Data may be fetched all at once, or in pieces using the GET DATA statement. Data may be supplied to the server all at once, or in pieces by appending to a database variable using the SET statement.

The structures are defined in the *install-dir\SDK\Include\sqlca.h* file. The VARCHAR, BINARY, and TYPE\_DECIMAL types contain a one-character array and are not useful for declaring host variables. However, they are useful for allocating variables dynamically or typecasting other variables.

### DATE and TIME database types

There are no corresponding embedded SQL interface data types for the various DATE and TIME database types. These database types are fetched and updated either using the SQLDATETIME structure or using character strings.

There are no embedded SQL interface data types for LONG VARCHAR and LONG BINARY database types.

## Host variable usage

Host variables can be used in the following circumstances:

- In a SELECT, INSERT, UPDATE, or DELETE statement in any place where a number or string constant is allowed.
- In the INTO clause of a SELECT or FETCH statement.
- In CONNECT, DISCONNECT, and SET CONNECT statements, a host variable can be used in place of a user ID, password, connection name, or database name.

Host variables can *never* be used in place of a table name or a column name.

## The scope of host variables

A host-variable declaration section can appear anywhere that C variables can normally be declared, including the parameter declaration section of a C function. The C variables have their normal scope (available within the block in which they are defined). However, since the SQL preprocessor does not scan C code, it does not respect C blocks.

### Preprocessor assumes all host variables are global

As far as the SQL preprocessor is concerned, host variables are globally known in the source module following their declaration. Two host variables cannot have the same name. The only exception to this rule is that two host variables can have the same name if they have identical types (including any necessary lengths).

The best practice is to give each host variable a unique name.

### Examples

Because the SQL preprocessor can not parse C code, it assumes all host variables, no matter where they are declared, are known globally following their declaration.

```
// Example demonstrating poor coding
EXEC SQL BEGIN DECLARE SECTION;
    long emp_id;
EXEC SQL END DECLARE SECTION;
long getManagerID( void )
{
    EXEC SQL BEGIN DECLARE SECTION;
        long manager_id = 0;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SELECT manager_id
        INTO :manager_id
        FROM employee
        WHERE emp_number = :emp_id;
    return( manager_number );
}
void setManagerID( long manager_id )
{
    EXEC SQL UPDATE employee
        SET manager_number = :manager_id
        WHERE emp_number = :emp_id;
}
```

Although the above code works, it is confusing because the SQL preprocessor relies on the declaration inside *getManagerID* when processing the statement within *setManagerID*. You should rewrite this code as follows:

```
// Rewritten example
#if 0
    // Declarations for the SQL preprocessor
    EXEC SQL BEGIN DECLARE SECTION;
        long emp_id;
        long manager_id;
    EXEC SQL END DECLARE SECTION;
#endif
long getManagerID( long emp_id )
{
    long manager_id = 0;
    EXEC SQL SELECT manager_id
        INTO :manager_id
        FROM employee
        WHERE emp_number = :emp_id;
    return( manager_number );
}
void setManagerID( long emp_id, long manager_id )
{
    EXEC SQL UPDATE employee
        SET manager_number = :manager_id
```

```

        WHERE emp_number = :emp_id;
    }

```

The SQL preprocessor sees the declaration of the host variables contained within the `#if` directive because it ignores these directives. On the other hand, it ignores the declarations within the procedures because they are not inside a `DECLARE SECTION`. Conversely, the C compiler ignores the declarations within the `#if` directive and uses those within the procedures.

These declarations work only because variables having the same name are declared to have exactly the same type.

## Using expressions as host variables

Host variables must be simple names because the SQL preprocessor does not recognize pointer or reference expressions. For example, the following statement *does not work* because the SQL preprocessor does not understand the dot operator. The same syntax has a different meaning in SQL.

```

// Incorrect statement:
EXEC SQL SELECT LAST sales_id INTO :mystruct.mymember;

```

Although the above syntax is not allowed, you can still use an expression with the following technique:

- Wrap the SQL declaration section in an `#if 0` preprocessor directive. The SQL preprocessor will read the declarations and use them for the rest of the module because it ignores preprocessor directives.
- Define a macro with the same name as the host variable. Since the SQL declaration section is not seen by the C compiler because of the `#if` directive, no conflict will arise. Ensure that the macro evaluates to the same type host variable.

The following code demonstrates this technique to hide the *host\_value* expression from the SQL preprocessor.

```

#include <sqlerr.h>
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
typedef struct my_struct {
    long    host_field;
} my_struct;
#if 0
    // Because it ignores #if preprocessing directives,
    // SQLPP reads the following declaration.
    EXEC SQL BEGIN DECLARE SECTION;
        long    host_value;
    EXEC SQL END DECLARE SECTION;
#endif
// Make C/C++ recognize the 'host_value' identifier
// as a macro that expands to a struct field.
#define host_value my_s.host_field

```

Since the SQLPP processor ignores directives for conditional compilation, *host\_value* is treated as a *long* host variable and will emit that name when it is subsequently used as a host variable. The C/C++ compiler processes the emitted file and will substitute *my\_s.host\_field* for all such uses of that name.

With the above declarations in place, you can proceed to access *host\_field* as follows.

```

void main( void )
{
    my_struct      my_s;
    db_init( &sqlca );
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "SQL";
    EXEC SQL DECLARE my_table_cursor CURSOR FOR
        SELECT int_col FROM my_table order by int_col;
    EXEC SQL OPEN my_table_cursor;
    for( ; ; ) {
        // :host_value references my_s.host_field
        EXEC SQL FETCH NEXT AllRows INTO :host_value;
        if( SQLCODE == SQLE_NOTFOUND ) {
            break;
        }
        printf( "%ld\n", my_s.host_field );
    }
    EXEC SQL CLOSE my_table_cursor;
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
}

```

You can use the same technique to use other lvalues as host variables:

- pointer indirections

```

*ptr
p_struct->ptr
(*pp_struct)->ptr

```

- array references

```

my_array[ I ]

```

- arbitrarily complex lvalues

## Using host variables in C++

A similar situation arises when using host variables within C++ classes. It is frequently convenient to declare your class in a separate header file. This header file might contain, for example, the following declaration of *my\_class*.

```

typedef short a_bool;
#define TRUE ((a_bool)(1==1))
#define FALSE ((a_bool)(0==1))
public class {
    long host_member;
    my_class(); // Constructor
    ~my_class(); // Destructor
    a_bool FetchNextRow( void );
    // Fetch the next row into host_member
} my_class;

```

In this example, each method is implemented in an embedded SQL source file. Only simple variables can be used as host variables. The technique introduced in the preceding section can be used to access a data member of a class.

```

EXEC SQL INCLUDE SQLCA;
#include "my_class.hpp"

```



```

#if 0
    // Because it ignores #if preprocessing directives,
    // SQLPP reads the following declaration.
    EXEC SQL BEGIN DECLARE SECTION;
        long this_host_member;
    EXEC SQL END DECLARE SECTION;
#endif
// Macro used by the C++ compiler only.
#define this_host_member this->host_member
my_class::my_class()
{
    EXEC SQL DECLARE my_table_cursor CURSOR FOR
        SELECT int_col FROM my_table order by int_col;
    EXEC SQL OPEN my_table_cursor;
}
my_class::~my_class()
{
    EXEC SQL CLOSE my_table_cursor;
}
a_bool my_class::FetchNextRow( void )
{
    // :this_host_member references this->host_member
    EXEC SQL FETCH NEXT AllRows INTO :this_host_member;
    return( SQLCODE != SQLE_NOTFOUND );
}
void main( void )
{
    db_init( &sqlca );
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "SQL";
    {
        my_class mc; // Created after connecting.
        while( mc.FetchNextRow() ) {
            printf( "%ld\n", mc.host_member );
        }
    }
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
}

```

The above example declares `this_host_member` for the SQL preprocessor, but the macro causes C++ to convert it to `this->host_member`. The preprocessor would otherwise not know the type of this variable. Many C/C++ compilers do not tolerate duplicate declarations. The `#if` directive hides the second declaration from the compiler, but leaves it visible to the SQL preprocessor.

While multiple declarations can be useful, you must ensure that each declaration assigns the same variable name to the same type. The preprocessor assumes that each host variable is globally known following its declaration because it can not fully parse the C language.

## Using indicator variables

Indicator variables are C variables that hold supplementary information about a particular host variable. You can use a host variable when fetching or putting data. Use indicator variables to handle NULL values.

An indicator variable is a host variable of type `a_sql_len` that is placed immediately following a regular host variable in a SQL statement. To detect or specify a NULL value, place the indicator variable immediately following a regular host variable in a SQL statement.

### Example

For example, in the following INSERT statement, **:ind\_phone** is an indicator variable.

```
EXEC SQL INSERT INTO Employee
VALUES (:employee_number, :employee_name,
:employee_initials, :employee_phone:ind_phone );
```

On a fetch or execute where no rows are received from the database server (such as when an error or end of result set occurs), then indicator values are unchanged.

**Note**

To allow for the future use of 32 and 64-bit lengths and indicators, the use of short int for embedded SQL indicator variables is deprecated. Use `a_sql_len` instead.

### Indicator variable values

The following table provides a summary of indicator variable usage:

Indicator value	Supplying value to database	Receiving value from database
0	Host variable value	Fetches a non-NULL value.
-1	NULL value	Fetches a NULL value

## Using indicator variables to handle NULL

Do not confuse the SQL concept of NULL with the C-language constant of the same name. In the SQL language, NULL represents either an unknown attribute or inapplicable information. The C-language constant represents a pointer value that does not point to a memory location.

When NULL is used in the SQL Anywhere documentation, it refers to the SQL database meaning given above. The C language constant is referred to as the null pointer (lowercase).

NULL is not the same as any value of the column's defined type. To pass NULL values to the database or receive NULL results back, you require something beyond regular host variables. Indicator variables serve this purpose.

### Using indicator variables when inserting NULL

An INSERT statement can include an indicator variable as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
short int employee_number;
char employee_name[50];
char employee_initials[6];
char employee_phone[15];
a_sql_len ind_phone;
EXEC SQL END DECLARE SECTION;
/* set values of empnum, empname,
initials, and homephone */
```

```
if( /* phone number is known */ ) {
    ind_phone = 0;
} else {
    ind_phone = -1; /* NULL */
}
EXEC SQL INSERT INTO Employee
VALUES (:employee_number, :employee_name,
       :employee_initials, :employee_phone:ind_phone );
```

If the indicator variable has a value of -1, a NULL is written. If it has a value of 0, the actual value of `employee_phone` is written.

### Using indicator variables when fetching NULL

Indicator variables are also used when receiving data from the database. They are used to indicate that a NULL value was fetched (indicator is negative). If a NULL value is fetched from the database and an indicator variable is not supplied, the `SQLE_NO_INDICATOR` error is generated.

For more information about errors and warnings returned in the SQLCA structure, see [“Initializing the SQL Communications Area” on page 31](#).

## Fetching data

Fetching data in embedded SQL is done using the `SELECT` statement. There are two cases:

1. The `SELECT` statement returns no rows or returns exactly one row.
2. The `SELECT` statement returns multiple rows.

### Fetching one row

A single row query retrieves at most one row from the database. A single row query `SELECT` statement may have an `INTO` clause following the select list and before the `FROM` clause. The `INTO` clause contains a list of host variables to receive the value for each select list item. There must be the same number of host variables as there are select list items. The host variables may be accompanied by indicator variables to indicate NULL results.

When the `SELECT` statement is executed, the database server retrieves the results and places them in the host variables.

- If the query returns more than one row, the database server returns the `SQLE_TOO_MANY_RECORDS` error.
- If the query returns no rows, the `SQLE_NOTFOUND` warning is returned.

For more information about errors and warnings returned in the SQLCA structure, see [“Initializing the SQL Communications Area” on page 31](#).

## Example

For example, the following code fragment returns 1 if a row from the employee table is successfully fetched, 0 if the row doesn't exist, and -1 if an error occurs.

```
EXEC SQL BEGIN DECLARE SECTION;
    long int    emp_id;
    char        name[41];
    char        sex;
    char        birthdate[15];
    a_sql_len   ind_birthdate;
EXEC SQL END DECLARE SECTION;
int find_employee( long employee )
{
    emp_id = employee;
    EXEC SQL SELECT emp_fname || ' ' || emp_lname,
        sex, birth_date
        INTO :name, :sex, birthdate:ind_birthdate
        FROM "DBA".employee
        WHERE emp_id = :emp_id;
    if( SQLCODE == SQLE_NOTFOUND ) {
        return( 0 ); /* employee not found */
    } else if( SQLCODE < 0 ) {
        return( -1 ); /* error */
    } else {
        return( 1 ); /* found */
    }
}
```

## Fetching multiple rows

You use a cursor to retrieve rows from a query that has multiple rows in the result set. A cursor is a handle or an identifier for the SQL query result set and a position within that result set.

For an introduction to cursors, see [“Working with cursors” \[SQL Anywhere Server - Programming\]](#).

### To manage a cursor in embedded SQL

1. Declare a cursor for a particular SELECT statement, using the DECLARE statement.
2. Open the cursor using the OPEN statement.
3. Retrieve rows from the cursor one at a time using the FETCH statement.
  - Fetch rows until the SQLE\_NOTFOUND warning is returned. Error and warning codes are returned in the variable SQLCODE, defined in the SQL communications area structure.
4. Close the cursor, using the CLOSE statement.

Cursors in UltraLite applications are always opened using the WITH HOLD option. They are never closed automatically. You must explicitly close each cursor using the CLOSE statement.

The following is a simple example of cursor usage:

```
void print_employees( void )
{
```

```
int status;
EXEC SQL BEGIN DECLARE SECTION;
char name[50];
char sex;
char birthdate[15];
a_sql_len ind_birthdate;
EXEC SQL END DECLARE SECTION;
/* 1. Declare the cursor. */
EXEC SQL DECLARE C1 CURSOR FOR
    SELECT emp_fname || ' ' || emp_lname,
           sex, birth_date
    FROM "DBA".employee
    ORDER BY emp_fname, emp_lname;
/* 2. Open the cursor. */
EXEC SQL OPEN C1;
/* 3. Fetch each row from the cursor. */
for( ;; ) {
    EXEC SQL FETCH C1 INTO :name, :sex,
                          :birthdate:ind_birthdate;
    if( SQLCODE == SQLE_NOTFOUND ) {
        break; /* no more rows */
    } else if( SQLCODE < 0 ) {
        break; /* the FETCH caused an error */
    }
    if( ind_birthdate < 0 ) {
        strcpy( birthdate, "UNKNOWN" );
    }
    printf( "Name: %s Sex: %c Birthdate:
           %s\n", name, sex, birthdate );
}
/* 4. Close the cursor. */
EXEC SQL CLOSE C1;
}
```

For more information about the FETCH statement, see [“FETCH statement \[ESQL\] \[SP\]” \[SQL Anywhere Server - SQL Reference\]](#).

### Cursor positioning

A cursor is positioned in one of three places:

- On a row
- Before the first row
- After the last row

Absolute row from start		Absolute row from end
0	Before first row	$-n - 1$
1		$-n$
2		$-n + 1$
3		$-n + 2$
$n - 2$		$-3$
$n - 1$		$-2$
$n$		$-1$
$n + 1$	After last row	0

### Order of rows in a cursor

You control the order of rows in a cursor by including an `ORDER BY` clause in the `SELECT` statements that defines that cursor. If you omit this clause, the order of the rows is unpredictable.

If you don't explicitly define an order, the only guarantee is that fetching repeatedly will return each row in the result set once and only once before `SQL_NOTFOUND` is returned.

### Repositioning a cursor

When you open a cursor, it is positioned before the first row. The `FETCH` statement automatically advances the cursor position. An attempt to `FETCH` beyond the last row results in a `SQL_NOTFOUND` error, which can be used as a convenient signal to complete sequential processing of the rows.

You can also reposition the cursor to an absolute position relative to the start or end of the query results, or you can move the cursor relative to the current position. There are special *positioned* versions of the `UPDATE` and `DELETE` statements that can be used to update or delete the row at the current position of the cursor. If the cursor is positioned before the first row or after the last row, a `SQL_NOTFOUND` error is returned.

To avoid unpredictable results when using explicit positioning, you can include an `ORDER BY` clause in the `SELECT` statement that defines the cursor.

You can use the `PUT` statement to insert a row into a cursor.

## Cursor positioning after updates

After updating any information that is being accessed by an open cursor, it is best to fetch and display the rows again. If the cursor is being used to display a single row, `FETCH RELATIVE 0` will re-fetch the current row. When the current row has been deleted, the next row will be fetched from the cursor (or `SQLE_NOTFOUND` is returned if there are no more rows).

When a temporary table is used for the cursor, inserted rows in the underlying tables do not appear at all until that cursor is closed and reopened. It can be difficult to detect whether a temporary table is involved in a `SELECT` statement without examining the code generated by the SQL preprocessor or by becoming knowledgeable about the conditions under which temporary tables are used. Temporary tables can usually be avoided by having an index on the columns used in the `ORDER BY` clause.

For more information about temporary tables, see [“Use work tables in query processing \(use All-rows optimization goal\)”](#) [*SQL Anywhere Server - SQL Usage*].

Inserts, updates, and deletes to non-temporary tables may affect the cursor positioning. Because UltraLite materializes cursor rows one at a time (when temporary tables are not used), the data from a freshly inserted row (or the absence of data from a freshly deleted row) may affect subsequent `FETCH` operations. In the simple case where (parts of) rows are being selected from a single table, an inserted or updated row will appear in the result set for the cursor when it satisfies the selection criteria of the `SELECT` statement. Similarly, a freshly deleted row that previously contributed to the result set will no longer be within it.

## Authenticating users

UltraLite databases are created with a default user ID of `DBA` and default password of `sql`; you must first connect as this initial user. New users must be added from an existing connection.

A user ID cannot be changed; instead, you add the new user ID and then delete the existing user ID. A maximum of four user IDs are permitted for each UltraLite database.

### User authentication example

A complete sample can be found in the `samples-dir\UltraLite\esqlauth` directory. The code below is taken from `samples-dir\UltraLite\esqlauth\sample.sqc`.

```
//embedded SQL
app() {
    ...
    /* Declare fields */
    EXEC SQL BEGIN DECLARE SECTION;
        char uid[31];
        char pwd[31];
    EXEC SQL END DECLARE SECTION;
    db_init( &sqlca );
    ...
    EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";
    if( SQLCODE == SQLE_NOERROR ) {
        printf("Enter new user ID and password\n" );
        scanf( "%s %s", uid, pwd );
        ULGrantConnectTo( &sqlca,
```

```
        UL_TEXT( uid ), UL_TEXT( pwd ) );
    if( SQLCODE == SQLE_NOERROR ) {
        // new user added: remove DBA
        ULRevokeConnectFrom( &sqlca, UL_TEXT("DBA" ) );
    }
    EXEC SQL DISCONNECT;
}
// Prompt for password
printf("Enter user ID and password\n" );
scanf( "%s %s", uid, pwd );
EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;
```

The code carries out the following tasks:

1. Initiate database functionality by calling `db_init`.
2. Attempt to connect using the default user ID and password.
3. If the connection attempt is successful, add a new user.
4. If the new user is successfully added, delete the DBA user from the UltraLite database.
5. Disconnect. An updated user ID and password is now added to the database.
6. Connect using the updated user ID and password.

See:

- [“ULGrantConnectTo method” on page 296](#)
- [“ULRevokeConnectFrom method” on page 303](#)

## Encrypting data

You can encrypt or obfuscate your UltraLite database using the UltraLite embedded SQL.

See [“Encrypting data” on page 48](#).

### Encryption

When an UltraLite database is created (using Sybase Central for example), an optional encryption key may be specified. The encryption key is used to encrypt the database. Once the database is encrypted, all subsequent connection attempts must supply the encryption key. The supplied key is checked against the original encryption key and the connection fails unless the key matches.

Choose an encryption key value that cannot be easily guessed. The key can be of arbitrary length, but generally a longer key is better, because a shorter key is easier to guess. Including a combination of numbers, letters, and special characters decreases the chances of someone guessing the key.

Do not include semicolons in your key. Do not put the key itself in quotes, or the quotes will be considered part of the key.



### To connect to an encrypted UltraLite database

1. Specify the encryption key in the connection string used in the EXEC SQL CONNECT statement.

The encryption key is specified with the key= connection string parameter.

You must supply this key each time you want to connect to the database. Lost or forgotten keys result in completely inaccessible databases.

2. Handle attempts to open an encrypted database with the wrong key.

If an attempt is made to open an encrypted database and the wrong key is supplied, db\_init returns ul\_false and SQLCODE -840 is set.

### Changing the encryption key

You can change the encryption key for a database. The application must already be connected to the database using the existing key before the change can be made.

#### To change the encryption key on an UltraLite database

- Call the ULChangeEncryptionKey function, supplying the new key as an argument.

The application must already be connected to the database using the old key before this function is called.

See [“ULChangeEncryptionKey method” on page 271](#).

### Obfuscation

#### To obfuscate an UltraLite database

- An alternative to using database encryption is to specify that the database is to be obfuscated. Obfuscation is a simple masking of the data in the database that is intended to prevent browsing the data in the database with a low level file examination utility. Obfuscation is a database creation option and must be specified when the database is created.

See [“Choosing database creation parameters for UltraLite” \[UltraLite - Database Management and Reference\]](#).

## Adding synchronization to your application

Synchronization is a key feature of many UltraLite applications. This section describes how to add synchronization to your application.

The synchronization logic that keeps UltraLite applications up to date with the consolidated database is not held in the application itself. Synchronization scripts stored in the consolidated database, together with the MobiLink server and the UltraLite runtime library, control how changes are processed when they are uploaded and determines which changes are to be downloaded.

## Overview

The specifics of each synchronization are controlled by a set of synchronization parameters. These parameters are gathered into a structure, which is then supplied as an argument in a function call to synchronize. The outline of the method is the same in each development model.

### To add synchronization to your application

1. Initialize the structure that holds the synchronization parameters.

See [“Initializing the synchronization parameters” on page 50](#).

2. Assign the parameter values for your application.

See [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

3. Call the synchronization function, supplying the structure or object as argument.

See [“Invoking synchronization” on page 51](#).

You must ensure that there are no uncommitted changes when you synchronize.

## Synchronization parameters

The `ul_sync_info` structure is documented in the C/C++ component chapter; however, members of the structures are common to embedded SQL development as well. See [“ul\\_sync\\_info structure” on page 81](#).

For a general description of synchronization parameters, see [“Synchronization parameters for UltraLite” \[UltraLite - Database Management and Reference\]](#).

## Initializing the synchronization parameters

The synchronization parameters are stored in a structure.

The members of the structure undefined on initialization. You must set your parameters to their initial values with a call to a special function. The synchronization parameters are defined in a structure declared in the UltraLite header file `install-dir\SDK\Include\ulglobal.h`.

### To initialize the synchronization parameters (embedded SQL)

- Call the `ULInitSyncInfo` function. For example:

```
ul_sync_info synch_info;  
ULInitSyncInfo( &synch_info );
```

## Setting synchronization parameters

The following code initiates TCP/IP synchronization. The MobiLink user name is `Betty Best`, with password `TwentyFour`, the script version is `default`, and the MobiLink server is running on the host computer `test.internal`, on port 2439:

```
ul_sync_info synch_info;
ULInitSyncInfo( &synch_info );
synch_info.user_name = UL_TEXT("Betty Best");
synch_info.password = UL_TEXT("TwentyFour");
synch_info.version = UL_TEXT("default");
synch_info.stream = ULStream();
synch_info.stream_parms =
    UL_TEXT("host=test.internal;port=2439");
ULSynchronize( &sqlca, &synch_info );
```

## Invoking synchronization

The details of how to invoke synchronization depends on your target platform and on the synchronization stream.

The synchronization process can only work if the device running the UltraLite application is able to communicate with the MobiLink server. For some platforms, this means that the device needs to be physically connected by placing it in its cradle or by attaching it to a server computer using the appropriate cable. If the synchronization cannot be completed, add error handling code to your application.

### To invoke synchronization (TCP/IP, TLS, HTTP, or HTTPS streams)

- Call `ULInitSyncInfo` to initialize the synchronization parameters, and call `ULSynchronize` to synchronize.

The synchronization call requires a structure that holds a set of parameters describing the specifics of the synchronization. The particular parameters used depend on the stream.

## Commit all changes before synchronizing

An UltraLite database cannot have uncommitted changes when it is synchronized. If you attempt to synchronize an UltraLite database when any connection has an uncommitted transaction, the synchronization fails, an exception is thrown and the `SQLE_UNCOMMITTED_TRANSACTIONS` error is set. This error code also appears in the MobiLink server log.

For more information about download-only synchronizations, see [“Download Only synchronization parameter”](#) [*UltraLite - Database Management and Reference*].

## Adding initial data to your application

Many UltraLite applications need data to start working. You can download data into your application by synchronizing. You may want to add logic to your application to ensure that, the first time it is run, it downloads all necessary data before any other actions are carried out.

**Performance tip**

It is easier to locate errors if you develop an application in stages. When developing a prototype, temporarily use INSERT statements in your application to provide data for testing and demonstration purposes. Once your prototype is working correctly, replace the temporary INSERT statements with the code to perform the synchronization.

For more synchronization development tips, see [“MobiLink development tips” \[MobiLink - Server Administration\]](#).

## Handling synchronization communications errors

The following code illustrates how to handle communications errors from embedded SQL applications:

```
if( psqlca->sqlcode == SQLE_COMMUNICATIONS_ERROR ) {
    printf( " Stream error information:\n"
           "   stream_error_code = %ld\t(ss_error_code)\n"
           "   error_string      = \"%s\"\n"
           "   system_error_code = %ld\n",
           (long)info.stream_error.stream_error_code,
           info.stream_error.error_string,
           (long)info.stream_error.system_error_code );
}
```

SQLE\_COMMUNICATIONS\_ERROR is the general error code for communications errors.

To keep UltraLite small, the runtime reports numbers rather than messages.

## Monitoring and canceling synchronization

This section describes how to monitor and cancel synchronization from UltraLite applications.

### Monitoring synchronization

- Specify the name of your callback function in the observer member of the synchronization structure (ul\_synch\_info).
- Call the synchronization function or method to start synchronization.
- UltraLite calls your callback function whenever the synchronization state changes. The following section describes the synchronization state.

The following code shows how this sequence of tasks can be implemented in an embedded SQL application:

```
ULInitSyncInfo( &info );
info.user_name = m_EmpIDStr;
...
```

```
//The info parameter of ULSynchronize() contains
// a pointer to the observer function
info.observer = ObserverFunc;
ULSynchronize( &sqlca, &info );
```

## Handling synchronization status information

The callback function that monitors synchronization takes a `ul_sync_status` structure as parameter. For more information, see “[ul\\_sync\\_status structure](#)” on page 87.

The `ul_sync_status` structure has the following members:

```
struct ul_sync_status {
    struct {
        ul_u_long    bytes;
        ul_u_long    inserts;
        ul_u_long    updates;
        ul_u_long    deletes;
    }
    struct {
        ul_u_long    bytes;
        ul_u_long    inserts;
        ul_u_long    updates;
        ul_u_long    deletes;
    }
    received;
    p_ul_sync_info  info;
    ul_sync_state   state;
    ul_u_short      db_tableCount;
    ul_u_short      table_id;
    char            table_name[];
    ul_wchar        table_name_w2[];
    ul_u_short      sync_table_count;
    ul_u_short      sync_table_index;
    ul_sync_state   state;
    ul_bool         stop;
    ul_u_short      flags;
    ul_void *       user_data;
    SQLCA *         sqlca;
}
```

- **sent.inserts** The number of inserted rows that have been uploaded so far.
- **sent.updates** The number of updated rows that have been uploaded so far.
- **sent.deletes** The number of deleted rows that have been uploaded so far.
- **sent.bytes** The number of bytes that have been uploaded so far.
- **received.inserts** The number of inserted rows that have been downloaded so far.
- **received.updates** The number of updated rows that have been downloaded so far.
- **received.deletes** The number of deleted rows that have been downloaded so far.
- **received.bytes** The number of bytes that have been downloaded so far.

- **info** A pointer to the `ul_sync_info` structure. See [“ul\\_sync\\_info structure” on page 81](#).
- **db\_tableCount** Returns the number of tables in the database.
- **table\_id** The current table number (relative to 1) that is being uploaded or downloaded. This number may skip values when not all tables are being synchronized and is not necessarily increasing.
- **table\_name[]** Name of the current table.
- **table\_name\_w2[]** Name of the current table (wide character version). This field is only populated in the Windows (desktop and Mobile) environment.
- **sync\_table\_count** Returns the number of tables being synchronized.
- **sync\_table\_index** The number of the table that is being uploaded or downloaded, starting at 1 and ending at the `sync_table_count` value. This number may skip values when not all tables are being synchronized.
- **state** One of the following states:
  - **UL\_SYNC\_STATE\_STARTING** No synchronization actions have yet been taken.
  - **UL\_SYNC\_STATE\_CONNECTING** The synchronization stream has been built, but not yet opened.
  - **UL\_SYNC\_STATE\_SENDING\_HEADER** The synchronization stream has been opened, and the header is about to be sent.
  - **UL\_SYNC\_STATE\_SENDING\_TABLE** A table is being sent.
  - **UL\_SYNC\_STATE\_SENDING\_DATA** Schema information or data is being sent.
  - **UL\_SYNC\_STATE\_FINISHING\_UPLOAD** The upload stage is completed and a commit is being carried out.
  - **UL\_SYNC\_STATE\_RECEIVING\_UPLOAD\_ACK** An acknowledgement that the upload is complete is being received.
  - **UL\_SYNC\_STATE\_RECEIVING\_TABLE** A table is being received.
  - **UL\_SYNC\_STATE\_RECEIVING\_DATA** Schema information or data is being received.
  - **UL\_SYNC\_STATE\_COMMITTING\_DOWNLOAD** The download stage is completed and a commit is being carried out.
  - **UL\_SYNC\_STATE\_SENDING\_DOWNLOAD\_ACK** An acknowledgement that download is complete is being sent.
  - **UL\_SYNC\_STATE\_DISCONNECTING** The synchronization stream is about to be closed.
  - **UL\_SYNC\_STATE\_DONE** Synchronization has completed successfully.

- **UL\_SYNC\_STATE\_ERROR** Synchronization has completed, but with an error.
- **UL\_SYNC\_STATE\_ROLLING\_BACK\_DOWNLOAD** An error occurred during download and the download is being rolled back.

For more information about the synchronization process, see [“The synchronization process” \[MobiLink - Getting Started\]](#).

- **stop** Set this member to true to interrupt the synchronization. The SQL exception `SQLE_INTERRUPTED` is set, and the synchronization stops as if a communications error had occurred. The observer is *always* called with either the `DONE` or `ERROR` state so that it can do proper cleanup.
- **flags** Returns the current synchronization flags indicating additional information related to the current state.
- **user\_data** Returns the user data object that is passed as an argument to the `ULSetSynchronizationCallback` function.
- **sqlca** Pointer to the connection's active `SQLCA`.

## Example

The following code illustrates a very simple observer function:

```
extern void __stdcall ObserverFunc(
    p_ul_sync_status status )
{
    switch( status->state ) {
        case UL_SYNC_STATE_STARTING:
            printf( "Starting\n" );
            break;
        case UL_SYNC_STATE_CONNECTING:
            printf( "Connecting\n" );
            break;
        case UL_SYNC_STATE_SENDING_HEADER:
            printf( "Sending Header\n" );
            break;
        case UL_SYNC_STATE_SENDING_TABLE:
            printf( "Sending Table %d of %d\n",
                status->tableIndex + 1,
                status->tableCount );
            break;
        case UL_SYNC_STATE_RECEIVING_UPLOAD_ACK:
            printf( "Receiving Upload Ack\n" );
            break;
        case UL_SYNC_STATE_RECEIVING_TABLE:
            printf( "Receiving Table %d of %d\n",
                status->tableIndex + 1,
                status->tableCount );
            break;
        case UL_SYNC_STATE_SENDING_DOWNLOAD_ACK:
            printf( "Sending Download Ack\n" );
            break;
        case UL_SYNC_STATE_DISCONNECTING:
            printf( "Disconnecting\n" );
            break;
        case UL_SYNC_STATE_DONE:
            break;
    }
}
```

```
        printf( "Done\n" );
        break;
    break;
    ...
```

This observer produces the following output when synchronizing two tables:

```
Starting
Connecting
Sending Header
Sending Table 1 of 2
Sending Table 2 of 2
Receiving Upload Ack
Receiving Table 1 of 2
Receiving Table 2 of 2
Sending Download Ack
Disconnecting
Done
```

### CustDB example

An example of an observer function is included in the CustDB sample application. The implementation in CustDB provides a window that displays synchronization progress and allows the user to cancel synchronization. The user-interface component makes the observer function platform specific.

The CustDB sample code is in the *samples-dir\UltraLite\CustDB* directory. The observer function is contained in platform-specific subdirectories of the *CustDB* directory.

## Building embedded SQL applications

This section describes a general build procedure for UltraLite embedded SQL applications.

This section assumes a familiarity with the overall embedded SQL development model.

## General build procedure

### Sample code

You can find a makefile that uses this process in the *samples-dir\UltraLite\ESQLSecurity* directory.

#### Separately licensed component required

ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See “Separately licensed components” [[SQL Anywhere 12 - Introduction](#)].



## Procedure

### To build an UltraLite embedded SQL application

1. Run the SQL preprocessor on *each* embedded SQL source file.

The SQL preprocessor is the `sqlpp` command line utility. It preprocesses the embedded SQL source files, producing C++ source files to be compiled into your application.

For more information about the SQL preprocessor, see [“SQL preprocessor for UltraLite utility \(sqlpp\)”](#) [*UltraLite - Database Management and Reference*].

#### Caution

`sqlpp` overwrites the output file without regard to its contents. Ensure that the output file name does not match the name of any of your source files. By default, `sqlpp` constructs the output file name by changing the suffix of your source file to `.cpp`. When in doubt, specify the output file name explicitly, following the name of the source file.

2. Compile *each* C++ source file for the target platform of your choice. Include:
  - each C++ file generated by the SQL preprocessor
  - any additional C or C++ source files required by your application
3. Link *all* these object files, together with the UltraLite runtime library.

## Configuring development tools for embedded SQL development

Many development tools use a dependency model, sometimes expressed as a makefile, in which the timestamp on each source file is compared with that on the target file (usually the object file) to decide whether the target file needs to be regenerated.

With UltraLite development, a change to any SQL statement in a development project means that the generated code needs to be regenerated. Changes are not reflected in the timestamp on any individual source file because the SQL statements are stored in the reference database.

This section describes how to incorporate UltraLite application development, specifically the SQL preprocessor, into a dependency-based build environment. The specific instructions provided are for Visual C++, and you may need to modify them for your own development tool.

### SQL preprocessing

The first set of instructions describes how to add instructions to run the SQL preprocessor to your development tool.

#### To add embedded SQL preprocessing into a dependency-based development tool

1. Add the `.sql` files to your development project.

The development project is defined in your development tool.

2. Add a custom build rule for each *.sql* file.

- The custom build rule should run the SQL preprocessor. In Visual C++, the build rule should have the following command (entered on a single line):

```
"%SQLANY12%\Bin32\sqlpp.exe" -q -u $(InputPath) $(InputName).cpp
```

where SQLANY12 is an environment variable that points to your SQL Anywhere installation directory.

For a full description of the SQL preprocessor command line, see [“SQL preprocessor for UltraLite utility \(sqlpp\)” \[UltraLite - Database Management and Reference\]](#).

- Set the output for the command to **\$(InputName).cpp**.

3. Compile the *.sql* files, and add the generated *.cpp* files to your development project.

You need to add the generated files to your project even though they are not source files, so that you can set up dependencies and build options.

4. For each generated *.cpp* file, set the preprocessor definitions.

- Under General or Preprocessor, add UL\_USE\_DLL to the Preprocessor definitions.
- Under Preprocessor, add  $$(SQLANY12)\SDK\Include$  and any other include folders you require to your include path, as a comma-separated list.

---

# Developing UltraLite applications for Windows Mobile

Microsoft Visual Studio 2005 and later can be used to develop applications for the Windows Mobile environment.

Applications targeting Windows Mobile should use the default setting for `wchar_t` and link against the UltraLite runtime libraries in `install-dir\ultralite\ce\arm.50\lib\`.

For a list of supported host platforms and development tools for Windows Mobile development, and for a list of supported target Windows Mobile platforms, see [“Supported platforms” \[SQL Anywhere 12 - Introduction\]](#).

You can test your applications under an emulator on most Windows Mobile target platforms.

## Choosing how to link the runtime library

Windows Mobile supports dynamic link libraries. At link time, you have the option of linking your UltraLite application to the runtime DLL using an imports library, or statically linking your application using the UltraLite runtime library.

### Performance tip

If you have a single UltraLite application on your target device, a statically linked library uses less memory. If you have multiple UltraLite applications on your target device, using the DLL may be more economical in memory use.

If you are repeatedly downloading UltraLite applications to a device, over a slow link, then you may want to use the DLL to minimize the size of the downloaded executable, after the initial download.

### To build and deploy an application using the UltraLite runtime DLL

1. Preprocess your code, then compile the output with `UL_USE_DLL`.
2. Link your application using the UltraLite imports library.
3. Copy both your application executable and the UltraLite runtime DLL to your target device.

## Building the CustDB sample application

CustDB is a simple sales-status application. It is located in the `samples-dir\UltraLite\` directory of your SQL Anywhere installation.

The CustDB application is provided as a Visual Studio solution.

For a diagram of the sample database schema, see “[The CustDB sample database application](#)” [[SQL Anywhere 12 - Introduction](#)].

### To build the CustDB sample application

1. Start Visual Studio.
2. Open the project file that corresponds to your version of Visual Studio.
  - *samples-dir\UltraLite\CustDB\VS8* for Visual Studio 2005.
  - *samples-dir\UltraLite\CustDB\VS9* for Visual Studio 2008.
  - *samples-dir\UltraLite\CustDB\VS10* for Visual Studio 2010.
3. Choose **Build » Set Configuration Manager** to set the target platform.
  - Set an active solution platform of your choice.
4. Build the application:
  - Choose **Build » Deploy Solution** to build and deploy CustDB.  
When the application is built it will be uploaded automatically to the remote device.
5. Start the MobiLink server:
  - To start the MobiLink server, from the **Start** menu choose **Programs » SQL Anywhere 12 » MobiLink » Synchronization Server Sample**.
6. Run the CustDB application:

Before running the CustDB application, the custdb database must be copied to the root folder of the device. Copy the database file named *samples-dir\UltraLite\CustDB\custdb.udb* to the root of the device.

Execute *\Program Files\CustDBC#\CustDB.exe* on the device or simulator.

#### Folder locations and environment variables

The sample project uses environment variables wherever possible. It may be necessary to adjust the project for the application to build properly. If you experience problems, try searching for missing files in the Microsoft Visual C++ folder(s) and adding the appropriate directory settings.

For embedded SQL, the build process uses the SQL preprocessor, *sqlpp*, to preprocess the file *CustDB.sqc* into the file *CustDB.cpp*. This one-step process is useful in smaller UltraLite applications where all the embedded SQL can be confined to one source module. In larger UltraLite applications, you need to use multiple *sqlpp* invocations.

See “[Building embedded SQL applications](#)” on page 56.

## Storing persistent data

The UltraLite database is stored in the Windows Mobile file system. The default file is `\UltraLiteDB\ul_store.udb`. You can override this choice using the **file\_name** connection parameter which specifies the full path name of the file-based persistent store.

The UltraLite runtime carries out no substitutions on the **file\_name** parameter. If a directory has to be created for the file name to be valid, the application must ensure that any directories are created before calling **db\_init**.

As an example, you could make use of a flash memory storage card by scanning for storage cards and prefixing a name by the appropriate directory name for the storage card. For example,

```
file_name = "\\Storage Card\\My Documents\\flash.udb"
```

## Deploying Windows Mobile applications

When compiling UltraLite applications for Windows Mobile, you can link the UltraLite runtime library either statically or dynamically. If you link it dynamically, you must copy the UltraLite runtime library for your platform to the target device.

### To build and deploy an application using the UltraLite runtime DLL

1. Preprocess your code, then compile the output with `UL_USE_DLL`.
2. Link your application using the UltraLite imports library.
3. Copy both your application executable and the UltraLite runtime DLL to your target device.

The UltraLite runtime DLL is in chip-specific directories under the `\ultralite\ce` subdirectory of your SQL Anywhere installation directory.

To deploy the UltraLite runtime DLL for the Windows Mobile emulator, connect to the emulator using ActiveSync and copy the DLL to the device using Windows Explorer.

## Deploying applications that use ActiveSync

Applications that use ActiveSync synchronization must be registered with ActiveSync and copied to the device. See [“Register applications with the ActiveSync Manager”](#) [*UltraLite - Database Management and Reference*].

The MobiLink provider for ActiveSync must also be installed. See [“Deploy the ActiveSync provider for UltraLite”](#) [*UltraLite - Database Management and Reference*].

## Assigning class names for applications

When registering applications for use with ActiveSync you must supply a window class name. Assigning class names is carried out at development time and your application development tool documentation is the primary source of information on the topic.

Microsoft Foundation Classes (MFC) dialog boxes are given a generic class name of **Dialog**, which is shared by all dialogs in the system. This section describes how to assign a distinct class name for your application if you are using MFC.

### To assign a window class name for MFC applications

1. Create and register a custom window class for dialog boxes, based on the default class.

Add the following code to your application's startup code. The code must be executed before any dialogs get created:

```
WNDCLASS wc;
if( ! GetClassInfo( NULL, L"Dialog", &wc ) ) {
    AfxMessageBox( L"Error getting class info" );
}
wc.lpszClassName = L"MY_APP_CLASS";
if( ! AfxRegisterClass( &wc ) ) {
    AfxMessageBox( L"Error registering class" );
}
```

where *MY\_APP\_CLASS* is the unique class name for your application.

2. Determine which dialog is the main dialog for your application.

If your project was created with the MFC Application Wizard, this is likely to be a dialog named **MyAppDlg**.

3. Find and record the resource ID for the main dialog.

The resource ID is a constant of the same general form as `IDD_MYAPP_DIALOG`.

4. Ensure that the main dialog remains open any time your application is running.

Add the following line to your application's **InitInstance** function. The line ensures that if the main dialog **dlg** is closed, the application also closes.

```
m_pMainWnd = &dlg;
```

For more information, see the Microsoft documentation for **CWinThread::m\_pMainWnd**.

If the dialog does not remain open for the duration of your application, you must change the window class of other dialogs as well.

5. Save your changes.
6. Modify the resource file for your project.
  - Open your resource file (which has an extension of *.rc*) in a text editor such as Notepad.

Locate the resource ID of your main dialog.

- Change the main dialog's definition to use the new window class as in the following example. The *only* change that you should make is the addition of the **CLASS** line:

```

IDD_MYAPP_DIALOG DIALOG DISCARDABLE 0, 0, 139, 103
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION
EXSTYLE WS_EX_APPWINDOW | WS_EX_CAPTIONOKBTN
CAPTION "MyApp"
FONT 8, "System"
CLASS "MY_APP_CLASS"
BEGIN
    LTEXT "TODO: Place dialog controls here.", IDC_STATIC,
    13, 33, 112, 17
END

```

where *MY\_APP\_CLASS* is the name of the window class you used earlier.

- Save the *.rc* file.
7. Add code to catch the synchronization message.

See [“Adding ActiveSync synchronization \(MFC\)”](#) on page 64.

## Synchronization on Windows Mobile

UltraLite applications on Windows Mobile can synchronize through the following stream types:

- **ActiveSync** See [“Adding ActiveSync synchronization to your application”](#) on page 63.
- **TCP/IP** See [“TCP/IP, HTTP, or HTTPS synchronization from Windows Mobile”](#) on page 66.
- **HTTP** See [“TCP/IP, HTTP, or HTTPS synchronization from Windows Mobile”](#) on page 66.

The *user\_name* and *stream\_parms* parameters must be surrounded by the **UL\_TEXT()** macro for Windows Mobile when initializing, since the compilation environment is Unicode wide characters.

For more information about synchronization parameters, see [“Synchronization parameters for UltraLite”](#) [[UltraLite - Database Management and Reference](#)].

## Adding ActiveSync synchronization to your application

ActiveSync is software from Microsoft that handles data synchronization between a desktop computer running Windows and a connected Windows Mobile handheld device. UltraLite supports ActiveSync versions 3.5 and later.

This section describes how to add ActiveSync provider to your application, and how to register your application for use with ActiveSync on your end users' computers.

If you use ActiveSync, synchronization can be initiated only by ActiveSync itself. ActiveSync automatically initiates a synchronization when the device is placed in the cradle or when **Synchronize** is

selected from the ActiveSync window. The MobiLink provider starts the application, if it is not already running, and sends a message to the application.

For more information about setting up ActiveSync synchronization, see [“Deploying applications that use ActiveSync” on page 61](#).

The ActiveSync provider uses the **wParam** parameter. A **wParam** value of 1 indicates that the MobiLink provider for ActiveSync launched the application. The application must then shut itself down after it has finished synchronizing. If the application was already running when called by the MobiLink provider for ActiveSync, **wParam** is 0. The application can ignore the **wParam** parameter if it wants to keep running.

To determine which platforms the provider is supported on, see [SQL Anywhere Components by Platform](#).

Adding synchronization depends on whether you are addressing the Windows API directly or whether you are using the Microsoft Foundation Classes. Both development models are described here.

## Adding ActiveSync synchronization (Windows API)

If you are programming directly to the Windows API, you must handle the message from the MobiLink provider in your application's **WindowProc** function, using the **ULIsSynchronizeMessage** function to determine if it has received the message.

Here is an example of how to handle the message:

```
LRESULT CALLBACK WindowProc( HWND hwnd,
                             UINT uMsg,
                             WPARAM wParam,
                             LPARAM lParam )
{
    if( ULIsSynchronizeMessage( uMsg ) ) {
        DoSync();
        if( wParam == 1 ) DestroyWindow( hwnd );
        return 0;
    }
    switch( uMsg ) {
        // code to handle other windows messages
        default:
            return DefWindowProc( hwnd, uMsg, wParam, lParam );
    }
    return 0;
}
```

where **DoSync** is the method that actually calls **ULSynchronize**.

See [“ULIsSynchronizeMessage method” on page 298](#).

## Adding ActiveSync synchronization (MFC)

If you are using Microsoft Foundation Classes to develop your application, you can catch the synchronization message in the main dialog class or in your application class. Both methods are described here.



Your application must create and register a custom window class name for notification. See [“Assigning class names for applications” on page 61](#).

### To add ActiveSync synchronization in the main dialog class

1. Add a registered message and declare a message handler.

Find the message map in the source file for your main dialog (the name is of the same form as *CMyAppDlg.cpp*). Add a registered message using the **static** and declare a message handler using `ON_REGISTERED_MESSAGE` as in the following example:

```
static UINT WM_ULTRALITE_SYNC_MESSAGE =
    ::RegisterWindowMessage( UL_AS_SYNCHRONIZE );
BEGIN_MESSAGE_MAP(CMyAppDlg, CDialog)
    //{{AFX_MSG_MAP(CMyAppDlg)
    //}}AFX_MSG_MAP
    ON_REGISTERED_MESSAGE( WM_ULTRALITE_SYNC_MESSAGE,
        OnDoUltraLiteSync )
END_MESSAGE_MAP()
```

2. Implement the message handler.

Add a method to the main dialog class with the following signature. This method is automatically executed any time the MobiLink provider for ActiveSync requests that your application synchronize. The method should call **ULSynchronize**.

```
LRESULT CMyAppDlg::OnDoUltraLiteSync(
    WPARAM wParam,
    LPARAM lParam
);
```

The return value of this function should be 0.

For more information about handling the synchronization message, see [“ULIsSynchronizeMessage method” on page 298](#).

### To add ActiveSync synchronization in the Application class

1. Open the **Class Wizard** for the application class.
2. In the **Messages** list, highlight **PreTranslateMessage** and then click **Add Function**.
3. Click **Edit Code**. The `PreTranslateMessage` function appears. Change it to read as follows:

```
BOOL CMyApp::PreTranslateMessage(MSG* pMsg)
{
    if( ULIsSynchronizeMessage(pMsg->message) ) {
        DoSync();
        // close application if launched by provider
        if( pMsg->wParam == 1 ) {
            ASSERT( AfxGetMainWnd() != NULL );
            AfxGetMainWnd()->SendMessage( WM_CLOSE );
        }
        return TRUE; // message has been processed
    }
    return CWinApp::PreTranslateMessage(pMsg);
}
```

where **DoSync** is the function that actually calls `ULSynchronize`.

For more information about handling the synchronization message, see [“ULIsSynchronizeMessage method” on page 298](#).

## **TCP/IP, HTTP, or HTTPS synchronization from Windows Mobile**

For TCP/IP, HTTP, or HTTPS synchronization, the application controls when synchronization occurs. Your application should provide a menu item or user interface control so that the user can request synchronization.

# API reference

This section provides the UltraLite C/C++ API.



---

# UltraLite C/C++ common API reference

This section lists functions and macros that you can use with either the embedded SQL or C++ interface. Most of the functions in this section require a SQL Communications Area.

## Header file

- `ulglobal.h`

## Macros and compiler directives for UltraLite C/C++ applications

Unless otherwise stated otherwise, directives apply to both embedded SQL and C++ API applications.

You can supply compiler directives:

- On your compiler command line. You commonly set a directive with the `/D` option. For example, to compile an UltraLite application with user authentication, a makefile for the Microsoft Visual C++ compiler may look as follows:

```
CompileOptions=/c /DPRWIN32 /Od /Zi /DWIN32
/DUL_USE_DLL

IncludeFolders= \
/I"$(VCDIR)\include" \
/I"$(SQLANY12)\SDK\Include"

sample.obj: sample.cpp
cl $(CompileOptions) $(IncludeFolders) sample.cpp
```

`VCDIR` is your Visual C++ directory and `SQLANY12` is your SQL Anywhere installation directory.

- In the compiler settings window of your user interface.
- In source code. You supply directives with the `#define` statement.

## UL\_TEXT macro

Prepares constant strings to be compiled as single-byte strings or wide-character strings. Use this macro to enclose all constant strings if you plan to compile the application to use Unicode and non-Unicode representations of strings. This macro properly defines strings in all environments and platforms.

## UL\_USE\_DLL macro

Sets the application to use the runtime library DLL, rather than a static runtime library.

**Remarks**

Applies to Windows Mobile and Windows applications.

## UNDER\_CE macro

By default, this macro is defined in all new Visual C++ Smart Device projects.

**Remarks**

Applies to Windows Mobile applications.

**See also**

- [“Developing UltraLite applications for Windows Mobile” on page 59](#)

**Example**

```
/D UNDER_CE
```

## UL\_RS\_STATE enumeration

Specifies possible result set or cursor states.

**Syntax**

```
public enum UL_RS_STATE
```

**Members**

Member name	Description
UL_RS_STATE_ERROR	Error.
UL_RS_STATE_UNPREPARED	Not prepared.
UL_RS_STATE_ON_ROW	On a valid row.
UL_RS_STATE_BEFORE_FIRST	Before the first row.
UL_RS_STATE_AFTER_LAST	After the last row.
UL_RS_STATE_COMPLETED	Closed.

## ul\_column\_sql\_type enumeration

Represents the SQL types for a column.

**Syntax**

```
public enum ul_column_sql_type
```

**Members**

Member name	Description
UL_SQLTYPE_BAD_INDEX	The column at the specified index does not exist.
UL_SQLTYPE_S_LONG	The column contains a signed long.
UL_SQLTYPE_U_LONG	The column contains an unsigned long.
UL_SQLTYPE_S_SHORT	The column contains a signed short.
UL_SQLTYPE_U_SHORT	The column contains an unsigned short.
UL_SQLTYPE_S_BIG	The column contains a signed 64-bit integer.
UL_SQLTYPE_U_BIG	The column contains an unsigned 64-bit integer.
UL_SQLTYPE_TINY	The column contains an unsigned 8-bit integer.
UL_SQLTYPE_BIT	The column contains a 1-bit flag.
UL_SQLTYPE_TIMESTAMP	The column contains timestamp information.
UL_SQLTYPE_DATE	The column contains date information.
UL_SQLTYPE_TIME	The column contains time information.
UL_SQLTYPE_DOUBLE	The column contains a double precision floating-point number. (8 bytes).
UL_SQLTYPE_REAL	The column contains a single precision floating-point number. (4 bytes).
UL_SQLTYPE_NUMERIC	The column contains exact numerical data, with specified precision and scale.
UL_SQLTYPE_BINARY	The column contains binary data with a specified maximum length.
UL_SQLTYPE_CHAR	The column contains character data with a specified length.
UL_SQLTYPE_LONGVARCHAR	The column contains character data with variable length.
UL_SQLTYPE_LONGBINARY	The column contains binary data with variable length.
UL_SQLTYPE_UUID	The column contains a UUID.

Member name	Description
UL_SQLTYPE_ST_GEOMETRY	The column contains spatial data in the form of points.
UL_SQLTYPE_TIME-STAMP_WITH_TIME_ZONE	The column contains timestamp and time zone information.

**Remarks**

These values correspond to SQL column types.

## ul\_column\_storage\_type enumeration

Represents the host variable types for a column.

**Syntax**

```
public enum ul_column_storage_type
```

**Members**

Member name	Description
UL_TYPE_BAD_INDEX	Invalid value.
UL_TYPE_S_LONG	ul_s_long (32 bit signed int)
UL_TYPE_U_LONG	ul_u_long (32 bit unsigned int)
UL_TYPE_S_SHORT	ul_s_short (16 bit signed int)
UL_TYPE_U_SHORT	ul_u_short (16 bit unsigned int)
UL_TYPE_S_BIG	ul_s_big (64 bit signed int)
UL_TYPE_U_BIG	ul_u_big (64 bit unsigned int)
UL_TYPE_TINY	ul_byte (8 bit unsigned)
UL_TYPE_BIT	ul_byte (8 bit unsigned, 1 bit used)
UL_TYPE_DOUBLE	ul_double (double)
UL_TYPE_REAL	ul_real (float)
UL_TYPE_BINARY	ul_binary (2 byte length followed by byte array)
UL_TYPE_TIMESTAMP_STRUCT	DECL_DATETIME.



Member name	Description
UL_TYPE_TCHAR	character array (string buffer)
UL_TYPE_CHAR	char array (string buffer)
UL_TYPE_WCHAR	ul_wchar (UTF16) array
UL_TYPE_GUID	GUID structure.

**Remarks**

These values are used to identify the host variable type required for a column, and to indicate how UltraLite should fetch values.

## ul\_database\_option\_id enumeration

Specifies possible database options that users can set.

**Syntax**

```
public enum ul_database_option_id
```

**Members**

Member name	Description
ul_option_global_database_id	The global database ID is set using an unsigned long.
ul_option_ml_remote_id	The remote ID is set using a string.
ul_option_commit_flush_timeout	The database commit flush timeout is set as an integer, representing a time threshold measured in milliseconds.
ul_option_commit_flush_count	The database commit flush count is set as integer, representing a commit count threshold.
ul_option_isolation_level	The connection isolation level is set as string. (read_committed/read_uncommitted)

**Remarks**

These database options are used with the SetDatabaseOption method. See [“SetDatabaseOption method” on page 129](#).

## ul\_database\_property\_id enumeration

Specifies possible database properties that users can retrieve.

### Syntax

```
public enum ul_database_property_id
```

### Members

Member name	Description
ul_property_date_format	Date format. (date_format)
ul_property_date_order	Date order. (date_order)
ul_property_nearest_century	Nearest century. (nearest_century)
ul_property_precision	Precision. (precision)
ul_property_scale	Scale. (scale)
ul_property_time_format	Time format. (time_format)
ul_property_timestamp_format	Timestamp format. (timestamp_format)
ul_property_timestamp_increment	Timestamp increment. (timestamp_increment)
ul_property_name	Name. (Name)

<b>Member name</b>	<b>Description</b>
ul_property_file	File. (File)
ul_property_encryption	Encryption. (Encryption)
ul_property_global_database_id	Global database ID. (global_database_id)
ul_property_ml_remote_id	Remote ID. (ml_remote_id)
ul_property_char_set	Character set. (CharSet)
ul_property_collation	collation sequence. (Collation)
ul_property_page_size	Page size. (PageSize)
ul_property_case_sensitive	CaseSensitive. (CaseSensitive)
ul_property_conn_count	Connection count. (ConnCount)
ul_property_max_hash_size	Default maximum index hash. (MaxHashSize)
ul_property_checksum_level	Database checksum level. (ChecksumLevel)
ul_property_checkpoint_count	Database checkpoint count. (CheckpointCount)

Member name	Description
ul_property_commit_flush_timeout	Database commit flush timeout. (commit_flush_timeout)
ul_property_commit_flush_count	Database commit flush count. (commit_flush_count)
ul_property_isolation_level	Connection isolation level. (isolation_level)
ul_property_timestamp_with_time_zone_format	Timestamp with time zone format. (timestamp_with_time_zone_format)

**Remarks**

These properties are used with the `ULConnection::GetDatabaseProperty` method. See [“GetDatabaseProperty method” on page 110](#).

## ul\_error\_action enumeration

Specifies possible error actions returned from callback. Not all actions apply to all error codes.

**Syntax**

```
public enum ul_error_action
```

**Members**

Member name	Description
UL_ERROR_ACTION_DEFAULT	Behave as if there is no error callback.
UL_ERROR_ACTION_CANCEL	Cancel the operation that raised the error.
UL_ERROR_ACTION_TRY_AGAIN	Retry the operation that raised the error.
UL_ERROR_ACTION_CONTINUE	Continue execution, ignoring the operation that raised the error.

## ul\_sync\_state enumeration

Indicates the current stage of synchronization.

### Syntax

```
public enum ul_sync_state
```

### Members

Member name	Description
UL_SYNC_STATE_STARTING	The synchronization is starting; initial parameter validation is complete and synchronization result will be saved.
UL_SYNC_STATE_CONNECTING	Connecting to the MobiLink server.
UL_SYNC_STATE_SENDING_HEADER	The synchronization connection is established and initial data is about to be sent.
UL_SYNC_STATE_SENDING_TABLE	A table is about to be sent.
UL_SYNC_STATE_SENDING_DATA	Schema information or row data is being sent.
UL_SYNC_STATE_FINISHING_UPLOAD	The upload stage is complete and state information is about to be committed.
UL_SYNC_STATE_RECEIVING_UPLOAD_ACK	About to read data from the server, starting with the upload acknowledgement.
UL_SYNC_STATE_RECEIVING_TABLE	A table is about to be received.
UL_SYNC_STATE_RECEIVING_DATA	Data for the most recently identified table is being received.
UL_SYNC_STATE_COMMITTING_DOWNLOAD	The download stage is complete and downloaded rows are about to be committed.
UL_SYNC_STATE_ROLLING_BACK_DOWNLOAD	An error occurred during download and the download is being rolled back.
UL_SYNC_STATE_SENDING_DOWNLOAD_ACK	An acknowledgement that the download is complete is being sent.
UL_SYNC_STATE_DISCONNECTING	About to disconnect from the server.
UL_SYNC_STATE_DONE	Synchronization has completed successfully.

Member name	Description
UL_SYNC_STATE_ERROR	Synchronization has completed, but with an error.

**Remarks**

You should not assume that the synchronization states occur in the order listed below.

## ul\_validate\_status\_id enumeration

Specifies possible status IDs for the UltraLite validation tool.

**Syntax**

```
public enum ul_validate_status_id
```

**Members**

Member name	Description	Value
UL_VALID_NO_ERROR	No error occurred.	0
UL_VALID_START	Start validation.	1
UL_VALID_END	End validation. Parm1 tracks the resulting sqlcode, which indicates success or failure.	2
UL_VALID_CHECKING_PAGE	Send a periodic status message while checking database pages. Parm1 tracks a number associated with the page. The order is not defined.	10
UL_VALID_CHECKING_TABLE	Checking a table. Parm1 tracks the table name.	20
UL_VALID_CHECKING_INDEX	Checking an index. Parm1 stores the table name and parm2 stores the index name.	21

Member name	Description	Value
UL_VALID_HASH_REPORT	Reporting on the index hash use.  (development version only) Parm1 tracks the table name, parm2 tracks the index name, parm3 tracks the number of visible rows, parm4 tracks the number of unique hash values, and parm5 tracks the maximum number of times a hash entry appears.	30
UL_VALID_REDUNDANT_INDEX	A redundant index was found.  (development version only) Parm1 tracks the table name, parm2 tracks the redundant index name, and parm3 tracks the name of index that makes it redundant.	31
UL_VALID_DUPLICATE_INDEX	Two indexes are the same.  (development version only) Parm1 tracks the table name, parm2 tracks the name of first index, and parm3 tracks the name of second index.	32
UL_VALID_DATABASE_ERROR	An error occurred accessing the database.  Check the SQLCODE for more information.	100
UL_VALID_STARTUP_ERROR	Error starting the database.  (for low-level access)	101
UL_VALID_CONNECT_ERROR	Error connecting to the database.	102
UL_VALID_INTERRUPTED	Validation process interrupted.	103
UL_VALID_CORRUPT_PAGE_TABLE	Page table is corrupt.	110
UL_VALID_ID_FAILED_CHECKSUM	Page checksum failed.  Parm1 tracks a number associated with the page.	111
UL_VALID_CORRUPT_PAGE	A page is corrupt.  Parm1 tracks a number associated with the page.	112

Member name	Description	Value
UL_VALID_ROW-COUNT_MISMATCH	The number of rows in the index is different from the table row count.  Parm1 tracks the table name, and parm2 tracks index name.	120
UL_VAL-ID_BAD_ROWID	There is an invalid row identifier in the index.  Parm1 tracks the table name, and parm2 tracks the index name.	121

## ul\_binary structure

Used to set and fetch binary values from a table in the database.

### Syntax

```
public struct ul_binary
```

### Members

Member name	Type	Description
data	ul_byte	The actual data to be set (for insert) or that was fetched (for select).
len	ul_length	The number of bytes in the value.

## ul\_error\_info structure

Stores complete information about an UltraLite error.

### Syntax

```
public struct ul_error_info
```

### Members

Member name	Type	Description
sqlcode	an_sql_code	The SQLCODE value.
sqlcount	ul_s_long	The SQLCOUNT value.

### Remarks

For more information about UltraLite errors, see [“ULErrorInfoString method” on page 286](#), [“ULErrorInfoURL method” on page 287](#), [“ULErrorInfoInitFromSqlca method” on page 285](#),



“ULErrorInfoParameterCount method” on page 286, and “ULErrorInfoParameterAt method” on page 286.

## ul\_stream\_error structure

Stores synchronization communication stream error information.

### Syntax

```
public struct ul_stream_error
```

### Members

Member name	Type	Description
error_string	char	An optional parameter string associated with the error. The meaning depends on the specific error.
stream_error_code	ss_error_code	The specific stream error. See the ss_error_code enumeration for possible values.
system_error_code	asa_int32	A system-specific error code. For more information about error codes, see your platform documentation.

## ul\_sync\_info structure

Stores synchronization data.

### Syntax

```
public struct ul_sync_info
```

### Members

Member name	Type	Description
additional_parms	const char *	A string of name value pairs "name=value;" with extra parameters.
auth_parms	const char **	An array of authentication parameters in MobiLink events.
auth_status	ul_auth_status	The status of MobiLink user authentication. The MobiLink server provides this information to the client.

Member name	Type	Description
auth_value	ul_s_long	The results of a custom MobiLink user authentication script. The MobiLink server provides this information to the client to determine the authentication status.
download_only	ul_bool	Do not upload any changes from the UltraLite database during the current synchronization.
ignored_rows	ul_bool	The status of ignored rows. This read-only field reports true if any rows were ignored by the MobiLink server during synchronization because of absent scripts.
init_verify	ul_sync_info *	Initialize verification.
keep_partial_download	ul_bool	When a download fails because of a communications error during synchronization, this parameter controls whether UltraLite holds on to the partial download rather than rolling back the changes.
new_password	const char *	A string specifying a new MobiLink password associated with the user name. This parameter is optional.
num_auth_parms	ul_byte	The number of authentication parameters being passed to authentication parameters in MobiLink events.
observer	ul_sync_observer_fn	A pointer to a callback function or event handler that monitors synchronization. This parameter is optional.
partial_download_retained	ul_bool	When a download fails because of a communications error during synchronization, this parameter indicates whether UltraLite applied those changes that were downloaded rather than rolling back the changes.
password	const char *	A string specifying the existing MobiLink password associated with the user name. This parameter is optional.
ping	ul_bool	Confirm communications between the UltraLite client and the MobiLink server. When this parameter is set to true, no synchronization takes place.
publications	const char *	A comma separated list of publications indicating what data to include in the synchronization.
resume_partial_download	ul_bool	Resume a failed download. The synchronization does not upload changes; it only downloads those changes that were to be downloaded in the failed download.

Member name	Type	Description
send_column_names	ul_bool	Instructs the application that column names should be sent to the MobiLink server in the upload.
send_download_ack	ul_bool	Instructs the MobiLink server whether or not the client will provide a download acknowledgement.
stream	const char *	The MobiLink network protocol to use for synchronization.
stream_error	ul_stream_error	The structure to hold communications error reporting information.
stream_parms	const char *	The options to configure the network protocol you selected.
upload_ok	ul_bool	The status of data uploaded to the MobiLink server. This field reports true if upload succeeded.
upload_only	ul_bool	Do not download any changes from the consolidated database during the current synchronization. This can save communication time, especially over slow communication links.
user_data	ul_void *	Make application-specific information available to the synchronization observer. This parameter is optional.
user_name	const char *	A string that the MobiLink server uses to identify a unique MobiLink user.
version	const char *	The version string allows an UltraLite application to choose from a set of synchronization scripts.

### Remarks

Synchronization parameters control the synchronization behavior between an UltraLite database and the MobiLink server. The Stream Type synchronization parameter, User Name synchronization parameter, and Version synchronization parameter are required. If you do not set them, the synchronization function returns an error (SQLE\_SYNC\_INFO\_INVALID or its equivalent). You can only specify one of Download Only, Ping, or Upload Only at a time. If you set more than one of these parameters to true, the synchronization function returns an error (SQLE\_SYNC\_INFO\_INVALID or its equivalent).

For more information about MobiLink client network protocol options, see [“MobiLink client network protocol options” \[MobiLink - Client Administration\]](#).

## ul\_sync\_info\_w2 structure

Stores synchronization data.

**Syntax**

```
public struct ul_sync_info_w2
```

**Members**

Member name	Type	Description
additional_parms	const ul_wchar *	A string of name value pairs "name=value;" with extra parameters.
auth_parms	const ul_wchar **	An array of authentication parameters in MobiLink events.
auth_status	ul_auth_status	The status of MobiLink user authentication. The MobiLink server provides this information to the client.
auth_value	ul_s_long	The MobiLink server provides this information to the client to determine the authentication status.
download_only	ul_bool	Do not upload any changes from the UltraLite database during the current synchronization.
ignored_rows	ul_bool	The status of ignored rows. This read-only field reports true if any rows were ignored by the MobiLink server during synchronization because of absent scripts.
init_verify	ul_sync_info_w2 *	Initialize verification.
keep_partial_download	ul_bool	When a download fails because of a communications error during synchronization, this parameter controls whether UltraLite holds on to the partial download rather than rolling back the changes.
new_password	const ul_wchar *	A string specifying a new MobiLink password associated with the user name. This parameter is optional.
num_auth_parms	ul_byte	The number of authentication parameters being passed to authentication parameters in MobiLink events.
observer	ul_sync_observer_fn	A pointer to a callback function or event handler that monitors synchronization. This parameter is optional.
partial_download_retained	ul_bool	When a download fails because of a communications error during synchronization, this parameter indicates whether UltraLite applied those changes that were downloaded rather than rolling back the changes.

Member name	Type	Description
password	const ul_wchar *	A string specifying the existing MobiLink password associated with the user name. This parameter is optional.
ping	ul_bool	Confirm communications between the UltraLite client and the MobiLink server. When this parameter is set to true, no synchronization takes place.
publications	const ul_wchar *	A comma separated list of publications indicating what data to include in the synchronization.
resume_partial_download	ul_bool	Resume a failed download. The synchronization does not upload changes; it only downloads those changes that were to be downloaded in the failed download.
send_column_names	ul_bool	Instructs the application that column names should be sent to the MobiLink server in the upload.
send_download_ack	ul_bool	Instructs the MobiLink server whether or not the client will provide a download acknowledgement.
stream	const char *	The MobiLink network protocol to use for synchronization.
stream_error	ul_stream_error	The structure to hold communications error reporting information.
stream_parms	const ul_wchar *	The options to configure the network protocol you selected.
upload_ok	ul_bool	The status of data uploaded to the MobiLink server. This field reports true if upload succeeded.
upload_only	ul_bool	Do not download any changes from the consolidated database during the current synchronization. This can save communication time, especially over slow communication links.
user_data	ul_void *	Make application-specific information available to the synchronization observer. This parameter is optional.
user_name	const ul_wchar *	A string that the MobiLink server uses to identify a unique MobiLink user.
version	const ul_wchar *	The version string allows an UltraLite application to choose from a set of synchronization scripts.

**Remarks****Note**

This structure prototype is used internally when you refer to `ul_sync_info` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this structure directly when creating an UltraLite application.

Synchronization parameters control the synchronization behavior between an UltraLite database and the MobiLink server. The Stream Type synchronization parameter, User Name synchronization parameter, and Version synchronization parameter are required. If you do not set them, the synchronization function returns an error (`SQLE_SYNC_INFO_INVALID` or its equivalent). You can only specify one of Download Only, Ping, or Upload Only at a time. If you set more than one of these parameters to true, the synchronization function returns an error (`SQLE_SYNC_INFO_INVALID` or its equivalent).

For more information about MobiLink client network protocol options, see [“MobiLink client network protocol options”](#) [*MobiLink - Client Administration*].

**See also**

- [“ul\\_sync\\_info structure”](#) on page 81

## ul\_sync\_result structure

Stores the synchronization result so that appropriate action can be taken in the application.

**Syntax**

```
public struct ul_sync_result
```

**Members**

Member name	Type	Description
<code>auth_status</code>	<code>ul_auth_status</code>	The synchronization authentication status.
<code>auth_value</code>	<code>ul_s_long</code>	The value used by the MobiLink server to determine the <code>auth_status</code> result.
<code>error_status</code>	<code>ul_error_info</code>	The error status from the last synchronization.
<code>ignored_rows</code>	<code>ul_bool</code>	True if uploaded rows were ignored; false otherwise.
<code>partial_download_retained</code>	<code>ul_bool</code>	The value that tells you that a partial download was retained. See <code>keep_partial_download</code> .
<code>received</code>	<code>ul_sync_stats</code>	Download statistics.
<code>sent</code>	<code>ul_sync_stats</code>	Upload statistics.

Member name	Type	Description
stream_error	ul_stream_error	The communication stream error information.
timestamp	SQLDATETIME	The time and date of the last synchronization.
upload_ok	ul_bool	True if the upload was successful; false otherwise.

## ul\_sync\_stats structure

Reports the statistics of the synchronization stream.

### Syntax

```
public struct ul_sync_stats
```

### Members

Member name	Type	Description
bytes	ul_u_long	The number of bytes currently sent.
deletes	ul_u_long	The number of deleted rows current sent.
inserts	ul_u_long	The number of rows currently inserted.
updates	ul_u_long	The number of updated rows currently sent.

## ul\_sync\_status structure

Returns synchronization progress monitoring data.

### Syntax

```
public struct ul_sync_status
```

### Members

Member name	Type	Description
db_table_count	ul_u_short	Returns the number of tables in database.
flags	ul_u_short	Returns the current synchronization flags indicating additional information relating to the current state.
info	ul_sync_info *	A pointer to the ul_sync_info_a structure.

Member name	Type	Description
received	ul_sync_stats	Returns download statistics.
sent	ul_sync_stats	Returns upload statistics.
sqlca	SQLCA *	The connection's active SQLCA.
state	ul_sync_state	One of the many supported states.
stop	ul_bool	A boolean that cancel synchronization. A value of true means that synchronization is canceled.
sync_table_count	ul_u_short	Returns the number of tables being synchronized.
sync_table_index	ul_u_short	1..sync_table_count
table_id	ul_u_short	The current table id which is being uploaded or downloaded (1-based). This number may skip values when not all tables are being synchronized, and is not necessarily increasing.
table_name	char	Name of the current table.
table_name_w2	ul_wchar	Name of the current table.
user_data	ul_void *	User data passed in to ULSetSynchronizationCallback or set in the ul_sync_info structure.

**See also**

- [“ul\\_sync\\_state enumeration” on page 76](#)

## ul\_validate\_data structure

Stores validation status information for the callback.

**Syntax**

```
public struct ul_validate_data
```

**Members**

Member name	Type	Description
i	ul_u_long	Parameter as an integer.
parm_count	ul_u_short	The number of parameters in the structure.



Member name	Type	Description
parm_type	enumeration	The possible parameter types.
parms	struct ul_validate_data::@3	Parameters for this validation status. Array of parameters
s	char	Parameter as a string (note that this is not a wide char).
status_id	ul_validate_status_id	Indicates what is being reported in the validation process.
stop	ul_bool	A boolean that cancels the validation. A value of true means that validation is canceled.
type	parm_type	Type of parameter stored.
user_data	ul_void *	User-defined data pointer passed into validation routine.

## ULVF\_DATABASE variable

Used to validate database.

### Syntax

```
#define ULVF_DATABASE 0x0004
```

## ULVF\_EXPRESS variable

Used to perform a faster, though less thorough, validation.

### Syntax

```
#define ULVF_EXPRESS 0x8000
```

## ULVF\_FULL\_VALIDATE variable

Performs all types of validation on the database.

### Syntax

```
#define ULVF_FULL_VALIDATE (ULVF_TABLE|ULVF_INDEX|ULVF_DATABASE)
```

## ULVF\_IDX\_HASH variable

Reports effectiveness of the index hashes (development version only).

### Syntax

```
#define ULVF_IDX_HASH 0x0200
```

## ULVF\_IDX\_REDUNDANT variable

Checks redundant indexes (development version only).

### Syntax

```
#define ULVF_IDX_REDUNDANT 0x0100
```

## ULVF\_INDEX variable

Used to validate indexes.

### Syntax

```
#define ULVF_INDEX 0x0002
```

## ULVF\_TABLE variable

Used to validate table(s).

### Syntax

```
#define ULVF_TABLE 0x0001
```

## UL\_AS\_SYNCHRONIZE variable

Provides the name of the callback message used to indicate an ActiveSync synchronization.

### Syntax

```
#define UL_AS_SYNCHRONIZE UL_TEXT( "dbas_synchronize" )
```

### See also

- [“Adding ActiveSync synchronization to your application” on page 63](#)

## UL\_SYNC\_ALL variable

Synchronizes all tables in the database that are not marked as "no sync", including tables that are not in any publication.

### Syntax

```
#define UL_SYNC_ALL UL_TEXT("")
```

### See also

- [“ul\\_sync\\_info structure” on page 81](#)
- [“ULGetLastDownloadTime method” on page 291](#)
- [“ULCountUploadRows method” on page 272](#)
- [“UL\\_SYNC\\_ALL\\_PUBS variable” on page 91](#)

## UL\_SYNC\_ALL\_PUBS variable

Synchronizes all tables in a publication.

### Syntax

```
#define UL_SYNC_ALL_PUBS UL_TEXT("")
```

### See also

- [“ul\\_sync\\_info structure” on page 81](#)
- [“ULGetLastDownloadTime method” on page 291](#)
- [“ULCountUploadRows method” on page 272](#)
- [“UL\\_SYNC\\_ALL variable” on page 91](#)

## UL\_SYNC\_STATUS\_FLAG\_IS\_BLOCKING variable

Defines a bit set in the `ul_sync_status.flags` field to indicate that the synchronization is blocked awaiting a response from the MobiLink server.

### Syntax

```
#define UL_SYNC_STATUS_FLAG_IS_BLOCKING 1
```

## UL\_VALID\_IS\_ERROR variable

Evaluates true if a given `ul_validate_status_id` is an error status.

**Syntax**

```
#define UL_VALID_IS_ERROR
```

## UL\_VALID\_IS\_INFO variable

Evaluates true if a given `ul_validate_status_id` is an informational status.

**Syntax**

```
#define UL_VALID_IS_INFO
```

---

# UltraLite C/C++ API reference

This section lists functions that you can use with the C++ interface.

## Header file

- `ulcpp.h`

### Note

This version of the UltraLite C/C++ API supersedes all previous versions which have been deprecated. For more information, see “[SQL Anywhere deprecated and discontinued features](#)” [[SQL Anywhere 12 - Changes and Upgrading](#)].

You can use the old implementation of the UltraLite C/C++ API by adding the `%SQLANY12%\SDK\ulcpp11.cpp` file to your UltraLite application project, where `SQLANY12` is an environment variable that points to your SQL Anywhere installation directory.

## ULConnection class

Represents a connection to an UltraLite database.

### Syntax

```
public class ULConnection
```

### Members

All members of `ULConnection` class, including all inherited members.

Name	Description
<a href="#">“CancelGetNotification method”</a>	Cancels any pending get-notification calls on all queues matching the given name.
<a href="#">“ChangeEncryptionKey method”</a>	Changes the database encryption key.
<a href="#">“Checkpoint method”</a>	Checkpoints the database.
<a href="#">“Close method”</a>	Destroys this connection and any remaining associated objects.
<a href="#">“Commit method”</a>	Commits the current transaction.
<a href="#">“CountUploadRows method”</a>	Determines the number of rows that need to be uploaded.
<a href="#">“CreateNotificationQueue method”</a>	Creates an event notification queue for this connection.

Name	Description
“DeclareEvent method”	Declares an event which can then be registered for and triggered.
“DestroyNotificationQueue method”	Destroys the given event notification queue.
“ExecuteScalar method”	Executes a SQL SELECT statement.
“ExecuteScalarV method”	Executes a SQL SELECT statement string, along with a list of substitution values.
“ExecuteStatement method”	Executes a SQL statement string directly.
“GetChildObjectCount method”	Gets the number of currently open child objects on the connection.
“GetDatabaseProperty method”	Gets a database property.
“GetDatabasePropertyInt method”	Gets a database property.
“GetDatabaseSchema method”	Returns an object pointer used to query the schema of the database.
“GetLastDownloadTime method”	Gets the time of the last download.
“GetLastError method”	Returns the error information associated with the last call.
“GetLastIdentity method”	Gets the @@identity value.
“GetNotification method”	Reads an event notification.
“GetNotificationParameter method”	Gets a parameter for the event notification just read by GetNotification().
“GetSqlca method”	Gets the communication area associated with this connection.
“GetSyncResult method”	Gets the result of the last synchronization.
“GlobalAutoincUsage method”	Gets the percent of the global autoincrement values used by the counter.
“GrantConnectTo method”	Creates a new user or changes the password of an existing one.
“InitSyncInfo method”	Initializes the synchronization information structure.
“OpenTable method”	Opens a table.
“PrepareStatement method”	Prepares a SQL statement.

Name	Description
“RegisterForEvent method”	Registers or unregisters a queue to receive notifications of an event.
“ResetLastDownloadTime method”	Resets the time of the last download of the named publication.
“RevokeConnectFrom method”	Deletes an existing user.
“Rollback method”	Rolls back the current transaction.
“RollbackPartialDownload method”	Rolls back a partial download.
“SendNotification method”	Sends a notification to all queues matching the given name.
“SetDatabaseOption method”	Sets the specified database option.
“SetDatabaseOptionInt method”	Sets a database option.
“SetSynchronizationCallback method”	Sets the callback to be invoked while performing a synchronization.
“SetSyncInfo method”	Creates a synchronization profile.
“StartSynchronizationDelete method”	Sets START SYNCHRONIZATION DELETE for this connection.
“StopSynchronizationDelete method”	Sets STOP SYNCHRONIZATION DELETE for this connection.
“Synchronize method”	Synchronizes the database.
“SynchronizeFromProfile method”	Synchronize the database using the given profile and merge parameters.
“TriggerEvent method”	Triggers a user-defined event and sends notifications to all registered queues.
“ValidateDatabase method”	Validates the database on this connection.

## CancelGetNotification method

Cancels any pending get-notification calls on all queues matching the given name.

**Overload list**

Name	Description
<a href="#">“CancelGetNotification(const char *) method”</a>	Cancels any pending get-notification calls on all queues matching the given name.
<a href="#">“CancelGetNotification(const ul_wchar *) method”</a>	Cancels any pending get-notification calls on all queues matching the given name.

## CancelGetNotification(const char \*) method

Cancels any pending get-notification calls on all queues matching the given name.

**Syntax**

```
public virtual ul_u_long CancelGetNotification(const char * queueName)
```

**Parameters**

- **queueName** The name of the queue.

**Returns**

The number of affected queues (not the number of blocked reads necessarily).

## CancelGetNotification(const ul\_wchar \*) method

Cancels any pending get-notification calls on all queues matching the given name.

**Syntax**

```
public virtual ul_u_long CancelGetNotification(  
    const ul_wchar * queueName  
)
```

**Parameters**

- **queueName** The name of the queue.

**Returns**

The number of affected queues (not the number of blocked reads necessarily).

## ChangeEncryptionKey method

Changes the database encryption key.



**Overload list**

Name	Description
<a href="#">“ChangeEncryptionKey(const char *) method”</a>	Changes the database encryption key.
<a href="#">“ChangeEncryptionKey(const ul_wchar *) method”</a>	Changes the database encryption key.

**ChangeEncryptionKey(const char \*) method**

Changes the database encryption key.

**Syntax**

```
public virtual bool ChangeEncryptionKey(const char * newKey)
```

**Parameters**

- **newKey** The new database encryption key.

**Returns**

True on success, otherwise false.

**ChangeEncryptionKey(const ul\_wchar \*) method**

Changes the database encryption key.

**Syntax**

```
public virtual bool ChangeEncryptionKey(const ul_wchar * newKey)
```

**Parameters**

- **newKey** The new database encryption key.

**Returns**

True on success, otherwise false.

**Checkpoint method**

Checkpoints the database.

**Syntax**

```
public virtual bool Checkpoint()
```

**Returns**

True on success. False on failure.

## Close method

Destroys this connection and any remaining associated objects.

### Syntax

```
public virtual void Close(ULError * error)
```

### Parameters

- **error** An optional ULError object to receive error information.

## Commit method

Commits the current transaction.

### Syntax

```
public virtual bool Commit()
```

### Returns

True on success, otherwise false.

## CountUploadRows method

Determines the number of rows that need to be uploaded.

### Overload list

Name	Description
<a href="#">“CountUploadRows(const char *, ul_u_long) method”</a>	Determines the number of rows that need to be uploaded.
<a href="#">“CountUploadRows(const ul_wchar *, ul_u_long) method”</a>	Determines the number of rows that need to be uploaded.

## CountUploadRows(const char \*, ul\_u\_long) method

Determines the number of rows that need to be uploaded.

### Syntax

```
public virtual ul_u_long CountUploadRows (  
    const char * pubList,  
    ul_u_long threshold  
)
```

**Parameters**

- **pubList** A comma separated list of publications to consider.
- **threshold** The limit on the number of rows to count.

**Returns**

The number of rows in the publication that would be uploaded at this time.

**CountUploadRows(const ul\_wchar \*, ul\_u\_long) method**

Determines the number of rows that need to be uploaded.

**Syntax**

```
public virtual ul_u_long CountUploadRows (
    const ul_wchar * pubList,
    ul_u_long threshold
)
```

**Parameters**

- **pubList** A comma separated list of publications to consider.
- **threshold** The limit on the number of rows to count.

**Returns**

The number of rows in the publication that would be uploaded at this time.

**CreateNotificationQueue method**

Creates an event notification queue for this connection.

**Overload list**

Name	Description
<a href="#">“CreateNotificationQueue(const char *, const char *) method”</a>	Creates an event notification queue for this connection.
<a href="#">“CreateNotificationQueue(const ul_wchar *, const ul_wchar *) method”</a>	Creates an event notification queue for this connection.

**CreateNotificationQueue(const char \*, const char \*) method**

Creates an event notification queue for this connection.

**Syntax**

```
public virtual bool CreateNotificationQueue(  
    const char * name,  
    const char * parameters  
)
```

**Parameters**

- **name** Name for the new queue.
- **parameters** Creation parameters; currently unused, set to NULL.

**Returns**

True on success, otherwise false.

**Remarks**

Queue names are scoped per-connection, so different connections can create queues with the same name. When an event notification is sent, all queues in the database with a matching name receive (a separate instance of) the notification. Names are case insensitive. A default queue is created on demand for each connection when calling RegisterForEvent() if no queue is specified. This call fails with an error if the name already exists or isn't valid.

## CreateNotificationQueue(const ul\_wchar \*, const ul\_wchar \*) method

Creates an event notification queue for this connection.

**Syntax**

```
public virtual bool CreateNotificationQueue(  
    const ul_wchar * name,  
    const ul_wchar * parameters  
)
```

**Parameters**

- **name** Name for the new queue.
- **parameters** Creation parameters; currently unused, set to NULL.

**Returns**

True on success, otherwise false.

**Remarks**

Queue names are scoped per-connection, so different connections can create queues with the same name. When an event notification is sent, all queues in the database with a matching name receive (a separate instance of) the notification. Names are case insensitive. A default queue is created on demand for each connection when calling RegisterForEvent() if no queue is specified. This call fails with an error if the name already exists or isn't valid.

## DeclareEvent method

Declares an event which can then be registered for and triggered.

### Overload list

Name	Description
<code>“DeclareEvent(const char *) method”</code>	Declares an event which can then be registered for and triggered.
<code>“DeclareEvent(const ul_wchar *) method”</code>	Declares an event which can then be registered for and triggered.

## DeclareEvent(const char \*) method

Declares an event which can then be registered for and triggered.

### Syntax

```
public virtual bool DeclareEvent(const char * eventName)
```

### Parameters

- **eventName** The name for the new user-defined event.

### Returns

True if the event was declared successfully, false if the name is already used or not valid.

### Remarks

UltraLite predefines some system events triggered by operations on the database or the environment. This function declares user-defined events. User-defined events are triggered with `TriggerEvent()`. The event name must be unique. Names are case insensitive.

## DeclareEvent(const ul\_wchar \*) method

Declares an event which can then be registered for and triggered.

### Syntax

```
public virtual bool DeclareEvent(const ul_wchar * eventName)
```

### Parameters

- **eventName** The name for the new user-defined event.

### Returns

True if the event was declared successfully, false if the name is already used or not valid.

**Remarks**

UltraLite predefines some system events triggered by operations on the database or the environment. This function declares user-defined events. User-defined events are triggered with `TriggerEvent()`. The event name must be unique. Names are case insensitive.

## DestroyNotificationQueue method

Destroys the given event notification queue.

**Overload list**

Name	Description
<a href="#">“DestroyNotificationQueue(const char *) method”</a>	Destroys the given event notification queue.
<a href="#">“DestroyNotificationQueue(const ul_wchar *) method”</a>	Destroys the given event notification queue.

## DestroyNotificationQueue(const char \*) method

Destroys the given event notification queue.

**Syntax**

```
public virtual bool DestroyNotificationQueue(const char * name)
```

**Parameters**

- **name** The name of the queue to destroy.

**Returns**

True on success, otherwise false.

**Remarks**

A warning is signaled if unread notifications remain in the queue. Unread notifications are discarded. A connection's default event queue, if created, is destroyed when the connection is closed.

## DestroyNotificationQueue(const ul\_wchar \*) method

Destroys the given event notification queue.

**Syntax**

```
public virtual bool DestroyNotificationQueue(const ul_wchar * name)
```

**Parameters**

- **name** The name of the queue to destroy.

**Returns**

True on success, otherwise false.

**Remarks**

A warning is signaled if unread notifications remain in the queue. Unread notifications are discarded. A connection's default event queue, if created, is destroyed when the connection is closed.

## ExecuteScalar method

Executes a SQL SELECT statement.

**Overload list**

Name	Description
<a href="#">“ExecuteScalar(void *, size_t, ul_column_storage_type, const char *, ...) method”</a>	Executes a SQL SELECT statement.
<a href="#">“ExecuteScalar(void *, size_t, ul_column_storage_type, const ul_wchar *, ...) method”</a>	Executes a SQL SELECT statement.

## ExecuteScalar(void \*, size\_t, ul\_column\_storage\_type, const char \*, ...) method

Executes a SQL SELECT statement.

**Syntax**

```
public virtual bool ExecuteScalar(
    void * dstPtr,
    size_t dstSize,
    ul_column_storage_type dstType,
    const char * sql,
    ...
)
```

**Parameters**

- **dstPtr** A pointer to a variable of the required type to receive the value.
- **dstSize** The size of variable to receive value, if applicable.
- **dstType** The type of value to retrieve; must match variable type.
- **sql** The SELECT statement, optionally containing '?' parameters.
- **...** String (char \*) parameter values to substitute.

## Returns

True if the query is successfully executed and a value is successfully retrieved; otherwise, false is returned when a value is not fetched. Check the SQLCODE to determine why false is returned. The selected value is NULL if no warning or error (SQLE\_NOERROR) is indicated.

## Remarks

*dstPtr* must point to a variable of the correct type, matching the *dstType*. *dstSize* is only required for variable-sized values, such as strings and binaries, and is otherwise ignored. The variable list of parameter values must correspond to parameters in the statement, and all values are assumed to be strings (Internally, UltraLite casts the parameter values as required for the statement).

The following types are supported:

- **#UL\_TYPE\_BIT/#UL\_TYPE\_TINY** Use variable type `ul_byte` (8 bit, unsigned).
- **#UL\_TYPE\_U\_SHORT/#UL\_TYPE\_S\_SHORT** Use variable type `ul_u_short/ul_s_short` (16 bit).
- **#UL\_TYPE\_U\_LONG/#UL\_TYPE\_S\_LONG** Use variable type `ul_u_long/ul_s_long` (32 bit).
- **#UL\_TYPE\_U\_BIG/#UL\_TYPE\_S\_BIG** Use variable type `ul_u_big/ul_s_big` (64 bit).
- **#UL\_TYPE\_DOUBLE** Use variable type `ul_double` (double).
- **#UL\_TYPE\_REAL** Use variable type `ul_real` (float).
- **#UL\_TYPE\_BINARY** Use variable type `ul_binary` and specify *dstSize* (as in `GetBinary()`).
- **#UL\_TYPE\_TIMESTAMP\_STRUCT** Use variable type `DECL_DATETIME`.
- **#UL\_TYPE\_CHAR** Use variable type `char []` (a character buffer), and set *dstSize* to the size of the buffer (as in `GetString()`).
- **#UL\_TYPE\_WCHAR** Use variable type `ul_wchar []` (a wide character buffer), and set *dstSize* to the size of the buffer (as in `GetString()`).
- **#UL\_TYPE\_TCHAR** Same as `UL_TYPE_CHAR` or `UL_TYPE_WCHAR`, depending on which version of the function is called.

For more information about these types, see [“ul\\_column\\_storage\\_type enumeration” on page 72](#).

The following example demonstrates integer fetching:

```
ul_u_long    val;
ok = conn->ExecuteScalar( &val, 0, UL_TYPE_U_LONG,
    "SELECT count(*) FROM t WHERE col LIKE ?", "ABC%" );
```

The following example demonstrates string fetching:

```
char    val[40];
ok = conn->ExecuteScalar( &val, sizeof(val), UL_TYPE_CHAR,
    "SELECT uuidtostr( newid() )" );
```



## ExecuteScalar(void \*, size\_t, ul\_column\_storage\_type, const ul\_wchar \*, ...) method

Executes a SQL SELECT statement.

### Syntax

```
public virtual bool ExecuteScalar(
    void * dstPtr,
    size_t dstSize,
    ul_column_storage_type dstType,
    const ul_wchar * sql,
    ...
)
```

### Parameters

- **dstPtr** A pointer to a variable of the required type to receive the value.
- **dstSize** The size of variable to receive value, if applicable.
- **dstType** The type of value to retrieve; must match variable type.
- **sql** The SELECT statement, optionally containing '?' parameters.
- **...** String (char \*) parameter values to substitute.

### Returns

True if the query is successfully executed and a value is successfully retrieved; otherwise, false is returned when a value is not fetched. Check the SQLCODE to determine why false is returned. The selected value is NULL if no warning or error (SQLE\_NOERROR) is indicated.

### Remarks

*dstPtr* must point to a variable of the correct type, matching the *dstType*. *dstSize* is only required for variable-sized values, such as strings and binaries, and is otherwise ignored. The variable list of parameter values must correspond to parameters in the statement, and all values are assumed to be strings (Internally, UltraLite casts the parameter values as required for the statement).

The following types are supported:

- **#UL\_TYPE\_BIT/#UL\_TYPE\_TINY** Use variable type ul\_byte (8 bit, unsigned).
- **#UL\_TYPE\_U\_SHORT/#UL\_TYPE\_S\_SHORT** Use variable type ul\_u\_short/ul\_s\_short (16 bit).
- **#UL\_TYPE\_U\_LONG/#UL\_TYPE\_S\_LONG** Use variable type ul\_u\_long/ul\_s\_long (32 bit).
- **#UL\_TYPE\_U\_BIG/#UL\_TYPE\_S\_BIG** Use variable type ul\_u\_big/ul\_s\_big (64 bit).
- **#UL\_TYPE\_DOUBLE** Use variable type ul\_double (double).
- **#UL\_TYPE\_REAL** Use variable type ul\_real (float).

- **#UL\_TYPE\_BINARY** Use variable type `ul_binary` and specify *dstSize* (as in `GetBinary()`).
- **#UL\_TYPE\_TIMESTAMP\_STRUCT** Use variable type `DECL_DATETIME`.
- **#UL\_TYPE\_CHAR** Use variable type `char []` (a character buffer), and set *dstSize* to the size of the buffer (as in `GetString()`).
- **#UL\_TYPE\_WCHAR** Use variable type `ul_wchar []` (a wide character buffer), and set *dstSize* to the size of the buffer (as in `GetString()`).
- **#UL\_TYPE\_TCHAR** Same as `UL_TYPE_CHAR` or `UL_TYPE_WCHAR`, depending on which version of the function is called.

For more information about these types, see [“ul\\_column\\_storage\\_type enumeration” on page 72](#).

The following example demonstrates integer fetching:

```
ul_u_long    val;
ok = conn->ExecuteScalar( &val, 0, UL_TYPE_U_LONG,
    "SELECT count(*) FROM t WHERE col LIKE ?", "ABC%" );
```

The following example demonstrates string fetching:

```
char    val[40];
ok = conn->ExecuteScalar( &val, sizeof(val), UL_TYPE_CHAR,
    "SELECT uuidtostr( newid() )" );
```

## ExecuteScalarV method

Executes a SQL SELECT statement string, along with a list of substitution values.

### Overload list

Name	Description
“ExecuteScalarV(void *, size_t, ul_column_storage_type, const char *, va_list) method”	Executes a SQL SELECT statement string, along with a list of substitution values.
“ExecuteScalarV(void *, size_t, ul_column_storage_type, const ul_wchar *, va_list) method”	Executes a SQL SELECT statement, along with a list of substitution values.

## ExecuteScalarV(void \*, size\_t, ul\_column\_storage\_type, const char \*, va\_list) method

Executes a SQL SELECT statement string, along with a list of substitution values.

### Syntax

```
public virtual bool ExecuteScalarV(
    void * dstPtr,
```

```

    size_t dstSize,
    ul_column_storage_type dstType,
    const char * sql,
    va_list args
)

```

### Parameters

- **dstPtr** A pointer to a variable of the required type to receive the value.
- **dstSize** The size of variable to receive value, if applicable.
- **dstType** The type of value to retrieve; must match variable type.
- **sql** The SELECT statement, optionally containing '?' parameters.
- **args** A list of string (char \*) values to substitute.

### Returns

True if the query is successfully executed and a value is successfully retrieved; otherwise, false is returned when a value is not fetched. Check the SQLCODE to determine why false is returned. The selected value is NULL if no warning or error (SQLE\_NOERROR) is indicated.

### Remarks

*dstPtr* must point to a variable of the correct type, matching the *dstType*. *dstSize* is only required for variable-sized values, such as strings and binaries, and is otherwise ignored. The variable list of parameter values must correspond to parameters in the statement, and all values are assumed to be strings (Internally, UltraLite casts the parameter values as required for the statement).

The following types are supported:

- **#UL\_TYPE\_BIT/#UL\_TYPE\_TINY** Use variable type ul\_byte (8 bit, unsigned).
- **#UL\_TYPE\_U\_SHORT/#UL\_TYPE\_S\_SHORT** Use variable type ul\_u\_short/ul\_s\_short (16 bit).
- **#UL\_TYPE\_U\_LONG/#UL\_TYPE\_S\_LONG** Use variable type ul\_u\_long/ul\_s\_long (32 bit).
- **#UL\_TYPE\_U\_BIG/#UL\_TYPE\_S\_BIG** Use variable type ul\_u\_big/ul\_s\_big (64 bit).
- **#UL\_TYPE\_DOUBLE** Use variable type ul\_double (double).
- **#UL\_TYPE\_REAL** Use variable type ul\_real (float).
- **#UL\_TYPE\_BINARY** Use variable type ul\_binary and specify *dstSize* (as in GetBinary()).
- **#UL\_TYPE\_TIMESTAMP\_STRUCT** Use variable type DECL\_DATETIME.
- **#UL\_TYPE\_CHAR** Use variable type char [] (a character buffer), and set *dstSize* to the size of the buffer (as in GetString()).

- **#UL\_TYPE\_WCHAR** Use variable type `ul_wchar []` (a wide character buffer), and set *dstSize* to the size of the buffer (as in `GetString()`).
- **#UL\_TYPE\_TCHAR** Same as `UL_TYPE_CHAR` or `UL_TYPE_WCHAR`, depending on which version of the function is called.

For more information about these types, see [“ul\\_column\\_storage\\_type enumeration” on page 72](#).

## ExecuteScalarV(void \*, size\_t, ul\_column\_storage\_type, const ul\_wchar \*, va\_list) method

Executes a SQL SELECT statement, along with a list of substitution values.

### Syntax

```
public virtual bool ExecuteScalarV(  
    void * dstPtr,  
    size_t dstSize,  
    ul_column_storage_type dstType,  
    const ul_wchar * sql,  
    va_list args  
)
```

### Parameters

- **dstPtr** A pointer to a variable of the required type to receive the value.
- **dstSize** The size of variable to receive value, if applicable.
- **dstType** The type of value to retrieve; must match variable type.
- **sql** The SELECT statement, optionally containing '?' parameters.
- **args** A list of string (char \*) values to substitute.

### Returns

True if the query is successfully executed and a value is successfully retrieved; otherwise, false is returned when a value is not fetched. Check the `SQLCODE` to determine why false is returned. The selected value is NULL if no warning or error (`SQLE_NOERROR`) is indicated.

### Remarks

*dstPtr* must point to a variable of the correct type, matching the *dstType*. *dstSize* is only required for variable-sized values, such as strings and binaries, and is otherwise ignored. The variable list of parameter values must correspond to parameters in the statement, and all values are assumed to be strings (Internally, UltraLite casts the parameter values as required for the statement).

The following types are supported:

- **#UL\_TYPE\_BIT/#UL\_TYPE\_TINY** Use variable type `ul_byte` (8 bit, unsigned).

- **#UL\_TYPE\_U\_SHORT/#UL\_TYPE\_S\_SHORT** Use variable type `ul_u_short/ul_s_short` (16 bit).
- **#UL\_TYPE\_U\_LONG/#UL\_TYPE\_S\_LONG** Use variable type `ul_u_long/ul_s_long` (32 bit).
- **#UL\_TYPE\_U\_BIG/#UL\_TYPE\_S\_BIG** Use variable type `ul_u_big/ul_s_big` (64 bit).
- **#UL\_TYPE\_DOUBLE** Use variable type `ul_double` (double).
- **#UL\_TYPE\_REAL** Use variable type `ul_real` (float).
- **#UL\_TYPE\_BINARY** Use variable type `ul_binary` and specify *dstSize* (as in `GetBinary()`).
- **#UL\_TYPE\_TIMESTAMP\_STRUCT** Use variable type `DECL_DATETIME`.
- **#UL\_TYPE\_CHAR** Use variable type `char []` (a character buffer), and set *dstSize* to the size of the buffer (as in `GetString()`).
- **#UL\_TYPE\_WCHAR** Use variable type `ul_wchar []` (a wide character buffer), and set *dstSize* to the size of the buffer (as in `GetString()`).
- **#UL\_TYPE\_TCHAR** Same as `UL_TYPE_CHAR` or `UL_TYPE_WCHAR`, depending on which version of the function is called.

For more information about these types, see [“ul\\_column\\_storage\\_type enumeration” on page 72](#).

## ExecuteStatement method

Executes a SQL statement string directly.

### Overload list

Name	Description
<a href="#">“ExecuteStatement(const char *) method”</a>	Executes a SQL statement string directly.
<a href="#">“ExecuteStatement(const ul_wchar *) method”</a>	Executes a SQL statement directly.

## ExecuteStatement(const char \*) method

Executes a SQL statement string directly.

### Syntax

```
public virtual bool ExecuteStatement(const char * sql)
```

### Parameters

- **sql** The SQL script to execute.

**Returns**

True on success; otherwise, false is returned.

## ExecuteStatement(const ul\_wchar \*) method

Executes a SQL statement directly.

**Syntax**

```
public virtual bool ExecuteStatement(const ul_wchar * sql)
```

**Parameters**

- **sql** The SQL script to execute.

**Returns**

True on success; otherwise, false is returned.

## GetChildObjectCount method

Gets the number of currently open child objects on the connection.

**Syntax**

```
public virtual ul_u_long GetChildObjectCount()
```

**Returns**

The number of currently open child objects.

**Remarks**

This method can be used to detect object leaks.

## GetDatabaseProperty method

Gets a database property.

**Overload list**

Name	Description
<a href="#">“GetDatabaseProperty(const char *) method”</a>	Gets a database property.
<a href="#">“GetDatabaseProperty(const ul_wchar *) method”</a>	Gets a database property.

---

## GetDatabaseProperty(const char \*) method

Gets a database property.

### Syntax

```
public virtual const char * GetDatabaseProperty(const char * propName)
```

### Parameters

- **propName** The name of the property being requested.

### Returns

A pointer to a string buffer containing the database property value is returned when run successfully; otherwise, NULL is returned.

### Remarks

The returned value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you must make a copy of the value if you need to save it.

## GetDatabaseProperty(const ul\_wchar \*) method

Gets a database property.

### Syntax

```
public virtual const ul_wchar * GetDatabaseProperty(  
    const ul_wchar * propName  
)
```

### Parameters

- **propName** The name of the property being requested.

### Returns

A pointer to a string buffer containing the database property value is returned when run successfully; otherwise, NULL is returned.

### Remarks

The returned value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you must make a copy of the value if you need to save it.

## GetDatabasePropertyInt method

Gets a database property.

**Overload list**

Name	Description
<a href="#">“GetDatabasePropertyInt(const char *) method”</a>	Gets a database property.
<a href="#">“GetDatabasePropertyInt(const ul_wchar *) method”</a>	Gets the database property specified by the provided ul_wchars.

## GetDatabasePropertyInt(const char \*) method

Gets a database property.

**Syntax**

```
public virtual ul_u_long GetDatabasePropertyInt(const char * propName)
```

**Parameters**

- **propName** The name of the property being requested.

**Returns**

If successful, the integer value of the property. 0 on failure.

## GetDatabasePropertyInt(const ul\_wchar \*) method

Gets the database property specified by the provided ul\_wchars.

**Syntax**

```
public virtual ul_u_long GetDatabasePropertyInt(  
    const ul_wchar * propName  
)
```

**Parameters**

- **propName** The name of the property being requested.

**Returns**

If successful, the integer value of the property. 0 on failure.

## GetDatabaseSchema method

Returns an object pointer used to query the schema of the database.

**Syntax**

```
public virtual ULDatabaseSchema * GetDatabaseSchema()
```



**Returns**

A ULDatabaseSchema object used to query the schema of the database.

**GetLastDownloadTime method**

Gets the time of the last download.

**Overload list**

Name	Description
“GetLastDownloadTime(const char *, DECL_DATETIME *) method”	Gets the time of the last download.
“GetLastDownloadTime(const ul_wchar *, DECL_DATETIME *) method”	Gets the time of the last download.

**GetLastDownloadTime(const char \*, DECL\_DATETIME \*) method**

Gets the time of the last download.

**Syntax**

```
public virtual bool GetLastDownloadTime(
    const char * pubList,
    DECL_DATETIME * value
)
```

**Parameters**

- **pubList** A comma separated list of publications to consider.
- **value** The last download time.

**Returns**

True on success, otherwise false.

**GetLastDownloadTime(const ul\_wchar \*, DECL\_DATETIME \*) method**

Gets the time of the last download.

**Syntax**

```
public virtual bool GetLastDownloadTime(
    const ul_wchar * pubList,
    DECL_DATETIME * value
)
```

### Parameters

- **pubList** A comma separated list of publications to consider.
- **value** The last download time.

### Returns

True on success, otherwise false.

## GetLastError method

Returns the error information associated with the last call.

### Syntax

```
public virtual const ULError * GetLastError()
```

### Returns

A pointer to the ULError object with information associated with the last call.

## GetLastIdentity method

Gets the @@identity value.

### Syntax

```
public virtual ul_u_big GetLastIdentity()
```

### Returns

The last value inserted into an autoincrement or global autoincrement column

### Remarks

This value is the last value inserted into an autoincrement or global autoincrement column for the database. This value is not recorded when the database is shutdown, so calling this method before any autoincrement values have been inserted returns 0.

<b>Note</b>
-------------

The last value inserted may have been on another connection.
--------------------------------------------------------------

## GetNotification method

Reads an event notification.

**Overload list**

Name	Description
<a href="#">“GetNotification(const char *, ul_u_long) method”</a>	Reads an event notification.
<a href="#">“GetNotification(const ul_wchar *, ul_u_long) method”</a>	Reads an event notification.

**GetNotification(const char \*, ul\_u\_long) method**

Reads an event notification.

**Syntax**

```
public virtual const char * GetNotification(
    const char * queueName,
    ul_u_long waitms
)
```

**Parameters**

- **queueName** The queue to read or NULL for default connection queue.
- **waitms** The time to wait (block) before returning.

**Returns**

The name of the event read or NULL on error.

**Remarks**

This call blocks until a notification is received or until the given wait period expires. To wait indefinitely, pass `UL_READ_WAIT_INFINITE` for *wait\_ms*. To cancel a wait, send another notification to the given queue or use `CancelGetNotification()`. Use `GetNotificationParameter()` after reading a notification to retrieve additional parameters by name.

**See also**

- [“CancelGetNotification method” on page 95](#)
- [“GetNotificationParameter method” on page 116](#)

**GetNotification(const ul\_wchar \*, ul\_u\_long) method**

Reads an event notification.

**Syntax**

```
public virtual const ul_wchar * GetNotification(
    const ul_wchar * queueName,
    ul_u_long waitms
)
```

**Parameters**

- **queueName** The queue to read or NULL for default connection queue.
- **waitms** The time to wait (block) before returning.

**Returns**

The name of the event read or NULL on error.

**Remarks**

This call blocks until a notification is received or until the given wait period expires. To wait indefinitely, pass `UL_READ_WAIT_INFINITE` for *wait\_ms*. To cancel a wait, send another notification to the given queue or use `CancelGetNotification()`. Use `GetNotificationParameter()` after reading a notification to retrieve additional parameters by name.

**See also**

- [“CancelGetNotification method” on page 95](#)
- [“GetNotificationParameter method” on page 116](#)

## GetNotificationParameter method

Gets a parameter for the event notification just read by `GetNotification()`.

**Overload list**

Name	Description
<a href="#">“GetNotificationParameter(const char *, const char *) method”</a>	Gets a parameter for the event notification just read by <code>GetNotification()</code> .
<a href="#">“GetNotificationParameter(const ul_wchar *, const ul_wchar *) method”</a>	Gets a parameter for the event notification just read by <code>GetNotification()</code> .

## GetNotificationParameter(const char \*, const char \*) method

Gets a parameter for the event notification just read by `GetNotification()`.

**Syntax**

```
public virtual const char * GetNotificationParameter(  
    const char * queueName,  
    const char * parameterName  
)
```

**Parameters**

- **queueName** The queue name matching the `GetNotification()` call.

- **parameterName** The name of the parameter to read (or "\*").

### Returns

The parameter value or NULL on error.

### Remarks

Only the parameters from the most-recently read notification on the given queue are available. Parameters are retrieved by name. A parameter name of "\*" retrieves the entire parameter string.

### See also

- [“GetNotification method” on page 114](#)

## GetNotificationParameter(const ul\_wchar \*, const ul\_wchar \*) method

Gets a parameter for the event notification just read by GetNotification().

### Syntax

```
public virtual const ul_wchar * GetNotificationParameter(  
    const ul_wchar * queueName,  
    const ul_wchar * parameterName  
)
```

### Parameters

- **queueName** The queue name matching the GetNotification() call.
- **parameterName** The name of the parameter to read (or "\*").

### Returns

The parameter value or NULL on error.

### Remarks

Only the parameters from the most-recently read notification on the given queue are available. Parameters are retrieved by name. A parameter name of "\*" retrieves the entire parameter string.

### See also

- [“GetNotification method” on page 114](#)

## GetSqlca method

Gets the communication area associated with this connection.

### Syntax

```
public virtual SQLCA * GetSqlca()
```

**Returns**

A pointer to the SQLCA object for this connection.

## GetSyncResult method

Gets the result of the last synchronization.

**Syntax**

```
public virtual bool GetSyncResult(ul_sync_result * syncResult)
```

**Parameters**

- **syncResult** A pointer to the ul\_sync\_result structure that holds the synchronization results.

**Returns**

True on success, otherwise false.

**Remarks**

For more information about the ul\_sync\_result structure, see [“ul\\_sync\\_result structure” on page 86](#).

## GlobalAutoincUsage method

Gets the percent of the global autoincrement values used by the counter.

**Syntax**

```
public virtual ul_u_short GlobalAutoincUsage()
```

**Returns**

The percent of the global autoincrement values used by the counter.

## GrantConnectTo method

Creates a new user or changes the password of an existing one.

**Overload list**

Name	Description
<a href="#">“GrantConnectTo(const char *, const char *) method”</a>	Creates a new user or changes the password of an existing one.

Name	Description
“GrantConnectTo(const ul_wchar *, const ul_wchar *) method”	Creates a new user or changes the password of an existing one.

## GrantConnectTo(const char \*, const char \*) method

Creates a new user or changes the password of an existing one.

### Syntax

```
public virtual bool GrantConnectTo(const char * uid, const char * pwd)
```

### Parameters

- **uid** The name of the user to create or change the password for.
- **pwd** The password for this user.

### Returns

True on success, otherwise false.

## GrantConnectTo(const ul\_wchar \*, const ul\_wchar \*) method

Creates a new user or changes the password of an existing one.

### Syntax

```
public virtual bool GrantConnectTo(
    const ul_wchar * uid,
    const ul_wchar * pwd
)
```

### Parameters

- **uid** The name of the user to create or change the password for.
- **pwd** The password for this user.

### Returns

True on success, otherwise false.

## InitSyncInfo method

Initializes the synchronization information structure.

**Overload list**

Name	Description
<a href="#">“InitSyncInfo(ul_sync_info *) method”</a>	Initializes the synchronization information structure.
<a href="#">“InitSyncInfo(ul_sync_info_w2 *) method”</a>	Initializes the synchronization information structure.

## InitSyncInfo(ul\_sync\_info \*) method

Initializes the synchronization information structure.

**Syntax**

```
public virtual void InitSyncInfo(ul_sync_info * info)
```

**Parameters**

- **info** A pointer to the ul\_sync\_info structure that holds the synchronization parameters.

## InitSyncInfo(ul\_sync\_info\_w2 \*) method

Initializes the synchronization information structure.

**Syntax**

```
public virtual void InitSyncInfo(ul_sync_info_w2 * info)
```

**Parameters**

- **info** A pointer to the ul\_sync\_info structure that holds the synchronization parameters.

## OpenTable method

Opens a table.

**Overload list**

Name	Description
<a href="#">“OpenTable(const char *, const char *) method”</a>	Opens a table.
<a href="#">“OpenTable(const ul_wchar *, const ul_wchar *) method”</a>	Opens a table.

## OpenTable(const char \*, const char \*) method

Opens a table.



**Syntax**

```
public virtual ULTable * OpenTable(  
    const char * tableName,  
    const char * indexName  
)
```

**Parameters**

- **tableName** The name of the table to open.
- **indexName** The name of the index to open the table on. Pass NULL to open on the primary key and the empty string to open the table unordered.

**Returns**

The ULTable object when the call is successful; otherwise, NULL is returned.

**Remarks**

The cursor position is set to BeforeFirst() when the application first opens a table.

## OpenTable(const ul\_wchar \*, const ul\_wchar \*) method

Opens a table.

**Syntax**

```
public virtual ULTable * OpenTable(  
    const ul_wchar * tableName,  
    const ul_wchar * indexName  
)
```

**Parameters**

- **tableName** The name of the table to open.
- **indexName** The name of the index to open the table on. Pass NULL to open on the primary key and the empty string to open the table unordered.

**Returns**

The ULTable object when the call is successful; otherwise, NULL is returned.

**Remarks**

The cursor position is set to BeforeFirst() when the application first opens a table.

## PrepareStatement method

Prepares a SQL statement.

**Overload list**

Name	Description
<a href="#">“PrepareStatement(const char *) method”</a>	Prepares a SQL statement.
<a href="#">“PrepareStatement(const ul_wchar *) method”</a>	Prepares a SQL statement.

## PrepareStatement(const char \*) method

Prepares a SQL statement.

**Syntax**

```
public virtual ULPpreparedStatement * PrepareStatement(const char * sql)
```

**Parameters**

- **sql** The SQL statement to prepare.

**Returns**

On success, the ULPpreparedStatement object. Otherwise, NULL.

## PrepareStatement(const ul\_wchar \*) method

Prepares a SQL statement.

**Syntax**

```
public virtual ULPpreparedStatement * PrepareStatement(  
    const ul_wchar * sql  
)
```

**Parameters**

- **sql** The SQL statement to prepare.

**Returns**

On success, the ULPpreparedStatement object. Otherwise, NULL.

## RegisterForEvent method

Registers or unregisters a queue to receive notifications of an event.

## Overload list

Name	Description
<a href="#">“RegisterForEvent(const char *, const char *, const char *, bool) method”</a>	Registers or unregisters a queue to receive notifications of an event.
<a href="#">“RegisterForEvent(const ul_wchar *, const ul_wchar *, const ul_wchar *, bool) method”</a>	Registers or unregisters a queue to receive notifications of an event.

## RegisterForEvent(const char \*, const char \*, const char \*, bool) method

Registers or unregisters a queue to receive notifications of an event.

### Syntax

```
public virtual bool RegisterForEvent(
    const char * eventName,
    const char * objectName,
    const char * queueName,
    bool register_not_unreg
)
```

### Parameters

- **eventName** The system or user-defined event to register for.
- **objectName** The object to which the event applies (like table name).
- **queueName** NULL means default connection queue.
- **register\_not\_unreg** True to register, false to unregister.

### Returns

True if the registration succeeded, false if the queue or event does not exist.

### Remarks

If no queue name is supplied, the default connection queue is implied, and created if required. Certain system events allow specification of an object name to which the event applies. For example, the `TableModified` event can specify the table name. Unlike `SendNotification()`, only the specific queue registered receives notifications of the event. Other queues with the same name on different connections do not, unless they are also explicit registered.

The predefined system events are:

- **TableModified** Triggered when rows in a table are inserted, updated, or deleted. One notification is sent per request, no matter how many rows were affected by the request. The `object_name` parameter

specifies the table to monitor. A value of "\*" means all tables in the database. This event has a parameter named 'table\_name' whose value is the name of the modified table.

- **Commit** Triggered after any commit completes. This event has no parameters.
- **SyncComplete** Triggered after synchronization completes. This event has no parameters.

## RegisterForEvent(const ul\_wchar \*, const ul\_wchar \*, const ul\_wchar \*, bool) method

Registers or unregisters a queue to receive notifications of an event.

### Syntax

```
public virtual bool RegisterForEvent(  
    const ul_wchar * eventName,  
    const ul_wchar * objectName,  
    const ul_wchar * queueName,  
    bool register_not_unreg  
)
```

### Parameters

- **eventName** The system or user-defined event to register for.
- **objectName** The object to which the event applies (like table name).
- **queueName** NULL means default connection queue.
- **register\_not\_unreg** True to register, false to unregister.

### Returns

True if the registration succeeded, false if the queue or event does not exist.

### Remarks

If no queue name is supplied, the default connection queue is implied, and created if required. Certain system events allow specification of an object name to which the event applies. For example, the TableModified event can specify the table name. Unlike SendNotification(), only the specific queue registered receives notifications of the event. Other queues with the same name on different connections do not, unless they are also explicit registered.

The predefined system events are:

- **TableModified** Triggered when rows in a table are inserted, updated, or deleted. One notification is sent per request, no matter how many rows were affected by the request. The *object\_name* parameter specifies the table to monitor. A value of "\*" means all tables in the database. This event has a parameter named 'table\_name' whose value is the name of the modified table.
- **Commit** Triggered after any commit completes. This event has no parameters.

- **SyncComplete** Triggered after synchronization completes. This event has no parameters.

## ResetLastDownloadTime method

Resets the time of the last download of the named publication.

### Overload list

Name	Description
<a href="#">“ResetLastDownloadTime(const char *) method”</a>	Resets the time of the last download of the named publication.
<a href="#">“ResetLastDownloadTime(const ul_wchar *) method”</a>	Resets the time of the last download of the named publication.

## ResetLastDownloadTime(const char \*) method

Resets the time of the last download of the named publication.

### Syntax

```
public virtual bool ResetLastDownloadTime(const char * pubList)
```

### Parameters

- **pubList** A comma separated list of publications to reset.

### Returns

True on success, otherwise false.

## ResetLastDownloadTime(const ul\_wchar \*) method

Resets the time of the last download of the named publication.

### Syntax

```
public virtual bool ResetLastDownloadTime(const ul_wchar * pubList)
```

### Parameters

- **pubList** A comma separated list of publications to reset.

### Returns

True on success, otherwise false.

## RevokeConnectFrom method

Deletes an existing user.

### Overload list

Name	Description
<a href="#">“RevokeConnectFrom(const char *) method”</a>	Deletes an existing user.
<a href="#">“RevokeConnectFrom(const ul_wchar *) method”</a>	Deletes an existing user.

## RevokeConnectFrom(const char \*) method

Deletes an existing user.

### Syntax

```
public virtual bool RevokeConnectFrom(const char * uid)
```

### Parameters

- **uid** The name of the user to delete.

### Returns

True on success, otherwise false.

## RevokeConnectFrom(const ul\_wchar \*) method

Deletes an existing user.

### Syntax

```
public virtual bool RevokeConnectFrom(const ul_wchar * uid)
```

### Parameters

- **uid** The name of the user to delete.

### Returns

True on success, otherwise false.

## Rollback method

Rolls back the current transaction.

**Syntax**

```
public virtual bool Rollback()
```

**Returns**

True on success, otherwise false.

## RollbackPartialDownload method

Rolls back a partial download.

**Syntax**

```
public virtual bool RollbackPartialDownload()
```

**Returns**

True on success, otherwise false.

## SendNotification method

Sends a notification to all queues matching the given name.

**Overload list**

Name	Description
“SendNotification(const char *, const char *, const char *) method”	Sends a notification to all queues matching the given name.
“SendNotification(const ul_wchar *, const ul_wchar *, const ul_wchar *) method”	Send a notification to all queues matching the given name.

## SendNotification(const char \*, const char \*, const char \*) method

Sends a notification to all queues matching the given name.

**Syntax**

```
public virtual ul_u_long SendNotification(
    const char * queueName,
    const char * eventName,
    const char * parameters
)
```

**Parameters**

- **queueName** The target queue name (or "\*").

- **eventName** The identity for notification.
- **parameters** Optional parameters option list.

### Returns

The number of notifications sent. (the number of matching queues)

### Remarks

This includes any such queue on the current connection. This call does not block. Use the special queue name "\*" to send to all queues. The given event name does not need to correspond to any system or user-defined event; it is simply passed through to identify the notification when read and has meaning only to the sender and receiver.

The *parameters* argument specifies a semi-colon delimited name=value pairs option list. After the notification is read, the parameter values are read with `GetNotificationParameter()`.

### See also

- [“GetNotificationParameter method” on page 116](#)

## SendNotification(const ul\_wchar \*, const ul\_wchar \*, const ul\_wchar \*) method

Send a notification to all queues matching the given name.

### Syntax

```
public virtual ul_u_long SendNotification(  
    const ul_wchar * queueName ,  
    const ul_wchar * eventName ,  
    const ul_wchar * parameters  
)
```

### Parameters

- **queueName** The target queue name (or "\*").
- **eventName** The identity for notification.
- **parameters** Optional parameters option list.

### Returns

The number of notifications sent. (the number of matching queues)

### Remarks

This includes any such queue on the current connection. This call does not block. Use the special queue name "\*" to send to all queues. The given event name does not need to correspond to any system or user-defined event; it is simply passed through to identify the notification when read and has meaning only to the sender and receiver.



The *parameters* argument specifies a semi-colon delimited name=value pairs option list. After the notification is read, the parameter values are read with `GetNotificationParameter()`.

### See also

- [“GetNotificationParameter method” on page 116](#)

## SetDatabaseOption method

Sets the specified database option.

### Overload list

Name	Description
<a href="#">“SetDatabaseOption(const char *, const char *) method”</a>	Sets the specified database option.
<a href="#">“SetDatabaseOption(const ul_wchar *, const ul_wchar *) method”</a>	Sets the specified database option.

### SetDatabaseOption(const char \*, const char \*) method

Sets the specified database option.

#### Syntax

```
public virtual bool SetDatabaseOption(
    const char * optName,
    const char * value
)
```

#### Parameters

- **optName** The name of the option being set.
- **value** The new value of the option.

#### Returns

True on success, otherwise false.

### SetDatabaseOption(const ul\_wchar \*, const ul\_wchar \*) method

Sets the specified database option.

#### Syntax

```
public virtual bool SetDatabaseOption(
    const ul_wchar * optName,
    const ul_wchar * value
)
```

**Parameters**

- **optName** The name of the option being set.
- **value** The new value of the option.

**Returns**

True on success, otherwise false.

## SetDatabaseOptionInt method

Sets a database option.

**Overload list**

Name	Description
“SetDatabaseOptionInt(const char *, ul_u_long) method”	Sets a database option.
“SetDatabaseOptionInt(const ul_wchar *, ul_u_long) method”	Sets a database option.

## SetDatabaseOptionInt(const char \*, ul\_u\_long) method

Sets a database option.

**Syntax**

```
public virtual bool SetDatabaseOptionInt(  
    const char * optName,  
    ul_u_long value  
)
```

**Parameters**

- **optName** The name of the option being set.
- **value** The new value of the option.

**Returns**

True on success, otherwise false.

## SetDatabaseOptionInt(const ul\_wchar \*, ul\_u\_long) method

Sets a database option.

**Syntax**

```
public virtual bool SetDatabaseOptionInt(  
    const ul_wchar * optName,
```

```

        ul_u_long value
    )

```

### Parameters

- **optName** The name of the option being set.
- **value** The new value of the option.

### Returns

True on success, otherwise false.

## SetSynchronizationCallback method

Sets the callback to be invoked while performing a synchronization.

### Syntax

```

public virtual void SetSynchronizationCallback(
    ul_sync_observer_fn callback,
    void * userData
)

```

### Parameters

- **callback** The ul\_sync\_observer\_fn callback.
- **userData** User context information passed to the callback.

## SetSyncInfo method

Creates a synchronization profile.

### Overload list

Name	Description
<a href="#">“SetSyncInfo(char const *, ul_sync_info *) method”</a>	Creates a synchronization profile.
<a href="#">“SetSyncInfo(const ul_wchar *, ul_sync_info_w2 *) method”</a>	Creates a synchronization profile.

## SetSyncInfo(char const \*, ul\_sync\_info \*) method

Creates a synchronization profile.

### Syntax

```

public virtual bool SetSyncInfo(
    char const * profileName,

```

```
        ul_sync_info * info  
    )
```

### Parameters

- **profileName** The name of the synchronization profile.
- **info** A pointer to the `ul_sync_info` structure that holds the synchronization parameters.

### Returns

True on success, otherwise false.

### Remarks

The synchronization profile replaces any previous profile with the same name. The named profile is deleted by specifying a null pointer for the structure.

## SetSyncInfo(const ul\_wchar \*, ul\_sync\_info\_w2 \*) method

Creates a synchronization profile.

### Syntax

```
public virtual bool SetSyncInfo(  
    const ul_wchar * profileName,  
    ul_sync_info_w2 * info  
)
```

### Parameters

- **profileName** The name of the synchronization profile.
- **info** A pointer to the `ul_sync_info` structure that holds the synchronization parameters.

### Returns

True on success, otherwise false.

### Remarks

The synchronization profile replaces any previous profile with the same name. The named profile is deleted by specifying a null pointer for the structure.

## StartSynchronizationDelete method

Sets START SYNCHRONIZATION DELETE for this connection.

### Syntax

```
public virtual bool StartSynchronizationDelete()
```

**Returns**

True on success, otherwise false.

## StopSynchronizationDelete method

Sets STOP SYNCHRONIZATION DELETE for this connection.

**Syntax**

```
public virtual bool StopSynchronizationDelete()
```

**Returns**

True on success, otherwise false.

## Synchronize method

Synchronizes the database.

**Overload list**

Name	Description
<a href="#">“Synchronize(ul_sync_info *) method”</a>	Synchronizes the database.
<a href="#">“Synchronize(ul_sync_info_w2 *) method”</a>	Synchronizes the database.

## Synchronize(ul\_sync\_info \*) method

Synchronizes the database.

**Syntax**

```
public virtual bool Synchronize(ul_sync_info * info)
```

**Parameters**

- **info** A pointer to the ul\_sync\_info structure that holds the synchronization parameters.

**Returns**

True on success, otherwise false.

**Remarks**

The following example demonstrates database synchronization:

```
ul_sync_info info;
conn->InitSyncInfo( &info );
info.user_name = UL_TEXT( "user_name" );
```

```
info.version = UL_TEXT( "test" );  
conn->Synchronize( &info );
```

## Synchronize(*ul\_sync\_info\_w2* \*) method

Synchronizes the database.

### Syntax

```
public virtual bool Synchronize(ul_sync_info_w2 * info)
```

### Parameters

- **info** A pointer to the *ul\_sync\_info* structure that holds the synchronization parameters.

### Returns

True on success, otherwise false.

### Remarks

The following example demonstrates database synchronization:

```
ul_sync_info info;  
conn->InitSyncInfo( &info );  
info.user_name = UL_TEXT( "user_name" );  
info.version = UL_TEXT( "test" );  
conn->Synchronize( &info );
```

## SynchronizeFromProfile method

Synchronize the database using the given profile and merge parameters.

### Overload list

Name	Description
“SynchronizeFromProfile(const char *, const char *, <i>ul_sync_observer_fn</i> , void *) method”	Synchronize the database using the given profile and merge parameters.
“SynchronizeFromProfile(const <i>ul_wchar</i> *, const <i>ul_wchar</i> *, <i>ul_sync_observer_fn</i> , void *) method”	Synchronize the database using the given profile and merge parameters.

## SynchronizeFromProfile(const char \*, const char \*, *ul\_sync\_observer\_fn*, void \*) method

Synchronize the database using the given profile and merge parameters.

**Syntax**

```
public virtual bool SynchronizeFromProfile(
    const char * profileName,
    const char * mergeParms,
    ul_sync_observer_fn observer,
    void * userData
)
```

**Parameters**

- **profileName** Name of the profile to synchronize.
- **mergeParms** Merge parameters for the synchronization.
- **observer** Observer callback to send status updates to.
- **userData** User context data passed to callback.

**Returns**

True on success, otherwise false.

**Remarks**

This is identical to executing the SYNCHRONIZE statement.

## **SynchronizeFromProfile(const ul\_wchar \*, const ul\_wchar \*, ul\_sync\_observer\_fn, void \*) method**

Synchronize the database using the given profile and merge parameters.

**Syntax**

```
public virtual bool SynchronizeFromProfile(
    const ul_wchar * profileName,
    const ul_wchar * mergeParms,
    ul_sync_observer_fn observer,
    void * userData
)
```

**Parameters**

- **profileName** Name of the profile to synchronize.
- **mergeParms** Merge parameters for the synchronization.
- **observer** Observer callback to send status updates to.
- **userData** User context data passed to callback.

**Returns**

True on success, otherwise false.

**Remarks**

This is identical to executing the SYNCHRONIZE statement.

## TriggerEvent method

Triggers a user-defined event and sends notifications to all registered queues.

**Overload list**

Name	Description
<a href="#">“TriggerEvent(const char *, const char *) method”</a>	Triggers a user-defined event and sends notifications to all registered queues.
<a href="#">“TriggerEvent(const ul_wchar *, const ul_wchar *) method”</a>	Triggers a user-defined event and sends notifications to all registered queues.

## TriggerEvent(const char \*, const char \*) method

Triggers a user-defined event and sends notifications to all registered queues.

**Syntax**

```
public virtual ul_u_long TriggerEvent(  
    const char * eventName,  
    const char * parameters  
)
```

**Parameters**

- **eventName** The name of the system or user-defined event to trigger.
- **parameters** Optional parameters option list.

**Returns**

The number of event notifications sent.

**Remarks**

The *parameters* value specifies a semi-colon delimited name=value pairs option list. After the notification is read, the parameter values are read with `GetNotificationParameter()`.

**See also**

- [“GetNotificationParameter method” on page 116](#)



## TriggerEvent(const ul\_wchar \*, const ul\_wchar \*) method

Triggers a user-defined event and sends notifications to all registered queues.

### Syntax

```
public virtual ul_u_long TriggerEvent(
    const ul_wchar * eventName,
    const ul_wchar * parameters
)
```

### Parameters

- **eventName** The name of the system or user-defined event to trigger.
- **parameters** Optional parameters option list.

### Returns

The number of event notifications sent.

### Remarks

The *parameters* value specifies a semi-colon delimited name=value pairs option list. After the notification is read, the parameter values are read with GetNotificationParameter().

### See also

- [“GetNotificationParameter method” on page 116](#)

## ValidateDatabase method

Validates the database on this connection.

### Overload list

Name	Description
“ValidateDatabase(ul_u_short, ul_validate_callback_fn, void *, const char *) method”	Validates the database on this connection.
“ValidateDatabase(ul_u_short, ul_validate_callback_fn, void *, const ul_wchar *) method”	Validates the database on this connection.

## ValidateDatabase(ul\_u\_short, ul\_validate\_callback\_fn, void \*, const char \*) method

Validates the database on this connection.

## Syntax

```
public virtual bool ValidateDatabase(  
    ul_u_short flags,  
    ul_validate_callback_fn fn,  
    void * user_data,  
    const char * tableName  
)
```

## Parameters

- **flags** Flags controlling the type of validation; see below.
- **fn** Function to receive validation progress information.
- **user\_data** User data to send back to the caller via the callback.
- **tableName** Optionally, a specific table to validate.

## Returns

True on success, otherwise false.

## Remarks

Depending on the flags passed to this routine, the low level store and/or the indexes can be validated. To receive information during the validation, implement a callback function and pass the address to this routine. To limit the validation to a specific table, pass in the table name or ID as the last parameter.

The flags parameter is combination of the following values:

- **#ULVF\_TABLE** For more information, see [“ULVF\\_TABLE variable” on page 90](#).
- **#ULVF\_INDEX** For more information, see [“ULVF\\_INDEX variable” on page 90](#).
- **#ULVF\_DATABASE** For more information, see [“ULVF\\_DATABASE variable” on page 89](#).
- **#ULVF\_EXPRESS** For more information, see [“ULVF\\_EXPRESS variable” on page 89](#).
- **#ULVF\_FULL\_VALIDATE** For more information, see [“ULVF\\_FULL\\_VALIDATE variable” on page 89](#).

## Example

```
The following example demonstrates table and index validation in express  
mode:  
flags = ULVF_TABLE | ULVF_INDEX | ULVF_EXPRESS;
```

## **ValidateDatabase(ul\_u\_short, ul\_validate\_callback\_fn, void \*, const ul\_wchar \*) method**

Validates the database on this connection.

**Syntax**

```
public virtual bool ValidateDatabase(
    ul_u_short flags,
    ul_validate_callback_fn fn,
    void * user_data,
    const ul_wchar * tableName
)
```

**Parameters**

- **flags** Flags controlling the type of validation; see below.
- **fn** Function to receive validation progress information.
- **user\_data** User data to send back to the caller via the callback.
- **tableName** Optionally, a specific table to validate.

**Returns**

True on success, otherwise false.

**Remarks**

Depending on the flags passed to this routine, the low level store and/or the indexes can be validated. To receive information during the validation, implement a callback function and pass the address to this routine. To limit the validation to a specific table, pass in the table name or ID as the last parameter.

The flags parameter is combination of the following values:

- **#ULVF\_TABLE** For more information, see [“ULVF\\_TABLE variable” on page 90](#).
- **#ULVF\_INDEX** For more information, see [“ULVF\\_INDEX variable” on page 90](#).
- **#ULVF\_DATABASE** For more information, see [“ULVF\\_DATABASE variable” on page 89](#).
- **#ULVF\_EXPRESS** For more information, see [“ULVF\\_EXPRESS variable” on page 89](#).
- **#ULVF\_FULL\_VALIDATE** For more information, see [“ULVF\\_FULL\\_VALIDATE variable” on page 89](#).

## ULDatabaseManager class

Manages connections and databases.

**Syntax**

```
public class ULDatabaseManager
```

**Members**

All members of ULDatabaseManager class, including all inherited members.

Name	Description
“CreateDatabase method”	Creates a new database.
“DropDatabase method”	Erases an existing database that is not currently running.
“EnableAesDBEncryption method”	Enables AES database encryption.
“EnableAesFipsDBEncryption method”	Enables FIPS 140-2 certified AES database encryption.
“EnableEccE2ee method”	Enables ECC end-to-end encryption.
“EnableEccSyncEncryption method”	Enables ECC synchronization encryption.
“EnableHttpsSynchronization method”	Enables HTTPS synchronization.
“EnableHttpSynchronization method”	Enables HTTP synchronization.
“EnableRsaE2ee method”	Enables RSA end-to-end encryption.
“EnableRsaFipsE2ee method”	Enables FIPS 140-2 certified RSA end-to-end encryption.
“EnableRsaFipsSyncEncryption method”	Enables FIPS 140-2 certified RSA synchronization encryption.
“EnableRsaSyncEncryption method”	Enables RSA synchronization encryption.
“EnableTcpipSynchronization method”	Enables TCP/IP synchronization.
“EnableTlsSynchronization method”	Enables TLS synchronization.
“EnableZlibSyncCompression method”	Enables zlib synchronization compression.
“Fini method”	Finalizes the UltraLite runtime.
“Init method”	Initializes the UltraLite runtime.
“OpenConnection method”	Opens a new connection to an existing database.
“SetErrorCallback method”	Sets the callback to be invoked when an error occurs.
“ValidateDatabase method”	Performs low level and index validation on a database.

### Remarks

The Init method must be called in a thread-safe environment before any other class can be made. The Fini method must be called in a similarly thread-safe environment when finished.

**Note**

This class is static. Do not create an instance of it.

## CreateDatabase method

Creates a new database.

### Overload list

Name	Description
<a href="#">“CreateDatabase(const char *, const char *, ULError *) method”</a>	Creates a new database.
<a href="#">“CreateDatabase(const ul_wchar *, const ul_wchar *, ULError *) method”</a>	Creates a new database.

## CreateDatabase(const char \*, const char \*, ULError \*) method

Creates a new database.

### Syntax

```
public static ULConnection * CreateDatabase(
    const char * connParms,
    const char * createParms,
    ULError * error
)
```

### Parameters

- **connParms** The parameters used to connect to the new database.
- **createParms** The parameters used to create the database.
- **error** An optional ULError object to receive error information.

### Returns

A ULConnection object to the new database is returned if the function succeeds. NULL is returned if the function fails.

### Remarks

For more information about UltraLite connection parameters, see [“UltraLite connection parameters” \[UltraLite - Database Management and Reference\]](#).

For more information about UltraLite creation parameters, see [“UltraLite creation parameters” \[UltraLite - Database Management and Reference\]](#).

## CreateDatabase(const ul\_wchar \*, const ul\_wchar \*, ULError \*) method

Creates a new database.

### Syntax

```
public static ULConnection * CreateDatabase(  
    const ul_wchar * connParms,  
    const ul_wchar * createParms,  
    ULError * error  
)
```

### Parameters

- **connParms** The parameters used to connect to the new database.
- **createParms** The parameters used to create the database.
- **error** An optional ULError object to receive error information.

### Returns

A ULConnection object to the new database is returned if the function succeeds. NULL is returned if the function fails.

### Remarks

For more information about UltraLite connection parameters, see [“UltraLite connection parameters” \[UltraLite - Database Management and Reference\]](#).

For more information about UltraLite creation parameters, see [“UltraLite creation parameters” \[UltraLite - Database Management and Reference\]](#).

## DropDatabase method

Erases an existing database that is not currently running.

### Overload list

Name	Description
<a href="#">“DropDatabase(const char *, ULError *) method”</a>	Erases an existing database that is not currently running.
<a href="#">“DropDatabase(const ul_wchar *, ULError *) method”</a>	Erases an existing database that is not currently running.

## DropDatabase(const char \*, ULError \*) method

Erases an existing database that is not currently running.

### Syntax

```
public static bool DropDatabase(const char * parms, ULError * error)
```

### Parameters

- **parms** The database identification parameters.
- **error** An optional ULError object to receive error information.

### Returns

True if the database was successfully deleted; otherwise, false is returned.

## DropDatabase(const ul\_wchar \*, ULError \*) method

Erases an existing database that is not currently running.

### Syntax

```
public static bool DropDatabase(const ul_wchar * parms, ULError * error)
```

### Parameters

- **parms** The database identification parameters.
- **error** An optional ULError object to receive error information.

### Returns

True if the database was successfully deleted; otherwise, false is returned.

## EnableAesDBEncryption method

Enables AES database encryption.

### Syntax

```
public static void EnableAesDBEncryption()
```

## EnableAesFipsDBEncryption method

Enables FIPS 140-2 certified AES database encryption.

### Syntax

```
public static void EnableAesFipsDBEncryption()
```

## EnableEccE2ee method

Enables ECC end-to-end encryption.

### Syntax

```
public static void EnableEccE2ee()
```

## EnableEccSyncEncryption method

Enables ECC synchronization encryption.

### Syntax

```
public static void EnableEccSyncEncryption()
```

## EnableHttpsSynchronization method

Enables HTTPS synchronization.

### Syntax

```
public static void EnableHttpsSynchronization()
```

## EnableHttpSynchronization method

Enables HTTP synchronization.

### Syntax

```
public static void EnableHttpSynchronization()
```

## EnableRsaE2ee method

Enables RSA end-to-end encryption.

### Syntax

```
public static void EnableRsaE2ee()
```

## EnableRsaFipsE2ee method

Enables FIPS 140-2 certified RSA end-to-end encryption.



**Syntax**

```
public static void EnableRsaFipsE2ee()
```

## EnableRsaFipsSyncEncryption method

Enables FIPS 140-2 certified RSA synchronization encryption.

**Syntax**

```
public static void EnableRsaFipsSyncEncryption()
```

## EnableRsaSyncEncryption method

Enables RSA synchronization encryption.

**Syntax**

```
public static void EnableRsaSyncEncryption()
```

## EnableTcpipSynchronization method

Enables TCP/IP synchronization.

**Syntax**

```
public static void EnableTcpipSynchronization()
```

## EnableTlsSynchronization method

Enables TLS synchronization.

**Syntax**

```
public static void EnableTlsSynchronization()
```

## EnableZlibSyncCompression method

Enables zlib synchronization compression.

**Syntax**

```
public static void EnableZlibSyncCompression()
```

## Fini method

Finalizes the UltraLite runtime.

### Syntax

```
public static void Fini()
```

### Remarks

This method must be called only once by a single thread when the application is finished; this method is not thread-safe.

## Init method

Initializes the UltraLite runtime.

### Syntax

```
public static bool Init()
```

### Returns

True on success; otherwise, false is returned. False can also be returned if the function is called more than once.

### Remarks

This method must be called only once by a single thread before any other calls can be made; this method is not thread-safe.

This method does not usually fail unless memory is unavailable.

## OpenConnection method

Opens a new connection to an existing database.

### Overload list

Name	Description
<code>“OpenConnection(const char *, ULError *, void *) method”</code>	Opens a new connection to an existing database.
<code>“OpenConnection(const ul_wchar *, ULError *, void *) method”</code>	Opens a new connection to an existing database.

## OpenConnection(const char \*, ULError \*, void \*) method

Opens a new connection to an existing database.

### Syntax

```
public static ULConnection * OpenConnection(  
    const char * connParms,  
    ULError * error,  
    void * reserved  
)
```

### Parameters

- **connParms** The connection string.
- **error** An optional ULError object to return error information.
- **reserved** Reserved for internal use.

### Returns

A new ULConnection if the function succeeds. NULL is returned if the function fails.

### Remarks

To get error information, pass in a pointer to a ULError object. The following is a list of possible errors:

- **SQLC\_INVALID\_PARSE\_PARAMETER** *connParms* was not formatted properly.
- **SQLC\_UNRECOGNIZED\_OPTION** A connection option name was likely misspelled.
- **SQLC\_INVALID\_OPTION\_VALUE** A connection option value was not specified properly.
- **SQLC\_ULTRALITE\_DATABASE\_NOT\_FOUND** The specified database could not be found.
- **SQLC\_INVALID\_LOGON** You supplied an invalid user ID or an incorrect password.
- **SQLC\_TOO\_MANY\_CONNECTIONS** You exceeded the maximum number of concurrent database connections.

For more information about UltraLite connection parameters, see [“UltraLite connection parameters”](#) [*UltraLite - Database Management and Reference*].

## OpenConnection(const ul\_wchar \*, ULError \*, void \*) method

Opens a new connection to an existing database.

### Syntax

```
public static ULConnection * OpenConnection(  
    const ul_wchar * connParms,  
    ULError * error,  
    void * reserved  
)
```

**Parameters**

- **connParms** The connection string.
- **error** An optional ULError object to return error information.
- **reserved** Reserved for internal use.

**Returns**

A new ULConnection if the function succeeds. NULL is returned if the function fails.

**Remarks**

To get error information, pass in a pointer to a ULError object. The following is a list of possible errors:

- **SQLE\_INVALID\_PARSE\_PARAMETER** *connParms* was not formatted properly.
- **SQLE\_UNRECOGNIZED\_OPTION** A connection option name was likely misspelled.
- **SQLE\_INVALID\_OPTION\_VALUE** A connection option value was not specified properly.
- **SQLE\_ULTRALITE\_DATABASE\_NOT\_FOUND** The specified database could not be found.
- **SQLE\_INVALID\_LOGON** You supplied an invalid user ID or an incorrect password.
- **SQLE\_TOO\_MANY\_CONNECTIONS** You exceeded the maximum number of concurrent database connections.

For more information about UltraLite connection parameters, see [“UltraLite connection parameters” \[UltraLite - Database Management and Reference\]](#).

## SetErrorCallback method

Sets the callback to be invoked when an error occurs.

**Syntax**

```
public static void SetErrorCallback(  
    ul_cpp_error_callback_fn callback,  
    void * userData  
)
```

**Parameters**

- **callback** The callback function.
- **userData** User context information passed to the callback.

**Remarks**

This method is not thread-safe.

## ValidateDatabase method

Performs low level and index validation on a database.

### Overload list

Name	Description
“ValidateDatabase(const char *, ul_u_short, ul_validate_callback_fn, void *, ULError *) method”	Performs low level and index validation on a database.
“ValidateDatabase(const ul_wchar *, ul_u_short, ul_validate_callback_fn, void *, ULError *) method”	Performs low level and index validation on a database.

## ValidateDatabase(const char \*, ul\_u\_short, ul\_validate\_callback\_fn, void \*, ULError \*) method

Performs low level and index validation on a database.

### Syntax

```
public static bool ValidateDatabase(
    const char * connParms,
    ul_u_short flags,
    ul_validate_callback_fn fn,
    void * userData,
    ULError * error
)
```

### Parameters

- **connParms** The parameters used to connect to the database.
- **flags** The flags controlling the type of validation; see below.
- **fn** A function to receive validation progress information.
- **userData** The user data to send back to the caller via the callback.
- **error** An optional ULError object to receive error information.

### Returns

True if the validation succeeds. False otherwise.

### Remarks

The flags parameter is combination of the following values:

- **#ULVF\_TABLE** For more information, see “[ULVF\\_TABLE variable](#)” on page 90.

- **#ULVF\_INDEX** For more information, see [“ULVF\\_INDEX variable” on page 90](#).
- **#ULVF\_DATABASE** For more information, see [“ULVF\\_DATABASE variable” on page 89](#).
- **#ULVF\_EXPRESS** For more information, see [“ULVF\\_EXPRESS variable” on page 89](#).
- **#ULVF\_FULL\_VALIDATE** For more information, see [“ULVF\\_FULL\\_VALIDATE variable” on page 89](#).

### Example

The following example demonstrates table and index validation in express mode:

```
flags = ULVF_TABLE | ULVF_INDEX | ULVF_EXPRESS;
```

## ValidateDatabase(const ul\_wchar \*, ul\_u\_short, ul\_validate\_callback\_fn, void \*, ULError \*) method

Performs low level and index validation on a database.

### Syntax

```
public static bool ValidateDatabase(  
    const ul_wchar * connParms,  
    ul_u_short flags,  
    ul_validate_callback_fn fn,  
    void * userData,  
    ULError * error  
)
```

### Parameters

- **connParms** The parameters used to connect to the database.
- **flags** The flags controlling the type of validation; see below.
- **fn** A function to receive validation progress information.
- **userData** The user data to send back to the caller via the callback.
- **error** An optional ULError object to receive error information.

### Returns

True if the validation succeeds. False otherwise.

### Remarks

The flags parameter is combination of the following values:

- **#ULVF\_TABLE** For more information, see [“ULVF\\_TABLE variable” on page 90](#).
- **#ULVF\_INDEX** For more information, see [“ULVF\\_INDEX variable” on page 90](#).

- **#ULVF\_DATABASE** For more information, see [“ULVF\\_DATABASE variable” on page 89](#).
- **#ULVF\_EXPRESS** For more information, see [“ULVF\\_EXPRESS variable” on page 89](#).
- **#ULVF\_FULL\_VALIDATE** For more information, see [“ULVF\\_FULL\\_VALIDATE variable” on page 89](#).

## ULDatabaseSchema class

Represents the schema of an UltraLite database.

### Syntax

```
public class ULDatabaseSchema
```

### Members

All members of ULDatabaseSchema class, including all inherited members.

Name	Description
<a href="#">“Close method”</a>	Destroys this object.
<a href="#">“GetConnection method”</a>	Gets the connection object.
<a href="#">“GetNextPublication method”</a>	Gets the name of the next publication in the database.
<a href="#">“GetNextPublicationW2 method”</a>	Gets the name of the next publication in the database.
<a href="#">“GetNextTable method”</a>	Gets the next table (schema) in the database.
<a href="#">“GetPublicationCount method”</a>	Gets the number of publications in the database.
<a href="#">“GetTableCount method”</a>	Returns the number of tables in the database.
<a href="#">“GetTableSchema method”</a>	Returns the schema of the named table.

## Close method

Destroys this object.

### Syntax

```
public virtual void Close()
```

## GetConnection method

Gets the connection object.

**Syntax**

```
public virtual ULConnection * GetConnection()
```

**Returns**

The ULConnection associated with this object.

## GetNextPublication method

Gets the name of the next publication in the database.

**Syntax**

```
public virtual const char * GetNextPublication(  
    ul_publication_iter * iter  
)
```

**Parameters**

- **iter** Pointer to iterator variable.

**Returns**

The name of the next publication. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while. NULL is returned when the iteration is complete.

**Remarks**

Initialize the value in *iter* to `ul_publication_iter_start` before the first call.

**See also**

- [“ul\\_publication\\_iter\\_start variable” on page 257](#)

## GetNextPublicationW2 method

Gets the name of the next publication in the database.

**Syntax**

```
public virtual const ul_wchar * GetNextPublicationW2(  
    ul_publication_iter * iter  
)
```

**Parameters**

- **iter** Pointer to iterator variable.



**Returns**

The name of the next publication. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while. NULL is returned when the iteration is complete.

**Remarks**

Initialize the value in *iter* to `ul_publication_iter_start` before the first call.

**See also**

- [“ul\\_publication\\_iter\\_start variable” on page 257](#)

## GetNextTable method

Gets the next table (schema) in the database.

**Syntax**

```
public virtual ULTableSchema * GetNextTable(ul_table_iter * iter)
```

**Parameters**

- **iter** Pointer to iterator variable.

**Returns**

A ULTableSchema object or NULL when the iteration is complete.

**Remarks**

Initialize the value in *iter* to `ul_table_iter_start` before the first call.

**See also**

- [“ul\\_table\\_iter\\_start variable” on page 257](#)

## GetPublicationCount method

Gets the number of publications in the database.

**Syntax**

```
public virtual ul_publication_count GetPublicationCount()
```

**Returns**

The number of publications in the database.

**Remarks**

Publication IDs range from 1 to `GetPublicationCount()`

## GetTableCount method

Returns the number of tables in the database.

### Syntax

```
public virtual ul_table_num GetTableCount()
```

### Returns

An integer that represents the number of tables.

## GetTableSchema method

Returns the schema of the named table.

### Overload list

Name	Description
<a href="#">“GetTableSchema(const char *) method”</a>	Returns the schema of the named table.
<a href="#">“GetTableSchema(const ul_wchar *) method”</a>	Returns the schema of the named table.

## GetTableSchema(const char \*) method

Returns the schema of the named table.

### Syntax

```
public virtual ULTableSchema * GetTableSchema(const char * tableName)
```

### Parameters

- **tableName** The name of the table.

### Returns

A ULTableSchema object for the given table. UL\_NULL if the table does not exist.

## GetTableSchema(const ul\_wchar \*) method

Returns the schema of the named table.

### Syntax

```
public virtual ULTableSchema * GetTableSchema(  
    const ul_wchar * tableName  
)
```

**Parameters**

- **tableName** The name of the table.

**Returns**

A ULTableSchema object for the given table. UL\_NULL if the table does not exist.

## ULError class

Manages the errors returned from the UltraLite runtime.

**Syntax**

```
public class ULError
```

**Members**

All members of ULError class, including all inherited members.

Name	Description
“ULError constructor”	Constructs a ULError class.
“Clear method”	Clears the current error.
“GetErrorInfo method”	Gets a pointer to the underlying ul_error_info object.
“GetParameter method”	Copies the specified error parameter into the provided buffer.
“GetParameterCount method”	Gets the number of error parameters.
“GetSQLCode method”	Gets the error code (SQLCODE) for the last operation.
“GetSQLCount method”	Returns a value that depends on the result of the last operation.
“GetString method”	Gets a description of the current error.
“GetURL method”	Gets a URL to the documentation page for this error.
“IsOK method”	Tests the error code.

## ULError constructor

Constructs a ULError class.

**Syntax**

```
public ULError()
```

## Clear method

Clears the current error.

### Syntax

```
public void Clear()
```

### Remarks

The current error is cleared automatically on most calls, so this is not normally called by applications.

## GetErrorInfo method

Gets a pointer to the underlying `ul_error_info` object.

### Overload list

Name	Description
<a href="#">“GetErrorInfo() method”</a>	Gets a pointer to the underlying <code>ul_error_info</code> object.
<a href="#">“GetErrorInfo() method”</a>	Gets a pointer to the underlying <code>ul_error_info</code> object.

## GetErrorInfo() method

Gets a pointer to the underlying `ul_error_info` object.

### Syntax

```
public const ul_error_info * GetErrorInfo()
```

### Returns

A pointer to the underlying `ul_error_info` object.

### Remarks

For more information about the `ul_error_info` structure, see [“ul\\_error\\_info structure”](#) on page 80.

## GetErrorInfo() method

Gets a pointer to the underlying `ul_error_info` object.

### Syntax

```
public ul_error_info * GetErrorInfo()
```

**Returns**

A pointer to the underlying `ul_error_info` object.

**Remarks**

For more information about the `ul_error_info` structure, see “[ul\\_error\\_info structure](#)” on page 80.

## GetParameter method

Copies the specified error parameter into the provided buffer.

**Overload list**

Name	Description
<a href="#">“GetParameter(<code>ul_u_short</code>, <code>char *</code>, <code>size_t</code>) method”</a>	Copies the specified error parameter into the provided buffer.
<a href="#">“GetParameter(<code>ul_u_short</code>, <code>ul_wchar *</code>, <code>size_t</code>) method”</a>	Copies the specified error parameter into the provided buffer.

## GetParameter(`ul_u_short`, `char *`, `size_t`) method

Copies the specified error parameter into the provided buffer.

**Syntax**

```
public size_t GetParameter(ul_u_short parmNo, char * dst, size_t len)
```

**Parameters**

- **parmNo** A 1-based parameter number.
- **dst** The buffer to receive the parameter.
- **len** The size of the buffer.

**Returns**

The size required to store the parameter, or zero if the ordinal is not valid. The parameter is truncated if the return value is larger than *len*.

**Remarks**

The output string is always null-terminated, even if the buffer is too small and the string is truncated.

## GetParameter(`ul_u_short`, `ul_wchar *`, `size_t`) method

Copies the specified error parameter into the provided buffer.

**Syntax**

```
public size_t GetParameter(
    ul_u_short parmNo,
    ul_wchar * dst,
    size_t len
)
```

**Parameters**

- **parmNo** A 1-based parameter number.
- **dst** The buffer to receive the parameter.
- **len** The size of the buffer.

**Returns**

The size required to store the parameter, or zero if the ordinal is not valid. The parameter is truncated if the return value is larger than *len*.

**Remarks**

The output string is always null-terminated, even if the buffer is too small and the string is truncated.

## GetParameterCount method

Gets the number of error parameters.

**Syntax**

```
public ul_u_short GetParameterCount()
```

**Returns**

The number of error parameters.

## GetSQLCode method

Gets the error code (SQLCODE) for the last operation.

**Syntax**

```
public an_sql_code GetSQLCode()
```

**Returns**

The sqlcode value.

## GetSQLCount method

Returns a value that depends on the result of the last operation.

### Syntax

```
public ul_s_long GetSQLCount()
```

### Returns

The number of rows affected by the statement after an INSERT, UPDATE, or DELETE statement is executed. The return value is the offset into the associated dynamic SQL statement that corresponds to the error if a SQLE\_SYNTAX\_ERROR occurs.

## GetString method

Gets a description of the current error.

### Overload list

Name	Description
<a href="#">“GetString(char *, size_t) method”</a>	Gets a description of the current error.
<a href="#">“GetString(ul_wchar *, size_t) method”</a>	Gets a description of the current error and returns the size.

## GetString(char \*, size\_t) method

Gets a description of the current error.

### Syntax

```
public size_t GetString(char * dst, size_t len)
```

### Parameters

- **dst** The buffer to receive the error description.
- **len** The size, in bytes, of the buffer.

### Returns

The size required to store the string. The string is truncated when the return value is larger than *len*.

## GetString(ul\_wchar \*, size\_t) method

Gets a description of the current error and returns the size.

### Syntax

```
public size_t GetString(ul_wchar * dst, size_t len)
```

**Parameters**

- **dst** The buffer to receive the error description.
- **len** The size, in bytes, of the buffer.

**Returns**

The size required to store the string. The string is truncated when the return value is larger than *len*.

## GetURL method

Gets a URL to the documentation page for this error.

**Overload list**

Name	Description
<a href="#">“GetURL(char *, size_t, const char *) method”</a>	Gets a URL to the documentation page for this error.
<a href="#">“GetURL(ul_wchar *, size_t, const ul_wchar *) method”</a>	Gets a URL to the documentation page for this error.

## GetURL(char \*, size\_t, const char \*) method

Gets a URL to the documentation page for this error.

**Syntax**

```
public size_t GetURL(char * buffer, size_t len, const char * reserved)
```

**Parameters**

- **buffer** The buffer to receive the URL.
- **len** The size of the buffer.
- **reserved** Reserved for future use.

**Returns**

The size required to store the URL. The URL is truncated if the return value is larger is larger than *len*.

## GetURL(ul\_wchar \*, size\_t, const ul\_wchar \*) method

Gets a URL to the documentation page for this error.

**Syntax**

```
public size_t GetURL(  
    ul_wchar * buffer,
```



```

    size_t len,
    const ul_wchar * reserved
)

```

### Parameters

- **buffer** The buffer to receive the URL.
- **len** The size of the buffer.
- **reserved** Reserved for future use.

### Returns

The size required to store the URL. The URL is truncated if the return value is larger than *len*.

## IsOK method

Tests the error code.

### Syntax

```
public bool IsOK()
```

### Returns

True if the current code is `SQLE_NOERROR` or a warning; otherwise, false is returned if the current code indicates an error.

## ULIndexSchema class

Represents the schema of an UltraLite table index.

### Syntax

```
public class ULIndexSchema
```

### Members

All members of `ULIndexSchema` class, including all inherited members.

Name	Description
<a href="#">"Close method"</a>	Destroys this object.
<a href="#">"GetColumnCount method"</a>	Gets the number of columns in the index.
<a href="#">"GetColumnName method"</a>	Gets the name of the column given the position of the column in the index.

Name	Description
“GetColumnNameW2 method”	Gets the name of the column given the position of the column in the index.
“GetConnection method”	Gets the connection object.
“GetIndexColumnID method”	Gets the 1-based index column ID from its name.
“GetIndexFlags method”	Gets the index property flags bit field.
“GetName method”	Gets the name of the index.
“GetNameW2 method”	Gets the name of the index.
“GetReferencedIndexName method”	Gets the associated primary index name.
“GetReferencedIndexNameW2 method”	Gets the associated primary index name.
“GetReferencedTableName method”	Gets the associated primary table name.
“GetReferencedTableNameW2 method”	Gets the associated primary table name.
“GetTableName method”	Gets the name of the table containing this index.
“GetTableNameW2 method”	Gets the name of the table containing this index.
“IsColumnDescending method”	Determines if the column is in descending order.

## Close method

Destroys this object.

### Syntax

```
public virtual void Close()
```

## GetColumnCount method

Gets the number of columns in the index.

### Syntax

```
public virtual ul_column_num GetColumnCount()
```

### Returns

The number of columns in the index.

---

## GetColumnName method

Gets the name of the column given the position of the column in the index.

### Syntax

```
public virtual const char * GetColumnName(ul_column_num col_id_in_index)
```

### Parameters

- **col\_id\_in\_index** The 1-based ordinal number indicating the position of the column in the index.

### Returns

The name of the column. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetColumnNameW2 method

Gets the name of the column given the position of the column in the index.

### Syntax

```
public virtual const ul_wchar * GetColumnNameW2(  
    ul_column_num col_id_in_index  
)
```

### Parameters

- **col\_id\_in\_index** The 1-based ordinal number indicating the position of the column in the index.

### Returns

The name of the column. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetConnection method

Gets the connection object.

### Syntax

```
public virtual ULConnection * GetConnection()
```

### Returns

The ULConnection associated with this object.

## GetIndexColumnID method

Gets the 1-based index column ID from its name.

### Overload list

Name	Description
<a href="#">“GetIndexColumnID(const char *) method”</a>	Gets the 1-based index column ID from its name.
<a href="#">“GetIndexColumnID(const ul_wchar *) method”</a>	Gets the 1-based index column ID from its name.

## GetIndexColumnID(const char \*) method

Gets the 1-based index column ID from its name.

### Syntax

```
public virtual ul_column_num GetIndexColumnID(const char * columnName)
```

### Parameters

- **columnName** The column name.

### Returns

0 if the column does not exist. Sets `SQL_E_COLUMN_NOT_FOUND` if the column name does not exist.

## GetIndexColumnID(const ul\_wchar \*) method

Gets the 1-based index column ID from its name.

### Syntax

```
public virtual ul_column_num GetIndexColumnID(  
    const ul_wchar * columnName  
)
```

### Parameters

- **columnName** The column name.

### Returns

0 if the column does not exist. Sets `SQL_E_COLUMN_NOT_FOUND` if the column name does not exist.

## GetIndexFlags method

Gets the index property flags bit field.

**Syntax**

```
public virtual ul_index_flag GetIndexFlags()
```

**See also**

- [“ul\\_index\\_flag enumeration” on page 255](#)

## GetName method

Gets the name of the index.

**Syntax**

```
public virtual const char * GetName()
```

**Returns**

The name of the index. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetNameW2 method

Gets the name of the index.

**Syntax**

```
public virtual const ul_wchar * GetNameW2()
```

**Returns**

The name of the index. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetReferencedIndexName method

Gets the associated primary index name.

**Syntax**

```
public virtual const char * GetReferencedIndexName()
```

**Returns**

The name of the referenced index. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you call to keep it for a while.

**Remarks**

This function is for foreign keys only.

## GetReferencedIndexNameW2 method

Gets the associated primary index name.

### Syntax

```
public virtual const ul_wchar * GetReferencedIndexNameW2()
```

### Returns

The name of the referenced index. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you call to keep it for a while.

### Remarks

This function is for foreign keys only.

## GetReferencedTableName method

Gets the associated primary table name.

### Syntax

```
public virtual const char * GetReferencedTableName()
```

### Returns

The name of the referenced table. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

### Remarks

This method is for foreign keys only.

## GetReferencedTableNameW2 method

Gets the associated primary table name.

### Syntax

```
public virtual const ul_wchar * GetReferencedTableNameW2()
```

### Returns

The name of the referenced table. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

### Remarks

This method is for foreign keys only.

## GetTableName method

Gets the name of the table containing this index.

### Syntax

```
public virtual const char * GetTableName()
```

### Returns

The name of the table containing this index. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetTableNameW2 method

Gets the name of the table containing this index.

### Syntax

```
public virtual const ul_wchar * GetTableNameW2()
```

### Returns

The name of the table containing this index. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## IsColumnDescending method

Determines if the column is in descending order.

### Syntax

```
public virtual bool IsColumnDescending(ul_column_num cid)
```

### Parameters

- **cid** The 1-based ordinal column number.

### Returns

True if the column is in descending order; false, otherwise.

## ULPreparedStatement class

Represents a prepared SQL statement.

**Syntax**

```
public class ULPreparedStatement
```

**Members**

All members of ULPreparedStatement class, including all inherited members.

<b>Name</b>	<b>Description</b>
“AppendParameterByteChunk method”	Sets a large binary parameter broken down into several chunks.
“AppendParameterStringChunk method”	Sets a large string parameter broken down into several chunks.
“Close method”	Destroys this object.
“ExecuteQuery method”	Executes a SQL SELECT statement as a query.
“ExecuteStatement method”	Executes a statement that does not return a result set, such as a SQL INSERT, DELETE or UPDATE statement.
“GetConnection method”	Gets the connection object.
“GetParameterCount method”	Gets the number of input parameters for this statement.
“GetParameterType method”	Gets the storage/host variable type of a parameter.
“GetPlan method”	Gets a text-based description of the query execution plan.
“GetResultSetSchema method”	Gets the schema for the result set.
“GetRowsAffectedCount method”	Gets the number of rows affected by the last statement.
“HasResultSet method”	Determines if the SQL statement has a result set.
“SetParameterBinary method”	Sets a parameter to a ul_binary value.
“SetParameterDateTime method”	Sets a parameter to a DECL_DATETIME value.
“SetParameterDouble method”	Sets a parameter to a double value.
“SetParameterFloat method”	Sets a parameter to a float value.
“SetParameterGuid method”	Sets a parameter to a GUID value.
“SetParameterInt method”	Sets a parameter to an integer value.
“SetParameterIntWithType method”	Sets a parameter to an integer value of the specified integer type.



Name	Description
<a href="#">“SetParameterNull method”</a>	Sets a parameter to null.
<a href="#">“SetParameterString method”</a>	Sets a parameter to a string value.

## AppendParameterByteChunk method

Sets a large binary parameter broken down into several chunks.

### Syntax

```
public virtual bool AppendParameterByteChunk (
    ul_column_num pid,
    const ul_byte * value,
    size_t valueSize
)
```

### Parameters

- **pid** The 1-based ordinal of the parameter.
- **value** The byte chunk to append.
- **valueSize** The size of the buffer.

### Returns

True on success, otherwise false.

## AppendParameterStringChunk method

Sets a large string parameter broken down into several chunks.

### Overload list

Name	Description
<a href="#">“AppendParameterStringChunk(ul_column_num, const char *, size_t) method”</a>	Sets a large string parameter broken down into several chunks.
<a href="#">“AppendParameterStringChunk(ul_column_num, const ul_wchar *, size_t) method”</a>	Sets a large wide string parameter broken down into several chunks.

## AppendParameterStringChunk(*ul\_column\_num*, *const char \**, *size\_t*) method

Sets a large string parameter broken down into several chunks.

### Syntax

```
public virtual bool AppendParameterStringChunk(  
    ul_column_num pid,  
    const char * value,  
    size_t len  
)
```

### Parameters

- **pid** The 1-based ordinal of the parameter.
- **value** The string chunk to append.
- **len** The length of the string chunk in bytes or `UL_NULL_TERMINATED_STRING` if the string chunk is null-terminated.

### Returns

True on success, otherwise false.

## AppendParameterStringChunk(*ul\_column\_num*, *const ul\_wchar \**, *size\_t*) method

Sets a large wide string parameter broken down into several chunks.

### Syntax

```
public virtual bool AppendParameterStringChunk(  
    ul_column_num pid,  
    const ul_wchar * value,  
    size_t len  
)
```

### Parameters

- **pid** The 1-based ordinal of the parameter.
- **value** The string chunk to append.
- **len** The length of the string chunk in bytes or `UL_NULL_TERMINATED_STRING` if the string chunk is null-terminated.

### Returns

True on success, otherwise false.

## Close method

Destroys this object.

### Syntax

```
public virtual void Close()
```

## ExecuteQuery method

Executes a SQL SELECT statement as a query.

### Syntax

```
public virtual ULResultSet * ExecuteQuery()
```

### Returns

The ULResultSet of the query, as a set of rows.

## ExecuteStatement method

Executes a statement that does not return a result set, such as a SQL INSERT, DELETE or UPDATE statement.

### Syntax

```
public virtual bool ExecuteStatement()
```

### Returns

True on success, otherwise false.

## GetConnection method

Gets the connection object.

### Syntax

```
public virtual ULConnection * GetConnection()
```

### Returns

The ULConnection associated with this prepared statement.

## GetParameterCount method

Gets the number of input parameters for this statement.

**Syntax**

```
public virtual ul_u_short GetParameterCount()
```

**Returns**

The number of input parameters for this statement.

## GetParameterType method

Gets the storage/host variable type of a parameter.

**Syntax**

```
public virtual ul_column_storage_type GetParameterType(  
    ul_column_num pid  
)
```

**Parameters**

- **pid** The 1-based ordinal of the parameter.

**Returns**

The type of the specified parameter.

**Remarks**

For more information about supported storage types, see [“ul\\_column\\_storage\\_type enumeration” on page 72](#).

## GetPlan method

Gets a text-based description of the query execution plan.

**Overload list**

Name	Description
<a href="#">“GetPlan(char *, size_t) method”</a>	Gets a text-based description of the query execution plan.
<a href="#">“GetPlan(ul_wchar *, size_t) method”</a>	Gets a text-based description of the query execution plan.

## GetPlan(char \*, size\_t) method

Gets a text-based description of the query execution plan.

**Syntax**

```
public virtual size_t GetPlan(char * dst, size_t dstSize)
```

**Parameters**

- **dst** The destination buffer for the plan text. Pass NULL to determine the size of the buffer required to hold the plan.
- **dstSize** The size of the destination buffer.

**Returns**

The number of bytes copied to the buffer. The number of bytes required to store the plan, excluding the null-terminator, if *dst* is NULL.

**Remarks**

This function is intended primarily for use during development.

An empty string is returned if there is no plan. Plans exist when the prepared statement is a SQL query.

When the plan is obtained before the associated query has been executed, the plan shows the operations used to execute the query. The plan additionally shows the number of rows each operation produced when the plan is obtained after the query has been executed. This plan can be used to gain insight about the execution of the query.

## GetPlan(**ul\_wchar \***, **size\_t**) method

Gets a text-based description of the query execution plan.

**Syntax**

```
public virtual size_t GetPlan(ul_wchar * dst, size_t dstSize)
```

**Parameters**

- **dst** The destination buffer for the plan text. Pass NULL to determine the size of the buffer required to hold the plan.
- **dstSize** The size of the destination buffer.

**Returns**

The number of bytes copied to the buffer. The number of bytes required to store the plan, excluding the null-terminator, if *dst* is NULL.

**Remarks**

This function is intended primarily for use during development.

An empty string is returned if there is no plan. Plans exist when the prepared statement is a SQL query.

When the plan is obtained before the associated query has been executed, the plan shows the operations used to execute the query. The plan additionally shows the number of rows each operation produced when

the plan is obtained after the query has been executed. This plan can be used to gain insight about the execution of the query.

## GetResultSetSchema method

Gets the schema for the result set.

### Syntax

```
public virtual const ULResultSetSchema & GetResultSetSchema()
```

### Returns

A ULResultSetSchema object that can be used to get information about the schema of the result set.

## GetRowsAffectedCount method

Gets the number of rows affected by the last statement.

### Syntax

```
public virtual ul_s_long GetRowsAffectedCount()
```

### Returns

The number of rows affected by the last statement. If the number of rows is not available (for instance, the statement alters the schema rather than data) the return value is -1.

## HasResultSet method

Determines if the SQL statement has a result set.

### Syntax

```
public virtual bool HasResultSet()
```

### Returns

True if a result set is generated when this statement is executed. False if no result set is generated.

## SetParameterBinary method

Sets a parameter to a ul\_binary value.

### Syntax

```
public virtual bool SetParameterBinary(  
    ul_column_num pid,
```

```
        const p_ul_binary value  
    )
```

### Parameters

- **pid** The 1-based ordinal of the parameter.
- **value** The ul\_binary value.

### Returns

True on success, otherwise false.

## SetParameterDateTime method

Sets a parameter to a DECL\_DATETIME value.

### Syntax

```
public virtual bool SetParameterDateTime(  
    ul_column_num pid,  
    DECL_DATETIME * value  
)
```

### Parameters

- **pid** The 1-based ordinal of the parameter.
- **value** The DECL\_DATETIME value.

### Returns

True on success, otherwise false.

## SetParameterDouble method

Sets a parameter to a double value.

### Syntax

```
public virtual bool SetParameterDouble(  
    ul_column_num pid,  
    ul_double value  
)
```

### Parameters

- **pid** The 1-based ordinal of the parameter.
- **value** The double value.

### Returns

True on success, otherwise false.

## SetParameterFloat method

Sets a parameter to a float value.

### Syntax

```
public virtual bool SetParameterFloat(ul_column_num pid, ul_real value)
```

### Parameters

- **pid** The 1-based ordinal of the parameter.
- **value** The float value.

### Returns

True on success, otherwise false.

## SetParameterGuid method

Sets a parameter to a GUID value.

### Syntax

```
public virtual bool SetParameterGuid(ul_column_num pid, GUID * value)
```

### Parameters

- **pid** The 1-based ordinal of the parameter.
- **value** The GUID value.

### Returns

True on success, otherwise false.

## SetParameterInt method

Sets a parameter to an integer value.

### Syntax

```
public virtual bool SetParameterInt(ul_column_num pid, ul_s_long value)
```



**Parameters**

- **pid** The 1-based ordinal of the parameter.
- **value** The integer value.

**Returns**

True on success, otherwise false.

## SetParameterIntWithType method

Sets a parameter to an integer value of the specified integer type.

**Syntax**

```
public virtual bool SetParameterIntWithType(  
    ul_column_num pid,  
    ul_s_big value,  
    ul_column_storage_type type  
)
```

**Parameters**

- **pid** The 1-based ordinal of the parameter.
- **value** The integer value.
- **type** The integer type to treat the value as.

**Returns**

True on success, otherwise false.

**Remarks**

The following is a list of integer values that can be used for the value parameter:

- UL\_TYPE\_BIT
- UL\_TYPE\_TINY
- UL\_TYPE\_S\_SHORT
- UL\_TYPE\_U\_SHORT
- UL\_TYPE\_S\_LONG
- UL\_TYPE\_U\_LONG
- UL\_TYPE\_S\_BIG
- UL\_TYPE\_U\_BIG

## SetParameterNull method

Sets a parameter to null.

### Syntax

```
public virtual bool SetParameterNull(ul_column_num pid)
```

### Parameters

- **pid** The 1-based ordinal of the parameter.

### Returns

True on success, otherwise false.

## SetParameterString method

Sets a parameter to a string value.

### Overload list

Name	Description
<a href="#">“SetParameterString(ul_column_num, const char *, size_t) method”</a>	Sets a parameter to a string value.
<a href="#">“SetParameterString(ul_column_num, const ul_wchar *, size_t) method”</a>	Sets a parameter to a wide string value.

## SetParameterString(ul\_column\_num, const char \*, size\_t) method

Sets a parameter to a string value.

### Syntax

```
public virtual bool SetParameterString(  
    ul_column_num pid,  
    const char * value,  
    size_t len  
)
```

### Parameters

- **pid** The 1-based ordinal of the parameter.
- **value** The string value.

- **len** The length of the string in bytes or `UL_NULL_TERMINATED_STRING` if the string is null-terminated. `SQLE_INVALID_PARAMETER` is set if `len > 32K`. For large strings, call `AppendParameterStringChunk` instead.

### Returns

True on success, otherwise false.

## SetParameterString(*ul\_column\_num*, *const ul\_wchar \**, *size\_t*) method

Sets a parameter to a wide string value.

### Syntax

```
public virtual bool SetParameterString(  
    ul_column_num pid,  
    const ul_wchar * value,  
    size_t len  
)
```

### Parameters

- **pid** The 1-based ordinal of the parameter.
- **value** The string value.
- **len** The length of the string in bytes or `UL_NULL_TERMINATED_STRING` if the string is null-terminated. `SQLE_INVALID_PARAMETER` is set if `len > 32K`. For large strings, call `AppendParameterStringChunk` instead.

### Returns

True on success, otherwise false.

## ULResultSet class

Represents a result set in an UltraLite database.

### Syntax

```
public class ULResultSet
```

### Derived classes

- [“ULTable class” on page 235](#)

### Members

All members of `ULResultSet` class, including all inherited members.

Name	Description
“AfterLast method”	Moves the cursor after the last row.
“AppendByteChunk method”	Appends bytes to a column.
“AppendStringChunk method”	Appends a string chunk to a column.
“BeforeFirst method”	Moves the cursor before the first row.
“Close method”	Destroys this object.
“Delete method”	Deletes the current row and moves it to the next valid row.
“DeleteNamed method”	Deletes the current row and moves it to the next valid row.
“First method”	Moves the cursor to the first row.
“GetBinary method”	Fetches a value from a column as a ul_binary.
“GetBinaryLength method”	Gets the binary length of the value of a column.
“GetByteChunk method”	Gets a binary chunk from the column.
“GetConnection method”	Gets the connection object.
“GetDateTime method”	Fetches a value from a column as a DECL_DATETIME.
“GetDouble method”	Fetches a value from a column as a double.
“GetFloat method”	Fetches a value from a column as a float.
“GetGuid method”	Fetches a value from a column as a GUID.
“GetInt method”	Fetches a value from a column as an integer.
“GetIntWithType method”	Fetches a value from a column as the specified integer type.
“GetResultSetSchema method”	Returns an object that can be used to get information about the result set.
“GetRowCount method”	Gets the number of rows in the table.
“GetState method”	Gets the internal state of the cursor.
“GetString method”	Fetches a value from a column as a null-terminated string.
“GetStringChunk method”	Gets a string chunk from the column.
“GetStringLength method”	Gets the string length of the value of a column.

Name	Description
“IsNull method”	Checks if a column is NULL.
“Last method”	Moves the cursor to the last row.
“Next method”	Moves the cursor forward one row.
“Previous method”	Moves the cursor back one row.
“Relative method”	Moves the cursor by offset rows from the current cursor position.
“SetBinary method”	Sets a column to a ul_binary value.
“SetDateTime method”	Sets a column to a DECL_DATETIME value.
“SetDefault method”	Sets a column to its default value.
“SetDouble method”	Sets a column to a double value.
“SetFloat method”	Sets a column to a float value.
“SetGuid method”	Sets a column to a GUID value.
“SetInt method”	Sets a column to an integer value.
“SetIntWithType method”	Sets a column to an integer value of the specified integer type.
“SetNull method”	Sets a column to null.
“SetString method”	Sets a column to a string value.
“Update method”	Updates the current row.
“UpdateBegin method”	Selects update mode for setting columns.

## AfterLast method

Moves the cursor after the last row.

### Syntax

```
public virtual bool AfterLast()
```

### Returns

True on success, otherwise false.

## AppendByteChunk method

Appends bytes to a column.

### Overload list

Name	Description
<a href="#">“AppendByteChunk(const char *, const ul_byte *, size_t) method”</a>	Appends bytes to a column.
<a href="#">“AppendByteChunk(const ul_wchar *, const ul_byte *, size_t) method”</a>	Appends bytes to a column.
<a href="#">“AppendByteChunk(ul_column_num, const ul_byte *, size_t) method”</a>	Appends bytes to a column.

## AppendByteChunk(const char \*, const ul\_byte \*, size\_t) method

Appends bytes to a column.

### Syntax

```
public virtual bool AppendByteChunk(  
    const char * cname,  
    const ul_byte * value,  
    size_t valueSize  
)
```

### Parameters

- **cname** The name of the column.
- **value** The byte chunk to append.
- **valueSize** The size of the byte chunk in bytes.

### Returns

True on success, otherwise false.

### Remarks

The given bytes are appended to the end of the column written so far by by AppendBinaryChunk calls.

## AppendByteChunk(const ul\_wchar \*, const ul\_byte \*, size\_t) method

Appends bytes to a column.

### Syntax

```
public virtual bool AppendByteChunk(  
    const ul_wchar * cname,  
    const ul_byte * value,
```

```
        size_t valueSize  
    )
```

### Parameters

- **cname** The name of the column.
- **value** The byte chunk to append.
- **valueSize** The size of the byte chunk in bytes.

### Returns

True on success, otherwise false.

### Remarks

The given bytes are appended to the end of the column written so far by AppendBinaryChunk calls.

## AppendByteChunk(ul\_column\_num, const ul\_byte \*, size\_t) method

Appends bytes to a column.

### Syntax

```
public virtual bool AppendByteChunk(  
    ul_column_num cid,  
    const ul_byte * value,  
    size_t valueSize  
)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **value** The byte chunk to append.
- **valueSize** The size of the byte chunk in bytes.

### Returns

True on success, otherwise false.

### Remarks

The given bytes are appended to the end of the column written so far by AppendBinaryChunk calls.

## AppendStringChunk method

Appends a string chunk to a column.

**Overload list**

Name	Description
“AppendStringChunk(const char *, const char *, size_t) method”	Appends a string chunk to a column.
“AppendStringChunk(const ul_wchar *, const ul_wchar *, size_t) method”	Appends a wide string chunk to a column.
“AppendStringChunk(ul_column_num, const char *, size_t) method”	Appends a string chunk to a column.
“AppendStringChunk(ul_column_num, const ul_wchar *, size_t) method”	Appends a wide string chunk to a column.

## AppendStringChunk(const char \*, const char \*, size\_t) method

Appends a string chunk to a column.

**Syntax**

```
public virtual bool AppendStringChunk(  
    const char * cname,  
    const char * value,  
    size_t len  
)
```

**Parameters**

- **cname** The name of the column.
- **value** The string chunk to append.
- **len** The length of the string chunk in bytes or UL\_NULL\_TERMINATED\_STRING if the string is null-terminated.

**Returns**

True on success, otherwise false.

**Remarks**

Appends the given string to the end of the string written so far by AppendStringChunk calls.

## AppendStringChunk(const ul\_wchar \*, const ul\_wchar \*, size\_t) method

Appends a wide string chunk to a column.



**Syntax**

```
public virtual bool AppendStringChunk(  
    const ul_wchar * cname,  
    const ul_wchar * value,  
    size_t len  
)
```

**Parameters**

- **cname** The name of the column.
- **value** The wide string chunk to append.
- **len** The length of the string chunk in ul\_wchars or UL\_NULL\_TERMINATED\_STRING if the string is null-terminated.

**Returns**

True on success, otherwise false.

**Remarks**

The given string are appended to the end of the string written so far by AppendStringChunk calls.

**AppendStringChunk(ul\_column\_num, const char \*, size\_t) method**

Appends a string chunk to a column.

**Syntax**

```
public virtual bool AppendStringChunk(  
    ul_column_num cid,  
    const char * value,  
    size_t len  
)
```

**Parameters**

- **cid** The 1-based ordinal column number.
- **value** The string chunk to append.
- **len** The length of the string chunk in bytes or UL\_NULL\_TERMINATED\_STRING if the string is null-terminated.

**Returns**

True on success, otherwise false.

**Remarks**

Appends the given string to the end of the string written so far by AppendStringChunk calls.

## AppendStringChunk(*ul\_column\_num*, *const ul\_wchar \**, *size\_t*) method

Appends a wide string chunk to a column.

### Syntax

```
public virtual bool AppendStringChunk(  
    ul_column_num cid,  
    const ul_wchar * value,  
    size_t len  
)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **value** The wide string chunk to append.
- **len** The length of the string chunk in *ul\_wchars* or `UL_NULL_TERMINATED_STRING` if the string is null-terminated.

### Returns

True on success, otherwise false.

### Remarks

Appends the given string to the end of the string written so far by `AppendStringChunk` calls.

## BeforeFirst method

Moves the cursor before the first row.

### Syntax

```
public virtual bool BeforeFirst()
```

### Returns

True on success, otherwise false.

## Close method

Destroys this object.

### Syntax

```
public virtual void Close()
```

## Delete method

Deletes the current row and moves it to the next valid row.

### Syntax

```
public virtual bool Delete()
```

### Returns

True on success, otherwise false.

### Remarks

If this cursor is a table opened without an index, then the data is considered read only and rows cannot be deleted.

## DeleteNamed method

Deletes the current row and moves it to the next valid row.

### Overload list

Name	Description
<a href="#">“DeleteNamed(const char *) method”</a>	Deletes the current row and moves it to the next valid row.
<a href="#">“DeleteNamed(const ul_wchar *) method”</a>	Deletes the current row and moves it to the next valid row.

## DeleteNamed(const char \*) method

Deletes the current row and moves it to the next valid row.

### Syntax

```
public virtual bool DeleteNamed(const char * tableName)
```

### Parameters

- **tableName** A table name or its correlation (required when the database has multiple columns that share the same table name).

### Returns

True on success, otherwise false.

## DeleteNamed(const ul\_wchar \*) method

Deletes the current row and moves it to the next valid row.

**Syntax**

```
public virtual bool DeleteNamed(const ul_wchar * tableName)
```

**Parameters**

- **tableName** A table name or its correlation (required when the database has multiple columns that share the same table name).

**Returns**

True on success, otherwise false.

## First method

Moves the cursor to the first row.

**Syntax**

```
public virtual bool First()
```

**Returns**

True on success, otherwise false.

## GetBinary method

Fetches a value from a column as a `ul_binary`.

**Overload list**

Name	Description
<a href="#">“GetBinary(const char *, p_ul_binary, size_t) method”</a>	Fetches a value from a column as a <code>ul_binary</code> .
<a href="#">“GetBinary(const ul_wchar *, p_ul_binary, size_t) method”</a>	Fetches a value from a column as a <code>ul_binary</code> .
<a href="#">“GetBinary(ul_column_num, p_ul_binary, size_t) method”</a>	Fetches a value from a column as a <code>ul_binary</code> .

## GetBinary(const char \*, p\_ul\_binary, size\_t) method

Fetches a value from a column as a `ul_binary`.

**Syntax**

```
public virtual bool GetBinary(  
    const char * cname,
```

```
    p_ul_binary dst,  
    size_t len  
)
```

### Parameters

- **cname** The name of the column.
- **dst** The ul\_binary result.
- **len** The size of the ul\_binary object.

### Returns

True if the value was successfully fetched.

## GetBinary(const ul\_wchar \*, p\_ul\_binary, size\_t) method

Fetches a value from a column as a ul\_binary.

### Syntax

```
public virtual bool GetBinary(  
    const ul_wchar * cname,  
    p_ul_binary dst,  
    size_t len  
)
```

### Parameters

- **cname** The name of the column.
- **dst** The ul\_binary result.
- **len** The size of the ul\_binary object.

### Returns

True if the value was successfully fetched.

## GetBinary(ul\_column\_num, p\_ul\_binary, size\_t) method

Fetches a value from a column as a ul\_binary.

### Syntax

```
public virtual bool GetBinary(  
    ul_column_num cid,  
    p_ul_binary dst,  
    size_t len  
)
```

**Parameters**

- **cid** The 1-based ordinal column number.
- **dst** The `ul_binary` result.
- **len** The size of the `ul_binary` object.

**Returns**

True if the value was successfully fetched.

## GetBinaryLength method

Gets the binary length of the value of a column.

**Overload list**

Name	Description
<a href="#">“GetBinaryLength(const char *) method”</a>	Gets the binary length of the value of a column.
<a href="#">“GetBinaryLength(const ul_wchar *) method”</a>	Gets the binary length of the value of a column.
<a href="#">“GetBinaryLength(ul_column_num) method”</a>	Gets the binary length of the value of a column.

## GetBinaryLength(const char \*) method

Gets the binary length of the value of a column.

**Syntax**

```
public virtual size_t GetBinaryLength(const char * cname)
```

**Parameters**

- **cname** The name of the column.

**Returns**

The size of the column value as a binary

## GetBinaryLength(const ul\_wchar \*) method

Gets the binary length of the value of a column.

**Syntax**

```
public virtual size_t GetBinaryLength(const ul_wchar * cname)
```

**Parameters**

- **cname** The name of the column.

**Returns**

The size of the column value as a binary

**GetBinaryLength(ul\_column\_num) method**

Gets the binary length of the value of a column.

**Syntax**

```
public virtual size_t GetBinaryLength(ul_column_num cid)
```

**Parameters**

- **cid** The 1-based ordinal column number.

**Returns**

The size of the column value as a binary

**GetByteChunk method**

Gets a binary chunk from the column.

**Overload list**

Name	Description
<a href="#">“GetByteChunk(const char *, ul_byte *, size_t, size_t) method”</a>	Gets a binary chunk from the column.
<a href="#">“GetByteChunk(const ul_wchar *, ul_byte *, size_t, size_t) method”</a>	Gets a binary chunk from the column.
<a href="#">“GetByteChunk(ul_column_num, ul_byte *, size_t, size_t) method”</a>	Gets a binary chunk from the column.

**GetByteChunk(const char \*, ul\_byte \*, size\_t, size\_t) method**

Gets a binary chunk from the column.

**Syntax**

```
public virtual size_t GetByteChunk(
    const char * cname,
    ul_byte * dst,
    size_t len,
```

```
        size_t offset
    )
```

### Parameters

- **cname** The name of the column.
- **dst** The buffer to hold the bytes.
- **len** The size of the buffer in bytes.
- **offset** The offset into the value at which to start reading or `UL_BLOB_CONTINUE` to continue from where the last read ended.

### Returns

The number of bytes copied to the destination buffer. If *dst* is `NULL`, the number of bytes left is returned.

### Remarks

The end of the value has been reached if 0 is returned.

## GetByteChunk(const ul\_wchar \*, ul\_byte \*, size\_t, size\_t) method

Gets a binary chunk from the column.

### Syntax

```
public virtual size_t GetByteChunk(
    const ul_wchar * cname,
    ul_byte * dst,
    size_t len,
    size_t offset
)
```

### Parameters

- **cname** The name of the column.
- **dst** The buffer to hold the bytes.
- **len** The size of the buffer in bytes.
- **offset** The offset into the value at which to start reading or `UL_BLOB_CONTINUE` to continue from where the last read ended.

### Returns

The number of bytes copied to the destination buffer. If *dst* is `NULL`, the number of bytes left is returned.

### Remarks

The end of the value has been reached if 0 is returned.



## GetByteChunk(*ul\_column\_num*, *ul\_byte \**, *size\_t*, *size\_t*) method

Gets a binary chunk from the column.

### Syntax

```
public virtual size_t GetByteChunk(  
    ul_column_num cid,  
    ul_byte * dst,  
    size_t len,  
    size_t offset  
)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **dst** The buffer to hold the bytes.
- **len** The size of the buffer in bytes.
- **offset** The offset into the value at which to start reading or UL\_BLOB\_CONTINUE to continue from where the last read ended.

### Returns

The number of bytes copied to the destination buffer. If *dst* is NULL, the number of bytes left is returned.

### Remarks

The end of the value has been reached if 0 is returned.

## GetConnection method

Gets the connection object.

### Syntax

```
public virtual ULConnection * GetConnection()
```

### Returns

The ULConnection associated with this result set.

## GetDateTime method

Fetches a value from a column as a DECL\_DATETIME.

**Overload list**

Name	Description
“ <a href="#">GetDateTime(const char *, DECL_DATETIME *)</a> method”	Fetches a value from a column as a DECL_DATETIME.
“ <a href="#">GetDateTime(const ul_wchar *, DECL_DATETIME *)</a> method”	Fetches a value from a column as a DECL_DATETIME.
“ <a href="#">GetDateTime(ul_column_num, DECL_DATETIME *)</a> method”	Fetches a value from a column as a DECL_DATETIME.

## [GetDateTime\(const char \\*, DECL\\_DATETIME \\*\)](#) method

Fetches a value from a column as a DECL\_DATETIME.

**Syntax**

```
public virtual bool GetDateTime(const char * cname, DECL_DATETIME * dst)
```

**Parameters**

- **cname** The name of the column.
- **dst** The DECL\_DATETIME value.

**Returns**

True if the value was successfully fetched.

## [GetDateTime\(const ul\\_wchar \\*, DECL\\_DATETIME \\*\)](#) method

Fetches a value from a column as a DECL\_DATETIME.

**Syntax**

```
public virtual bool GetDateTime(  
    const ul_wchar * cname,  
    DECL_DATETIME * dst  
)
```

**Parameters**

- **cname** The name of the column.
- **dst** The DECL\_DATETIME value.

**Returns**

True if the value was successfully fetched.

## GetDateTime(*ul\_column\_num*, DECL\_DATETIME \*) method

Fetches a value from a column as a DECL\_DATETIME.

### Syntax

```
public virtual bool GetDateTime(ul_column_num cid, DECL_DATETIME * dst)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **dst** The DECL\_DATETIME value.

### Returns

True if the value was successfully fetched.

## GetDouble method

Fetches a value from a column as a double.

### Overload list

Name	Description
<a href="#">“GetDouble(const char *) method”</a>	Fetches a value from a column as a double.
<a href="#">“GetDouble(const ul_wchar *) method”</a>	Fetches a value from a column as a double.
<a href="#">“GetDouble(<i>ul_column_num</i>) method”</a>	Fetches a value from a column as a double.

## GetDouble(const char \*) method

Fetches a value from a column as a double.

### Syntax

```
public virtual ul_double GetDouble(const char * cname)
```

### Parameters

- **cname** The name of the column.

### Returns

The column value as a double.

## GetDouble(const ul\_wchar \*) method

Fetches a value from a column as a double.

**Syntax**

```
public virtual ul_double GetDouble(const ul_wchar * cname)
```

**Parameters**

- **cname** The name of the column.

**Returns**

The column value as a double.

## GetDouble(*ul\_column\_num*) method

Fetches a value from a column as a double.

**Syntax**

```
public virtual ul_double GetDouble(ul_column_num cid)
```

**Parameters**

- **cid** The 1-based ordinal column number.

**Returns**

The column value as a double.

## GetFloat method

Fetches a value from a column as a float.

**Overload list**

Name	Description
<a href="#">“GetFloat(const char *) method”</a>	Fetches a value from a column as a float.
<a href="#">“GetFloat(const ul_wchar *) method”</a>	Fetches a value from a column as a float.
<a href="#">“GetFloat(<i>ul_column_num</i>) method”</a>	Fetches a value from a column as a float.

## GetFloat(const char \*) method

Fetches a value from a column as a float.

**Syntax**

```
public virtual ul_real GetFloat(const char * cname)
```

**Parameters**

- **cname** The name of the column.

**Returns**

The column value as a float.

**GetFloat(const ul\_wchar \*) method**

Fetches a value from a column as a float.

**Syntax**

```
public virtual ul_real GetFloat(const ul_wchar * cname)
```

**Parameters**

- **cname** The name of the column.

**Returns**

The column value as a float.

**GetFloat(ul\_column\_num) method**

Fetches a value from a column as a float.

**Syntax**

```
public virtual ul_real GetFloat(ul_column_num cid)
```

**Parameters**

- **cid** The 1-based ordinal column number.

**Returns**

The column value as a float.

**GetGuid method**

Fetches a value from a column as a GUID.

**Overload list**

Name	Description
<a href="#">“GetGuid(const char *, GUID *) method”</a>	Fetches a value from a column as a GUID.
<a href="#">“GetGuid(const ul_wchar *, GUID *) method”</a>	Fetches a value from a column as a GUID.

Name	Description
<a href="#">“GetGuid(ul_column_num, GUID *) method”</a>	Fetches a value from a column as a GUID.

## GetGuid(const char \*, GUID \*) method

Fetches a value from a column as a GUID.

### Syntax

```
public virtual bool GetGuid(const char * cname, GUID * dst)
```

### Parameters

- **cname** The name of the column.
- **dst** The GUID value.

### Returns

True if the value was successfully fetched.

## GetGuid(const ul\_wchar \*, GUID \*) method

Fetches a value from a column as a GUID.

### Syntax

```
public virtual bool GetGuid(const ul_wchar * cname, GUID * dst)
```

### Parameters

- **cname** The name of the column.
- **dst** The GUID value.

### Returns

True if the value was successfully fetched.

## GetGuid(ul\_column\_num, GUID \*) method

Fetches a value from a column as a GUID.

### Syntax

```
public virtual bool GetGuid(ul_column_num cid, GUID * dst)
```

### Parameters

- **cid** The 1-based ordinal column number.

- **dst** The GUID value.

### Returns

True if the value was successfully fetched.

## GetInt method

Fetches a value from a column as an integer.

### Overload list

Name	Description
<a href="#">“GetInt(const char *) method”</a>	Fetches a value from a column as an integer.
<a href="#">“GetInt(const ul_wchar *) method”</a>	Fetches a value from a column as an integer.
<a href="#">“GetInt(ul_column_num) method”</a>	Fetches a value from a column as an integer.

### GetInt(const char \*) method

Fetches a value from a column as an integer.

#### Syntax

```
public virtual ul_s_long GetInt(const char * cname)
```

#### Parameters

- **cname** The name of the column.

#### Returns

The column value as an integer.

### GetInt(const ul\_wchar \*) method

Fetches a value from a column as an integer.

#### Syntax

```
public virtual ul_s_long GetInt(const ul_wchar * cname)
```

#### Parameters

- **cname** The name of the column.

#### Returns

The column value as an integer.

## GetInt(ul\_column\_num) method

Fetches a value from a column as an integer.

### Syntax

```
public virtual ul_s_long GetInt(ul_column_num cid)
```

### Parameters

- **cid** The 1-based ordinal column number.

### Returns

The column value as an integer.

## GetIntWithType method

Fetches a value from a column as the specified integer type.

### Overload list

Name	Description
“ <a href="#">GetIntWithType(const char *, ul_column_storage_type) method</a> ”	Fetches a value from a column as the specified integer type.
“ <a href="#">GetIntWithType(const ul_wchar *, ul_column_storage_type) method</a> ”	Fetches a value from a column as the specified integer type.
“ <a href="#">GetIntWithType(ul_column_num, ul_column_storage_type) method</a> ”	Fetches a value from a column as the specified integer type.

## GetIntWithType(const char \*, ul\_column\_storage\_type) method

Fetches a value from a column as the specified integer type.

### Syntax

```
public virtual ul_s_big GetIntWithType(  
    const char * cname,  
    ul_column_storage_type type  
)
```

### Parameters

- **cname** The name of the column.
- **type** The integer type to fetch as.



**Returns**

The column value as an integer.

**Remarks**

The following is a list of integer values that can be used for the type parameter:

- UL\_TYPE\_BIT
- UL\_TYPE\_TINY
- UL\_TYPE\_S\_SHORT
- UL\_TYPE\_U\_SHORT
- UL\_TYPE\_S\_LONG
- UL\_TYPE\_U\_LONG
- UL\_TYPE\_S\_BIG
- UL\_TYPE\_U\_BIG

**GetIntWithType(const ul\_wchar \*, ul\_column\_storage\_type) method**

Fetches a value from a column as the specified integer type.

**Syntax**

```
public virtual ul_s_big GetIntWithType(  
    const ul_wchar * cname,  
    ul_column_storage_type type  
)
```

**Parameters**

- **cname** The name of the column.
- **type** The integer type to fetch as.

**Returns**

The column value as an integer.

**Remarks**

The following is a list of integer values that can be used for the type parameter:

- UL\_TYPE\_BIT
- UL\_TYPE\_TINY
- UL\_TYPE\_S\_SHORT

- `UL_TYPE_U_SHORT`
- `UL_TYPE_S_LONG`
- `UL_TYPE_U_LONG`
- `UL_TYPE_S_BIG`
- `UL_TYPE_U_BIG`

## **GetIntWithType(*ul\_column\_num*, *ul\_column\_storage\_type*) method**

Fetches a value from a column as the specified integer type.

### **Syntax**

```
public virtual ul_s_big GetIntWithType(  
    ul_column_num cid,  
    ul_column_storage_type type  
)
```

### **Parameters**

- **cid** The 1-based ordinal column number.
- **type** The integer type to fetch as.

### **Returns**

The column value as an integer.

### **Remarks**

The following is a list of integer values that can be used for the type parameter:

- `UL_TYPE_BIT`
- `UL_TYPE_TINY`
- `UL_TYPE_S_SHORT`
- `UL_TYPE_U_SHORT`
- `UL_TYPE_S_LONG`
- `UL_TYPE_U_LONG`
- `UL_TYPE_S_BIG`
- `UL_TYPE_U_BIG`

---

## GetResultSetSchema method

Returns an object that can be used to get information about the result set.

### Syntax

```
public virtual const ULResultSetSchema & GetResultSetSchema()
```

### Returns

A ULResultSetSchema object that can be used to get information about the result set.

## GetRowCount method

Gets the number of rows in the table.

### Syntax

```
public virtual ul_u_long GetRowCount(ul_u_long threshold)
```

### Parameters

- **threshold** The limit on the number of rows to count. Zero indicates no limit.

### Returns

The number of rows in the table.

### Remarks

This method is equivalent to executing "SELECT COUNT(\*) FROM table".

## GetState method

Gets the internal state of the cursor.

### Syntax

```
public virtual UL_RS_STATE GetState()
```

### Returns

The state of the cursor.

### Remarks

For more information, see [“UL\\_RS\\_STATE enumeration” on page 70](#).

## GetString method

Fetches a value from a column as a null-terminated string.

#### Overload list

Name	Description
<a href="#">“GetString(const char *, char *, size_t) method”</a>	Fetches a value from a column as a null-terminated string.
<a href="#">“GetString(const ul_wchar *, ul_wchar *, size_t) method”</a>	Fetches a value from a column as a null-terminated wide string.
<a href="#">“GetString(ul_column_num, char *, size_t) method”</a>	Fetches a value from a column as a null-terminated string.
<a href="#">“GetString(ul_column_num, ul_wchar *, size_t) method”</a>	Fetches a value from a column as a null-terminated wide string.

## GetString(const char \*, char \*, size\_t) method

Fetches a value from a column as a null-terminated string.

#### Syntax

```
public virtual bool GetString(  
    const char * cname,  
    char * dst,  
    size_t len  
)
```

#### Parameters

- **cname** The name of the column.
- **dst** The buffer to hold the string value. The string is null-terminated even if truncated.
- **len** The size of the buffer in bytes.

#### Returns

True if the value was successfully fetched.

#### Remarks

The string is truncated in the buffer when it isn't large enough to hold the entire value.

## GetString(const ul\_wchar \*, ul\_wchar \*, size\_t) method

Fetches a value from a column as a null-terminated wide string.

**Syntax**

```
public virtual bool GetString(
    const ul_wchar * cname,
    ul_wchar * dst,
    size_t len
)
```

**Parameters**

- **cname** The name of the column.
- **dst** The buffer to hold the wide string value. The string is null-terminated even if truncated.
- **len** The size of the buffer in ul\_wchars.

**Returns**

True if the value was successfully fetched.

**Remarks**

The string is truncated in the buffer when it isn't large enough to hold the entire value.

## GetString(**ul\_column\_num**, **char \***, **size\_t**) method

Fetches a value from a column as a null-terminated string.

**Syntax**

```
public virtual bool GetString(ul_column_num cid, char * dst, size_t len)
```

**Parameters**

- **cid** The 1-based ordinal column number.
- **dst** The buffer to hold the string value. The string is null-terminated even if truncated.
- **len** The size of the buffer in bytes.

**Returns**

True if the value was successfully fetched.

**Remarks**

The string is truncated in the buffer when it isn't large enough to hold the entire value.

## GetString(**ul\_column\_num**, **ul\_wchar \***, **size\_t**) method

Fetches a value from a column as a null-terminated wide string.

**Syntax**

```
public virtual bool GetString(
    ul_column_num cid,
```

```
    ul_wchar * dst,  
    size_t len  
)
```

**Parameters**

- **cid** The 1-based ordinal column number.
- **dst** The buffer to hold the wide string value. The string is null-terminated even if truncated.
- **len** The size of the buffer in `ul_wchars`.

**Returns**

True if the value was successfully fetched.

**Remarks**

The string is truncated in the buffer when it isn't large enough to hold the entire value.

## GetStringChunk method

Gets a string chunk from the column.

**Overload list**

Name	Description
“GetStringChunk(const char *, char *, size_t, size_t) method”	Gets a string chunk from the column.
“GetStringChunk(const ul_wchar *, ul_wchar *, size_t, size_t) method”	Gets a wide string chunk from the column.
“GetStringChunk(ul_column_num, char *, size_t, size_t) method”	Gets a string chunk from the column.
“GetStringChunk(ul_column_num, ul_wchar *, size_t, size_t) method”	Gets a wide string chunk from the column.

## GetStringChunk(const char \*, char \*, size\_t, size\_t) method

Gets a string chunk from the column.

**Syntax**

```
public virtual size_t GetStringChunk(  
    const char * cname,  
    char * dst,  
    size_t len,
```

```

        size_t offset
    )

```

### Parameters

- **cname** The name of the column.
- **dst** The buffer to hold the string chunk. The string is null-terminated even if truncated.
- **len** The size of the buffer in bytes.
- **offset** The offset into the value at which to start reading or `UL_BLOB_CONTINUE` to continue from where the last read ended.

### Returns

The number of bytes copied to the destination buffer excluding the null-terminator. If `dst` is `NULL`, the number of bytes left in the string is returned.

### Remarks

The end of the value has been reached if 0 is returned.

## GetStringChunk(const ul\_wchar \*, ul\_wchar \*, size\_t, size\_t) method

Gets a wide string chunk from the column.

### Syntax

```

public virtual size_t GetStringChunk(
    const ul_wchar * cname,
    ul_wchar * dst,
    size_t len,
    size_t offset
)

```

### Parameters

- **cname** The name of the column.
- **dst** The buffer to hold the string chunk. The string is null-terminated even if truncated.
- **len** The size of the buffer in `ul_wchars`.
- **offset** The offset into the value at which to start reading or `UL_BLOB_CONTINUE` to continue from where the last read ended.

### Returns

The number of `ul_wchars` copied to the destination buffer excluding the null-terminator. If `dst` is `NULL`, the number of `ul_wchars` left in the string is returned.

### Remarks

The end of the value has been reached if 0 is returned.

## GetStringChunk(*ul\_column\_num*, *char \**, *size\_t*, *size\_t*) method

Gets a string chunk from the column.

### Syntax

```
public virtual size_t GetStringChunk(  
    ul_column_num cid,  
    char * dst,  
    size_t len,  
    size_t offset  
)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **dst** The buffer to hold the string chunk. The string is null-terminated even if truncated.
- **len** The size of the buffer in bytes.
- **offset** The offset into the value at which to start reading or `UL_BLOB_CONTINUE` to continue from where the last read ended.

### Returns

The number of bytes copied to the destination buffer excluding the null-terminator. If *dst* is `NULL`, the number of bytes left in the string is returned

### Remarks

The end of the value has been reached if 0 is returned.

## GetStringChunk(*ul\_column\_num*, *ul\_wchar \**, *size\_t*, *size\_t*) method

Gets a wide string chunk from the column.

### Syntax

```
public virtual size_t GetStringChunk(  
    ul_column_num cid,  
    ul_wchar * dst,  
    size_t len,  
    size_t offset  
)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **dst** The buffer to hold the string chunk. The string is null-terminated even if truncated.
- **len** The size of the buffer in `ul_wchars`.



- **offset** The offset into the value at which to start reading or UL\_BLOB\_CONTINUE to continue from where the last read ended.

### Returns

The number of ul\_wchars copied to the destination buffer excluding the null-terminator. If dst is NULL, the number of ul\_wchars left in the string is returned.

### Remarks

The end of the value has been reached if 0 is returned.

## GetStringLength method

Gets the string length of the value of a column.

### Overload list

Name	Description
<a href="#">“GetStringLength(const char *) method”</a>	Gets the string length of the value of a column.
<a href="#">“GetStringLength(const ul_wchar *) method”</a>	Gets the string length of the value of a column.
<a href="#">“GetStringLength(ul_column_num) method”</a>	Gets the string length of the value of a column.

## GetStringLength(const char \*) method

Gets the string length of the value of a column.

### Syntax

```
public virtual size_t GetStringLength(const char * cname)
```

### Parameters

- **cname** The name of the column.

### Returns

The number of bytes or characters required to hold the string returned by one of the GetString() methods, not including the null-terminator.

### Remarks

The following example demonstrates how to get the string length of a column:

```
len = result_set->GetStringLength( cid );
dst = new char[ len + 1 ];
result_set->GetString( cid, dst, len + 1 );
```

For wide characters, the usage is as follows:

```
len = result_set->GetStringLength( cid );
dst = new ul_wchar[ len + 1 ];
result_set->GetString( cid, dst, len + 1 );
```

## GetStringLength(const ul\_wchar \*) method

Gets the string length of the value of a column.

### Syntax

```
public virtual size_t GetStringLength(const ul_wchar * cname)
```

### Parameters

- **cname** The name of the column.

### Returns

The number of bytes or characters required to hold the string returned by one of the GetString() methods, not including the null-terminator.

### Remarks

The following example demonstrates how to get the string length of a column:

```
len = result_set->GetStringLength( cid );
dst = new char[ len + 1 ];
result_set->GetString( cid, dst, len + 1 );
```

For wide characters, the usage is as follows:

```
len = result_set->GetStringLength( cid );
dst = new ul_wchar[ len + 1 ];
result_set->GetString( cid, dst, len + 1 );
```

## GetStringLength(ul\_column\_num) method

Gets the string length of the value of a column.

### Syntax

```
public virtual size_t GetStringLength(ul_column_num cid)
```

### Parameters

- **cid** The 1-based ordinal column number.

### Returns

The number of bytes or characters required to hold the string returned by one of the GetString() methods, not including the null-terminator.

### Remarks

The following example demonstrates how to get the string length of a column:

```
len = result_set->GetStringLength( cid );
dst = new char[ len + 1 ];
result_set->GetString( cid, dst, len + 1 );
```

For wide characters, the usage is as follows:

```
len = result_set->GetStringLength( cid );
dst = new ul_wchar[ len + 1 ];
result_set->GetString( cid, dst, len + 1 );
```

## IsNull method

Checks if a column is NULL.

### Overload list

Name	Description
<a href="#">“IsNull(const char *) method”</a>	Checks if a column is NULL.
<a href="#">“IsNull(const ul_wchar *) method”</a>	Checks if a column is NULL.
<a href="#">“IsNull(ul_column_num) method”</a>	Checks if a column is NULL.

## IsNull(const char \*) method

Checks if a column is NULL.

### Syntax

```
public virtual bool IsNull(const char * cname)
```

### Parameters

- **cname** The name of the column.

### Returns

True if the value for the column is NULL.

## IsNull(const ul\_wchar \*) method

Checks if a column is NULL.

### Syntax

```
public virtual bool IsNull(const ul_wchar * cname)
```

### Parameters

- **cname** The name of the column.

### Returns

True if the value for the column is NULL.

## IsNull(*ul\_column\_num*) method

Checks if a column is NULL.

### Syntax

```
public virtual bool IsNull(ul_column_num cid)
```

### Parameters

- **cid** The 1-based ordinal column number.

### Returns

True if the value for the column is NULL.

## Last method

Moves the cursor to the last row.

### Syntax

```
public virtual bool Last()
```

### Returns

True on success, otherwise false.

## Next method

Moves the cursor forward one row.

### Syntax

```
public virtual bool Next()
```

### Returns

True, if the cursor successfully moves forward. Despite returning true, an error may be signaled even when the cursor moves successfully to the next row. For example, there could be conversion errors while evaluating the SELECT expressions. In this case, errors are also returned when retrieving the column values. False is returned if it fails to move forward. For example, there may not be a next row. In this case, the resulting cursor position is AfterLast().

## Previous method

Moves the cursor back one row.

### Syntax

```
public virtual bool Previous()
```

### Returns

True, if the cursor successfully moves back one row. False, if it fails to move backward. The resulting cursor position is BeforeFirst().

## Relative method

Moves the cursor by offset rows from the current cursor position.

### Syntax

```
public virtual bool Relative(ul_fetch_offset offset)
```

### Parameters

- **offset** The number of rows to move.

### Returns

True on success, otherwise false.

## SetBinary method

Sets a column to a ul\_binary value.

### Overload list

Name	Description
<a href="#">“SetBinary(const char *, p_ul_binary) method”</a>	Sets a column to a ul_binary value.
<a href="#">“SetBinary(const ul_wchar *, p_ul_binary) method”</a>	Sets a column to a ul_binary value.
<a href="#">“SetBinary(ul_column_num, p_ul_binary) method”</a>	Sets a column to a ul_binary value.

## SetBinary(const char \*, p\_ul\_binary) method

Sets a column to a ul\_binary value.

### Syntax

```
public virtual bool SetBinary(const char * cname, p_ul_binary value)
```

### Parameters

- **cname** The name of the column.
- **value** The ul\_binary value. Passing NULL is equivalent to calling SetNull.

### Returns

True on success, otherwise false.

## SetBinary(const ul\_wchar \*, p\_ul\_binary) method

Sets a column to a ul\_binary value.

### Syntax

```
public virtual bool SetBinary(const ul_wchar * cname, p_ul_binary value)
```

### Parameters

- **cname** The name of the column.
- **value** The ul\_binary value. Passing NULL is equivalent to calling SetNull.

### Returns

True on success, otherwise false.

## SetBinary(ul\_column\_num, p\_ul\_binary) method

Sets a column to a ul\_binary value.

### Syntax

```
public virtual bool SetBinary(ul_column_num cid, p_ul_binary value)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **value** The ul\_binary value. Passing NULL is equivalent to calling SetNull.

### Returns

True on success, otherwise false.

## SetDateTime method

Sets a column to a DECL\_DATETIME value.

**Overload list**

Name	Description
“SetDateTime(const char *, DECL_DATETIME *) method”	Sets a column to a DECL_DATETIME value.
“SetDateTime(const ul_wchar *, DECL_DATETIME *) method”	Sets a column to a DECL_DATETIME value.
“SetDateTime(ul_column_num, DECL_DATETIME *) method”	Sets a column to a DECL_DATETIME value.

**SetDateTime(const char \*, DECL\_DATETIME \*) method**

Sets a column to a DECL\_DATETIME value.

**Syntax**

```
public virtual bool SetDateTime(
    const char * cname,
    DECL_DATETIME * value
)
```

**Parameters**

- **cname** The name of the column.
- **value** The DECL\_DATETIME value. Passing NULL is equivalent to calling SetNull.

**Returns**

True on success, otherwise false.

**SetDateTime(const ul\_wchar \*, DECL\_DATETIME \*) method**

Sets a column to a DECL\_DATETIME value.

**Syntax**

```
public virtual bool SetDateTime(
    const ul_wchar * cname,
    DECL_DATETIME * value
)
```

**Parameters**

- **cname** The name of the column.
- **value** The DECL\_DATETIME value. Passing NULL is equivalent to calling SetNull.

**Returns**

True on success, otherwise false.

## SetDateTime(*ul\_column\_num*, DECL\_DATETIME \*) method

Sets a column to a DECL\_DATETIME value.

**Syntax**

```
public virtual bool SetDateTime(  
    ul_column_num cid,  
    DECL_DATETIME * value  
)
```

**Parameters**

- **cid** The 1-based ordinal column number.
- **value** The DECL\_DATETIME value. Passing NULL is equivalent to calling SetNull.

**Returns**

True on success, otherwise false.

## SetDefault method

Sets a column to its default value.

**Overload list**

Name	Description
<a href="#">“SetDefault(const char *) method”</a>	Sets a column to its default value.
<a href="#">“SetDefault(const ul_wchar *) method”</a>	Sets a column to its default value.
<a href="#">“SetDefault(ul_column_num) method”</a>	Sets a column to its default value.

## SetDefault(const char \*) method

Sets a column to its default value.

**Syntax**

```
public virtual bool SetDefault(const char * cname)
```

**Parameters**

- **cname** The name of the column.



**Returns**

True on success, otherwise false.

**SetDefault(const ul\_wchar \*) method**

Sets a column to its default value.

**Syntax**

```
public virtual bool SetDefault(const ul_wchar * cname)
```

**Parameters**

- **cname** The name of the column.

**Returns**

True on success, otherwise false.

**SetDefault(ul\_column\_num) method**

Sets a column to its default value.

**Syntax**

```
public virtual bool SetDefault(ul_column_num cid)
```

**Parameters**

- **cid** The 1-based ordinal column number.

**Returns**

True on success, otherwise false.

**SetDouble method**

Sets a column to a double value.

**Overload list**

Name	Description
<a href="#">“SetDouble(const char *, ul_double) method”</a>	Sets a column to a double value.
<a href="#">“SetDouble(const ul_wchar *, ul_double) method”</a>	Sets a column to a double value.
<a href="#">“SetDouble(ul_column_num, ul_double) method”</a>	Sets a column to a double value.

## SetDouble(const char \*, ul\_double) method

Sets a column to a double value.

### Syntax

```
public virtual bool SetDouble(const char * cname, ul_double value)
```

### Parameters

- **cname** The name of the column.
- **value** The double value.

### Returns

True on success, otherwise false.

## SetDouble(const ul\_wchar \*, ul\_double) method

Sets a column to a double value.

### Syntax

```
public virtual bool SetDouble(const ul_wchar * cname, ul_double value)
```

### Parameters

- **cname** The name of the column.
- **value** The double value.

### Returns

True on success, otherwise false.

## SetDouble(ul\_column\_num, ul\_double) method

Sets a column to a double value.

### Syntax

```
public virtual bool SetDouble(ul_column_num cid, ul_double value)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **value** The double value.

### Returns

True on success, otherwise false.

## SetFloat method

Sets a column to a float value.

### Overload list

Name	Description
<a href="#">“SetFloat(const char *, ul_real) method”</a>	Sets a column to a float value.
<a href="#">“SetFloat(const ul_wchar *, ul_real) method”</a>	Sets a column to a float value.
<a href="#">“SetFloat(ul_column_num, ul_real) method”</a>	Sets a column to a float value.

### SetFloat(const char \*, ul\_real) method

Sets a column to a float value.

#### Syntax

```
public virtual bool SetFloat(const char * cname, ul_real value)
```

#### Parameters

- **cname** The name of the column.
- **value** The float value.

#### Returns

True on success, otherwise false.

### SetFloat(const ul\_wchar \*, ul\_real) method

Sets a column to a float value.

#### Syntax

```
public virtual bool SetFloat(const ul_wchar * cname, ul_real value)
```

#### Parameters

- **cname** The name of the column.
- **value** The float value.

#### Returns

True on success, otherwise false.

## SetFloat(*ul\_column\_num*, *ul\_real*) method

Sets a column to a float value.

### Syntax

```
public virtual bool SetFloat(ul_column_num cid, ul_real value)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **value** The float value.

### Returns

True on success, otherwise false.

## SetGuid method

Sets a column to a GUID value.

### Overload list

Name	Description
<a href="#">“SetGuid(const char *, GUID *) method”</a>	Sets a column to a GUID value.
<a href="#">“SetGuid(const ul_wchar *, GUID *) method”</a>	Sets a column to a GUID value.
<a href="#">“SetGuid(<i>ul_column_num</i>, GUID *) method”</a>	Sets a column to a GUID value.

## SetGuid(const char \*, GUID \*) method

Sets a column to a GUID value.

### Syntax

```
public virtual bool SetGuid(const char * cname, GUID * value)
```

### Parameters

- **cname** The name of the column.
- **value** The GUID value. Passing NULL is equivalent to calling SetNull.

### Returns

True on success, otherwise false.

## SetGuid(const ul\_wchar \*, GUID \*) method

Sets a column to a GUID value.

### Syntax

```
public virtual bool SetGuid(const ul_wchar * cname, GUID * value)
```

### Parameters

- **cname** The name of the column.
- **value** The GUID value. Passing NULL is equivalent to calling SetNull.

### Returns

True on success, otherwise false.

## SetGuid(ul\_column\_num, GUID \*) method

Sets a column to a GUID value.

### Syntax

```
public virtual bool SetGuid(ul_column_num cid, GUID * value)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **value** The GUID value. Passing NULL is equivalent to calling SetNull.

### Returns

True on success, otherwise false.

## SetInt method

Sets a column to an integer value.

### Overload list

Name	Description
<a href="#">“SetInt(const char *, ul_s_long) method”</a>	Sets a column to an integer value.
<a href="#">“SetInt(const ul_wchar *, ul_s_long) method”</a>	Sets a column to an integer value.
<a href="#">“SetInt(ul_column_num, ul_s_long) method”</a>	Sets a column to an integer value.

## SetInt(const char \*, ul\_s\_long) method

Sets a column to an integer value.

### Syntax

```
public virtual bool SetInt(const char * cname, ul_s_long value)
```

### Parameters

- **cname** The name of the column.
- **value** The signed integer value.

### Returns

True on success, otherwise false.

## SetInt(const ul\_wchar \*, ul\_s\_long) method

Sets a column to an integer value.

### Syntax

```
public virtual bool SetInt(const ul_wchar * cname, ul_s_long value)
```

### Parameters

- **cname** The name of the column.
- **value** The signed integer value.

### Returns

True on success, otherwise false.

## SetInt(ul\_column\_num, ul\_s\_long) method

Sets a column to an integer value.

### Syntax

```
public virtual bool SetInt(ul_column_num cid, ul_s_long value)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **value** The signed integer value.

### Returns

True on success, otherwise false.

## SetIntWithType method

Sets a column to an integer value of the specified integer type.

### Overload list

Name	Description
“SetIntWithType(const char *, ul_s_big, ul_column_storage_type) method”	Sets a column to an integer value of the specified integer type.
“SetIntWithType(const ul_wchar *, ul_s_big, ul_column_storage_type) method”	Sets a column to an integer value of the specified integer type.
“SetIntWithType(ul_column_num, ul_s_big, ul_column_storage_type) method”	Sets a column to an integer value of the specified integer type.

## SetIntWithType(const char \*, ul\_s\_big, ul\_column\_storage\_type) method

Sets a column to an integer value of the specified integer type.

### Syntax

```
public virtual bool SetIntWithType(
    const char * cname,
    ul_s_big value,
    ul_column_storage_type type
)
```

### Parameters

- **cname** The name of the column.
- **value** The integer value.
- **type** The integer type to treat the value as.

### Returns

True on success, otherwise false.

### Remarks

The following is a list of integer values that can be used for the value parameter:

- UL\_TYPE\_BIT
- UL\_TYPE\_TINY

- `UL_TYPE_S_SHORT`
- `UL_TYPE_U_SHORT`
- `UL_TYPE_S_LONG`
- `UL_TYPE_U_LONG`
- `UL_TYPE_S_BIG`
- `UL_TYPE_U_BIG`

## **SetIntWithType(const ul\_wchar \*, ul\_s\_big, ul\_column\_storage\_type) method**

Sets a column to an integer value of the specified integer type.

### **Syntax**

```
public virtual bool SetIntWithType(  
    const ul_wchar * cname,  
    ul_s_big value,  
    ul_column_storage_type type  
)
```

### **Parameters**

- **cname** The name of the column.
- **value** The integer value.
- **type** The integer type to treat the value as.

### **Returns**

True on success, otherwise false.

### **Remarks**

The following is a list of integer values that can be used for the value parameter:

- `UL_TYPE_BIT`
- `UL_TYPE_TINY`
- `UL_TYPE_S_SHORT`
- `UL_TYPE_U_SHORT`
- `UL_TYPE_S_LONG`
- `UL_TYPE_U_LONG`



- UL\_TYPE\_S\_BIG
- UL\_TYPE\_U\_BIG

## SetIntWithType(*ul\_column\_num*, *ul\_s\_big*, *ul\_column\_storage\_type*) method

Sets a column to an integer value of the specified integer type.

### Syntax

```
public virtual bool SetIntWithType(  
    ul_column_num cid,  
    ul_s_big value,  
    ul_column_storage_type type  
)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **value** The integer value.
- **type** The integer type to treat the value as.

### Returns

True on success, otherwise false.

### Remarks

The following is a list of integer values that can be used for the value parameter:

- UL\_TYPE\_BIT
- UL\_TYPE\_TINY
- UL\_TYPE\_S\_SHORT
- UL\_TYPE\_U\_SHORT
- UL\_TYPE\_S\_LONG
- UL\_TYPE\_U\_LONG
- UL\_TYPE\_S\_BIG
- UL\_TYPE\_U\_BIG

## SetNull method

Sets a column to null.

#### Overload list

Name	Description
<a href="#">“SetNull(const char *) method”</a>	Sets a column to null.
<a href="#">“SetNull(const ul_wchar *) method”</a>	Sets a column to null.
<a href="#">“SetNull(ul_column_num) method”</a>	Sets a column to null.

## SetNull(const char \*) method

Sets a column to null.

#### Syntax

```
public virtual bool SetNull(const char * cname)
```

#### Parameters

- **cname** The name of the column.

#### Returns

True on success, otherwise false.

## SetNull(const ul\_wchar \*) method

Sets a column to null.

#### Syntax

```
public virtual bool SetNull(const ul_wchar * cname)
```

#### Parameters

- **cname** The name of the column.

#### Returns

True on success, otherwise false.

## SetNull(ul\_column\_num) method

Sets a column to null.

#### Syntax

```
public virtual bool SetNull(ul_column_num cid)
```

**Parameters**

- **cid** The 1-based ordinal column number.

**Returns**

True on success, otherwise false.

## SetString method

Sets a column to a string value.

**Overload list**

Name	Description
<a href="#">“SetString(const char *, const char *, size_t) method”</a>	Sets a column to a string value.
<a href="#">“SetString(const ul_wchar *, const ul_wchar *, size_t) method”</a>	Sets a column to a wide string value.
<a href="#">“SetString(ul_column_num, const char *, size_t) method”</a>	Sets a column to a string value.
<a href="#">“SetString(ul_column_num, const ul_wchar *, size_t) method”</a>	Sets a column to a wide string value.

## SetString(const char \*, const char \*, size\_t) method

Sets a column to a string value.

**Syntax**

```
public virtual bool SetString(
    const char * cname,
    const char * value,
    size_t len
)
```

**Parameters**

- **cname** The name of the column.
- **value** The string value. Passing NULL is equivalent to calling SetNull.
- **len** The length of the string in bytes or UL\_NULL\_TERMINATED\_STRING if the string is null-terminated. SQLE\_INVALID\_PARAMETER is set if len > 32K. For large strings, call AppendStringChunk instead.

**Returns**

True on success, otherwise false.

## SetString(const ul\_wchar \*, const ul\_wchar \*, size\_t) method

Sets a column to a wide string value.

### Syntax

```
public virtual bool SetString(  
    const ul_wchar * cname,  
    const ul_wchar * value,  
    size_t len  
)
```

### Parameters

- **cname** The name of the column.
- **value** The wide string value. Passing NULL is equivalent to calling SetNull.
- **len** The length of the string in ul\_wchars or UL\_NULL\_TERMINATED\_STRING if the string is null-terminated. SQLE\_INVALID\_PARAMETER is set if len > 32K. For large strings, call AppendStringChunk instead.

### Returns

True on success, otherwise false.

## SetString(ul\_column\_num, const char \*, size\_t) method

Sets a column to a string value.

### Syntax

```
public virtual bool SetString(  
    ul_column_num cid,  
    const char * value,  
    size_t len  
)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **value** The string value. Passing NULL is equivalent to calling SetNull.
- **len** The length of the string in bytes or UL\_NULL\_TERMINATED\_STRING if the string is null-terminated. SQLE\_INVALID\_PARAMETER is set if len > 32K. For large strings, call AppendStringChunk instead.

### Returns

True on success, otherwise false.

---

## SetString(*ul\_column\_num*, *const ul\_wchar \**, *size\_t*) method

Sets a column to a wide string value.

### Syntax

```
public virtual bool SetString(  
    ul_column_num cid,  
    const ul_wchar * value,  
    size_t len  
)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **value** The wide string value. Passing NULL is equivalent to calling SetNull.
- **len** The length of the string in ul\_wchars or UL\_NULL\_TERMINATED\_STRING if the string is null-terminated. SQLE\_INVALID\_PARAMETER is set if len > 32K. For large strings, call AppendStringChunk instead.

### Returns

True on success, otherwise false.

## Update method

Updates the current row.

### Syntax

```
public virtual bool Update()
```

### Returns

True on success, otherwise false.

## UpdateBegin method

Selects update mode for setting columns.

### Syntax

```
public virtual bool UpdateBegin()
```

### Returns

True on success, otherwise false.

**Remarks**

Columns in the primary key may not be modified when in update mode. If this cursor is a table opened without an index, then the data is considered read only and cannot be modified.

## ULResultSetSchema class

Represents the schema of an UltraLite result set.

**Syntax**

```
public class ULResultSetSchema
```

**Derived classes**

- [“ULTableSchema class” on page 243](#)

**Members**

All members of ULResultSetSchema class, including all inherited members.

Name	Description
<a href="#">“GetColumnCount method”</a>	Gets the number of columns in the result set or table.
<a href="#">“GetColumnID method”</a>	Gets the 1-based column ID from its name.
<a href="#">“GetColumnName method”</a>	Gets the name of a column given its 1-based ID.
<a href="#">“GetColumnNameW2 method”</a>	Gets the name of a column given its 1-based ID.
<a href="#">“GetColumnPrecision method”</a>	Gets the precision of a numeric column.
<a href="#">“GetColumnScale method”</a>	Gets the scale of a numeric column.
<a href="#">“GetColumnSize method”</a>	Gets the size of the column.
<a href="#">“GetColumnSQLType method”</a>	Gets the SQL type of a column.
<a href="#">“GetColumnType method”</a>	Gets the storage/host variable type of a column.
<a href="#">“GetConnection method”</a>	Gets the connection object.
<a href="#">“IsAliased method”</a>	Indicates whether the column in a result set was given an alias.

## GetColumnCount method

Gets the number of columns in the result set or table.

**Syntax**

```
public virtual ul_column_num GetColumnCount()
```

**Returns**

The number of columns in the result set or table.

## GetColumnID method

Gets the 1-based column ID from its name.

**Overload list**

Name	Description
<a href="#">“GetColumnID(const char *) method”</a>	Gets the 1-based column ID from its name.
<a href="#">“GetColumnID(const ul_wchar *) method”</a>	Gets the 1-based column ID from its name.

## GetColumnID(const char \*) method

Gets the 1-based column ID from its name.

**Syntax**

```
public virtual ul_column_num GetColumnID(const char * columnName)
```

**Parameters**

- **columnName** The column name.

**Returns**

0 if the column does not exist. Sets SQLE\_COLUMN\_NOT\_FOUND if the column name does not exist.

## GetColumnID(const ul\_wchar \*) method

Gets the 1-based column ID from its name.

**Syntax**

```
public virtual ul_column_num GetColumnID(const ul_wchar * columnName)
```

**Parameters**

- **columnName** The column name.

**Returns**

0 if the column does not exist. Sets SQLE\_COLUMN\_NOT\_FOUND if the column name does not exist.

## GetColumnName method

Gets the name of a column given its 1-based ID.

### Syntax

```
public virtual const char * GetColumnName(  
    ul_column_num cid,  
    ul_column_name_type type  
)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **type** The desired column name type.

### Returns

If found, a pointer to a string buffer containing the column name. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while. NULL is returned if the column does not exist. `SQL_COLUMN_NOT_FOUND` is set if the column does not exist.

### Remarks

Depending on the type selected and how the column was declared in the SELECT statement, the column name may be returned in the form [table- name].[column-name].

The type parameter is used to specify what type of column name to return. See the documentation for `ul_column_name_type`.

## GetColumnNameW2 method

Gets the name of a column given its 1-based ID.

### Syntax

```
public virtual const ul_wchar * GetColumnNameW2(  
    ul_column_num cid,  
    ul_column_name_type type  
)
```

### Parameters

- **cid** The 1-based ordinal column number.
- **type** The desired column name type.

### Returns

If found, a pointer to a string buffer containing the column name. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the



value if you need to keep it for a while. NULL is returned if the column does not exist. SQLE\_COLUMN\_NOT\_FOUND is set if the column does not exist.

### Remarks

Depending on the type selected and how the column was declared in the SELECT statement, the column name may be returned in the form [table- name].[column-name].

The type parameter is used to specify what type of column name to return. See the documentation for ul\_column\_name\_type.

## GetColumnPrecision method

Gets the precision of a numeric column.

### Syntax

```
public virtual size_t GetColumnPrecision(ul_column_num cid)
```

### Parameters

- **cid** The 1-based ordinal column number.

### Returns

0 if the column is not a numeric type or if the column does not exist. SQLE\_COLUMN\_NOT\_FOUND is set if the column name does not exist. SQLE\_DATATYPE\_NOT\_ALLOWED is set if the column type is not numeric.

## GetColumnScale method

Gets the scale of a numeric column.

### Syntax

```
public virtual size_t GetColumnScale(ul_column_num cid)
```

### Parameters

- **cid** The 1-based ordinal column number.

### Returns

0 if the column is not a numeric type or if the column does not exist. SQLE\_COLUMN\_NOT\_FOUND is set if the column name does not exist. SQLE\_DATATYPE\_NOT\_ALLOWED is set if the column type is not numeric.

## GetColumnSize method

Gets the size of the column.

### Syntax

```
public virtual size_t GetColumnSize(ul_column_num cid)
```

### Parameters

- **cid** The 1-based ordinal column number.

### Returns

0 if the column does not exist or if the column type does not have a variable length.  
SQLE\_COLUMN\_NOT\_FOUND is set if the column name does not exist.  
SQLE\_DATATYPE\_NOT\_ALLOWED is set if the column type is not UL\_SQLTYPE\_CHAR or UL\_SQLTYPE\_BINARY.

## GetColumnSQLType method

Gets the SQL type of a column.

### Syntax

```
public virtual ul_column_sql_type GetColumnSQLType(ul_column_num cid)
```

### Parameters

- **cid** The 1-based ordinal column number.

### Returns

UL\_SQLTYPE\_BAD\_INDEX if the column does not exist.

### Remarks

For more information about supported SQL types, see [“ul\\_column\\_sql\\_type enumeration”](#) on page 70.

## GetColumnType method

Gets the storage/host variable type of a column.

### Syntax

```
public virtual ul_column_storage_type GetColumnType(ul_column_num cid)
```

### Parameters

- **cid** The 1-based ordinal column number.

### Returns

UL\_TYPE\_BAD\_INDEX if the column does not exist.

**Remarks**

For more information about supported storage types, see [“ul\\_column\\_storage\\_type enumeration” on page 72](#).

## GetConnection method

Gets the connection object.

**Syntax**

```
public virtual ULConnection * GetConnection()
```

**Returns**

The ULConnection associated with this result set schema.

## IsAliased method

Indicates whether the column in a result set was given an alias.

**Syntax**

```
public virtual bool IsAliased(ul_column_num cid)
```

**Parameters**

- **cid** The 1-based ordinal column number.

**Returns**

True if the column is aliased; otherwise, false is returned.

## ULTable class

Represents a table in an UltraLite database.

**Syntax**

```
public class ULTable : ULResultSet
```

**Base classes**

- [“ULResultSet class” on page 179](#)

**Members**

All members of ULTable class, including all inherited members.

Name	Description
“AfterLast method”	Moves the cursor after the last row.
“AppendByteChunk method”	Appends bytes to a column.
“AppendStringChunk method”	Appends a string chunk to a column.
“BeforeFirst method”	Moves the cursor before the first row.
“Close method”	Destroys this object.
“Delete method”	Deletes the current row and moves it to the next valid row.
“DeleteAllRows method”	Deletes all rows from table.
“DeleteNamed method”	Deletes the current row and moves it to the next valid row.
“Find method”	Performs an exact match lookup based on the current index scanning forward through the table.
“FindBegin method”	Prepares to perform a new Find on a table by entering find mode.
“FindFirst method”	Performs an exact match lookup based on the current index scanning forward through the table.
“FindLast method”	Performs an exact match lookup based on the current index scanning backward through the table.
“FindNext method”	Gets the next row that exactly matches the index.
“FindPrevious method”	Gets the previous row that exactly matches the index.
“First method”	Moves the cursor to the first row.
“GetBinary method”	Fetches a value from a column as a ul_binary.
“GetBinaryLength method”	Gets the binary length of the value of a column.
“GetByteChunk method”	Gets a binary chunk from the column.
“GetConnection method”	Gets the connection object.
“GetDateTime method”	Fetches a value from a column as a DECL_DATETIME.
“GetDouble method”	Fetches a value from a column as a double.
“GetFloat method”	Fetches a value from a column as a float.
“GetGuid method”	Fetches a value from a column as a GUID.

Name	Description
“GetInt method”	Fetches a value from a column as an integer.
“GetIntWithType method”	Fetches a value from a column as the specified integer type.
“GetResultSetSchema method”	Returns an object that can be used to get information about the result set.
“GetRowCount method”	Gets the number of rows in the table.
“GetState method”	Gets the internal state of the cursor.
“GetString method”	Fetches a value from a column as a null-terminated string.
“GetStringChunk method”	Gets a string chunk from the column.
“GetStringLength method”	Gets the string length of the value of a column.
“GetTableSchema method”	Returns a ULTableSchema object that can be use to get schema information about the table.
“Insert method”	Inserts a new row into the table.
“InsertBegin method”	Selects insert mode for setting columns.
“IsNull method”	Checks if a column is NULL.
“Last method”	Moves the cursor to the last row.
“Lookup method”	Performs a lookup based on the current index scanning forward through the table.
“LookupBackward method”	Performs a lookup based on the current index scanning backward through the table.
“LookupBegin method”	Prepares to perform a new Find on a table by entering lookup mode.
“LookupForward method”	Performs a lookup based on the current index scanning forward through the table.
“Next method”	Moves the cursor forward one row.
“Previous method”	Moves the cursor back one row.
“Relative method”	Moves the cursor by offset rows from the current cursor position.
“SetBinary method”	Sets a column to a ul_binary value.

Name	Description
“SetDateTime method”	Sets a column to a DECL_DATETIME value.
“SetDefault method”	Sets a column to its default value.
“SetDouble method”	Sets a column to a double value.
“SetFloat method”	Sets a column to a float value.
“SetGuid method”	Sets a column to a GUID value.
“SetInt method”	Sets a column to an integer value.
“SetIntWithType method”	Sets a column to an integer value of the specified integer type.
“SetNull method”	Sets a column to null.
“SetString method”	Sets a column to a string value.
“TruncateTable method”	Truncates the table and temporarily activates STOP SYNCHRONIZATION DELETE.
“Update method”	Updates the current row.
“UpdateBegin method”	Selects update mode for setting columns.

## DeleteAllRows method

Deletes all rows from table.

### Syntax

```
public virtual bool DeleteAllRows()
```

### Returns

True on success. False on failure. For example, the table is not open, or there was a SQL error, and so on.

### Remarks

In some applications, you may want to delete all rows from a table before downloading a new set of data into the table. If you set the stop sync property on the connection, the deleted rows are not synchronized.

#### Note

Any uncommitted inserts from other connections are not deleted. Also, any uncommitted deletes from other connections are not deleted, if the other connection does a rollback after it calls DeleteAllRows().

If this table has been opened without an index then it is considered read only and data cannot be deleted.

## Find method

Performs an exact match lookup based on the current index scanning forward through the table.

### Syntax

```
public virtual bool Find(ul_column_num ncols)
```

### Parameters

- **ncols** For composite indexes, the number of columns to use in the find.

### Returns

If no row matches the index value, the cursor position is AfterLast() and the function returns false.

### Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is positioned on the first row that exactly matches the index value.

## FindBegin method

Prepares to perform a new Find on a table by entering find mode.

### Syntax

```
public virtual bool FindBegin()
```

### Returns

True on success, otherwise false.

### Remarks

You may only set columns in the index that the table was opened with. If the table was opened without an index, this method cannot be called.

## FindFirst method

Performs an exact match lookup based on the current index scanning forward through the table.

### Syntax

```
public virtual bool FindFirst(ul_column_num ncols)
```

### Parameters

- **ncols** For composite indexes, the number of columns to use in the find.

### Returns

If no row matches the index value, the cursor position is `AfterLast()` and the function returns false.

### Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is positioned on the first row that exactly matches the index value.

## FindLast method

Performs an exact match lookup based on the current index scanning backward through the table.

### Syntax

```
public virtual bool FindLast(ul_column_num ncols)
```

### Parameters

- **ncols** For composite indexes, the number of columns to use in the find.

### Returns

If no row matches the index value, the cursor position is `BeforeFirst()` and the function returns false.

### Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is positioned on the first row that exactly matches the index value.

## FindNext method

Gets the next row that exactly matches the index.

### Syntax

```
public virtual bool FindNext(ul_column_num ncols)
```

### Parameters

- **ncols** For composite indexes, the number of columns to use in the find.

### Returns

False if no more rows match the index. In this case the cursor is positioned after the last row.

## FindPrevious method

Gets the previous row that exactly matches the index.



**Syntax**

```
public virtual bool FindPrevious(ul_column_num ncols)
```

**Parameters**

- **ncols** For composite indexes, the number of columns to use in the find.

**Returns**

False if no more rows match the index. In this case the cursor cursor is positioned before the first row.

## GetTableSchema method

Returns a ULTableSchema object that can be use to get schema information about the table.

**Syntax**

```
public virtual ULTableSchema * GetTableSchema()
```

**Returns**

A ULTableSchema object that can be used to get schema information about the table.

## Insert method

Inserts a new row into the table.

**Syntax**

```
public virtual bool Insert()
```

**Returns**

True on success, otherwise false.

## InsertBegin method

Selects insert mode for setting columns.

**Syntax**

```
public virtual bool InsertBegin()
```

**Returns**

True on success, otherwise false.

**Remarks**

All columns are set to their default value on Insert unless an alternative value is supplied via the Set calls.

## Lookup method

Performs a lookup based on the current index scanning forward through the table.

### Syntax

```
public virtual bool Lookup(ul_column_num ncols)
```

### Parameters

- **ncols** For composite indexes, the number of columns to use in the lookup.

### Returns

If resulting cursor position is AfterLast(), the return value is false.

### Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is positioned on the last row that matches or is less than the index value. For composite indexes, *ncols* specifies the number of columns to use in the lookup.

## LookupBackward method

Performs a lookup based on the current index scanning backward through the table.

### Syntax

```
public virtual bool LookupBackward(ul_column_num ncols)
```

### Parameters

- **ncols** For composite indexes, the number of columns to use in the lookup.

### Returns

If resulting cursor position is BeforeFirst(), the return value is false.

### Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is positioned on the last row that matches or is less than the index value. For composite indexes, *ncols* specifies the number of columns to use in the lookup.

## LookupBegin method

Prepares to perform a new Find on a table by entering lookup mode.

### Syntax

```
public virtual bool LookupBegin()
```

**Returns**

True on success, otherwise false.

**Remarks**

You may only set columns in the index that the table was opened with. If the table was opened without an index, this method cannot be called.

## LookupForward method

Performs a lookup based on the current index scanning forward through the table.

**Syntax**

```
public virtual bool LookupForward(ul_column_num ncols)
```

**Parameters**

- **ncols** For composite indexes, the number of columns to use in the lookup.

**Returns**

If resulting cursor position is AfterLast(), the return value is false.

**Remarks**

To specify the value to search for, set the column value for each column column in the index. The cursor is positioned on the last row that matches or is less than the index value. For composite indexes, *ncols* specifies the number of columns to use in the lookup.

## TruncateTable method

Truncates the table and temporarily activates STOP SYNCHRONIZATION DELETE.

**Syntax**

```
public virtual bool TruncateTable()
```

**Returns**

True on success, otherwise false.

**Remarks**

If this table has been opened without an index then it is considered read only and data cannot be deleted.

## ULTableSchema class

Represents the schema of an UltraLite table.

**Syntax**

```
public class ULTableSchema : ULResultSetSchema
```

**Base classes**

- [“ULResultSetSchema class” on page 230](#)

**Members**

All members of ULTableSchema class, including all inherited members.

<b>Name</b>	<b>Description</b>
<a href="#">“Close method”</a>	Destroys this object.
<a href="#">“GetColumnCount method”</a>	Gets the number of columns in the result set or table.
<a href="#">“GetColumnDefault method”</a>	Gets the default value for the column if it exists.
<a href="#">“GetColumnDefaultType method”</a>	Gets the type of column default.
<a href="#">“GetColumnDefaultW2 method”</a>	Gets the default value for the column if it exists.
<a href="#">“GetColumnID method”</a>	Gets the 1-based column ID from its name.
<a href="#">“GetColumnName method”</a>	Gets the name of a column given its 1-based ID.
<a href="#">“GetColumnNameW2 method”</a>	Gets the name of a column given its 1-based ID.
<a href="#">“GetColumnPrecision method”</a>	Gets the precision of a numeric column.
<a href="#">“GetColumnScale method”</a>	Gets the scale of a numeric column.
<a href="#">“GetColumnSize method”</a>	Gets the size of the column.
<a href="#">“GetColumnSQLType method”</a>	Gets the SQL type of a column.
<a href="#">“GetColumnType method”</a>	Gets the storage/host variable type of a column.
<a href="#">“GetConnection method”</a>	Gets the connection object.
<a href="#">“GetGlobalAutoincPartitionSize method”</a>	Gets the partition size.
<a href="#">“GetIndexCount method”</a>	Gets the number of indexes in the table.
<a href="#">“GetIndexSchema method”</a>	Gets the schema of an index given its name.
<a href="#">“GetName method”</a>	Gets the name of the table.
<a href="#">“GetNameW2 method”</a>	Gets the name of the table.

Name	Description
“ <a href="#">GetNextIndex method</a> ”	Gets the next index (schema) in the table.
“ <a href="#">GetOptimalIndex method</a> ”	Determines the best index to use for searching for a column value.
“ <a href="#">GetOptimalIndexW2 method</a> ”	Determines the best index to use for searching for a column value.
“ <a href="#">GetPrimaryKey method</a> ”	Gets the primary key for the table.
“ <a href="#">GetPublicationPredicate method</a> ”	Gets the publication predicate as a string.
“ <a href="#">GetTableSyncType method</a> ”	Gets the table synchronization type.
“ <a href="#">InPublication method</a> ”	Checks whether the table is contained in the named publication.
“ <a href="#">IsAliased method</a> ”	Indicates whether the column in a result set was given an alias.
“ <a href="#">IsColumnInIndex method</a> ”	Checks whether the column is contained in the named index.
“ <a href="#">IsColumnNullable method</a> ”	Checks whether the specified column is nullable.

## Close method

Destroys this object.

### Syntax

```
public virtual void Close()
```

## GetColumnDefault method

Gets the default value for the column if it exists.

### Syntax

```
public virtual const char * GetColumnDefault(ul_column_num cid)
```

### Parameters

- **cid** A 1-based ordinal column number.

## Returns

The default value. The empty string is returned if the column has no default. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetColumnDefaultType method

Gets the type of column default.

### Syntax

```
public virtual ul_column_default_type GetColumnDefaultType(  
    ul_column_num cid  
)
```

### Parameters

- **cid** A 1-based ordinal column number.

### Returns

The type of column default.

### See also

- [“ul\\_column\\_default\\_type enumeration” on page 254](#)

## GetColumnDefaultW2 method

Gets the default value for the column if it exists.

### Syntax

```
public virtual const ul_wchar * GetColumnDefaultW2(ul_column_num cid)
```

### Parameters

- **cid** A 1-based ordinal column number.

### Returns

The default value. The empty string is returned if the column has no default. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetGlobalAutoincPartitionSize method

Gets the partition size.

**Syntax**

```
public virtual bool GetGlobalAutoincPartitionSize(
    ul_column_num cid,
    ul_u_big * size
)
```

**Parameters**

- **cid** A 1-based ordinal column number.
- **size** An output parameter. The partition size for the column. All global autoincrement columns in a given table share the same global autoincrement partition.

**Returns**

True on success; false on failure.

## GetIndexCount method

Gets the number of indexes in the table.

**Syntax**

```
public virtual ul_index_num GetIndexCount()
```

**Returns**

The number of indexes in the table.

**Remarks**

Index IDs and counts may change during a schema upgrade. To correctly identify an index, access it by name or refresh any cached IDs and counts after a schema upgrade.

## GetIndexSchema method

Gets the schema of an index given its name.

**Overload list**

Name	Description
<a href="#">“GetIndexSchema(const char *) method”</a>	Gets the schema of an index given its name.
<a href="#">“GetIndexSchema(const ul_wchar *) method”</a>	Gets the schema of an index given its name.

## GetIndexSchema(const char \*) method

Gets the schema of an index given its name.

### Syntax

```
public virtual ULIndexSchema * GetIndexSchema(const char * indexName)
```

### Parameters

- **indexName** The name of the index.

### Returns

A ULIndexSchema object for the specified index or NULL if it doesn't exist.

## GetIndexSchema(const ul\_wchar \*) method

Gets the schema of an index given its name.

### Syntax

```
public virtual ULIndexSchema * GetIndexSchema(  
    const ul_wchar * indexName  
)
```

### Parameters

- **indexName** The name of the index.

### Returns

A ULIndexSchema object for the specified index or NULL if it doesn't exist.

## GetName method

Gets the name of the table.

### Syntax

```
public virtual const char * GetName()
```

### Returns

The name of the table. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetNameW2 method

Gets the name of the table.

### Syntax

```
public virtual const ul_wchar * GetNameW2()
```



**Returns**

The name of the table. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetNextIndex method

Gets the next index (schema) in the table.

**Syntax**

```
public virtual ULIndexSchema * GetNextIndex(ul_index_iter * iter)
```

**Parameters**

- **iter** Pointer to iterator variable.

**Returns**

A ULIndexSchema object or NULL when the iteration is complete.

**Remarks**

Initialize the value in *iter* to `ul_index_iter_start` before the first call.

**See also**

- [“ul\\_index\\_iter\\_start variable” on page 257](#)

## GetOptimalIndex method

Determines the best index to use for searching for a column value.

**Syntax**

```
public virtual const char * GetOptimalIndex(ul_column_num cid)
```

**Parameters**

- **cid** A 1-based ordinal column number.

**Returns**

The name of the index or NULL if the column isn't indexed. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetOptimalIndexW2 method

Determines the best index to use for searching for a column value.

**Syntax**

```
public virtual const ul_wchar * GetOptimalIndexW2(ul_column_num cid)
```

**Parameters**

- **cid** A 1-based ordinal column number.

**Returns**

The name of the index or NULL if the column isn't indexed. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetPrimaryKey method

Gets the primary key for the table.

**Syntax**

```
public virtual ULIndexSchema * GetPrimaryKey()
```

**Returns**

a ULIndexSchema object for the table's primary key.

## GetPublicationPredicate method

Gets the publication predicate as a string.

**Overload list**

Name	Description
<a href="#">“GetPublicationPredicate(const char *) method”</a>	Gets the publication predicate as a string.
<a href="#">“GetPublicationPredicate(const ul_wchar *) method”</a>	Gets the publication predicate as a string.

## GetPublicationPredicate(const char \*) method

Gets the publication predicate as a string.

**Syntax**

```
public virtual const char * GetPublicationPredicate(  
    const char * pubName  
)
```

### Parameters

- **pubName** The name of the publication.

### Returns

The publication predicate string for the specified publication. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetPublicationPredicate(const ul\_wchar \*) method

Gets the publication predicate as a string.

### Syntax

```
public virtual const ul_wchar * GetPublicationPredicate(  
    const ul_wchar * pubName  
)
```

### Parameters

- **pubName** The name of the publication.

### Returns

The publication predicate string for the specified publication. Warning: this points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you'll need to make a copy of the value if you need to keep it for a while.

## GetTableSyncType method

Gets the table synchronization type.

### Syntax

```
public virtual ul_table_sync_type GetTableSyncType()
```

### Returns

The synchronization type.

### Remarks

This indicates how the table participates in synchronization, and is defined when the table is created with the SYNCHRONIZE constraint clause of the CREATE TABLE statement.

### See also

- [“ul\\_table\\_sync\\_type enumeration” on page 256](#)

## InPublication method

Checks whether the table is contained in the named publication.

### Overload list

Name	Description
<code>InPublication(const char *) method</code>	Checks whether the table is contained in the named publication.
<code>InPublication(const ul_wchar *) method</code>	Checks whether the table is contained in the named publication.

## InPublication(const char \*) method

Checks whether the table is contained in the named publication.

### Syntax

```
public virtual bool InPublication(const char * pubName)
```

### Parameters

- **pubName** The name of the publication.

### Returns

True if the table is contained in the publication; otherwise, false is returned.

## InPublication(const ul\_wchar \*) method

Checks whether the table is contained in the named publication.

### Syntax

```
public virtual bool InPublication(const ul_wchar * pubName)
```

### Parameters

- **pubName** The name of the publication.

### Returns

True if the table is contained in the publication; otherwise, false is returned.

## IsColumnInIndex method

Checks whether the column is contained in the named index.

**Overload list**

Name	Description
<a href="#">“IsColumnInIndex(ul_column_num, const char *) method”</a>	Checks whether the column is contained in the named index.
<a href="#">“IsColumnInIndex(ul_column_num, const ul_wchar *) method”</a>	Checks whether the column is contained in the named index.

**IsColumnInIndex(ul\_column\_num, const char \*) method**

Checks whether the column is contained in the named index.

**Syntax**

```
public virtual bool IsColumnInIndex(
    ul_column_num cid,
    const char * indexName
)
```

**Parameters**

- **cid** A 1-based ordinal column number.
- **indexName** The name of the index.

**Returns**

True if the column is contained in the index; false, otherwise.

**IsColumnInIndex(ul\_column\_num, const ul\_wchar \*) method**

Checks whether the column is contained in the named index.

**Syntax**

```
public virtual bool IsColumnInIndex(
    ul_column_num cid,
    const ul_wchar * indexName
)
```

**Parameters**

- **cid** A 1-based ordinal column number.
- **indexName** The name of the index.

**Returns**

True if the column is contained in the index; false, otherwise.

## IsColumnNullable method

Checks whether the specified column is nullable.

### Syntax

```
public virtual bool IsColumnNullable(ul_column_num cid)
```

### Parameters

- **cid** A 1-based ordinal column number.

### Returns

True if the column is nullable; false, otherwise.

## ul\_column\_default\_type enumeration

Identifies a column default type.

### Syntax

```
public enum ul_column_default_type
```

### Members

Member name	Description
ul_column_default_none	The column has no default value.
ul_column_default_autoincrement	The column default is AUTOINCREMENT.
ul_column_default_global_autoincrement	The column default is GLOBAL AUTOINCREMENT.
ul_column_default_current_timestamp	The column default is CURRENT TIMESTAMP.
ul_column_default_current_utc_timestamp	The column default is CURRENT UTC TIMESTAMP.
ul_column_default_current_time	The column default is CURRENT TIME.
ul_column_default_current_date	The column default is CURRENT DATE.
ul_column_default_newid	The column default is NEWID().
ul_column_default_other	The column default is a user-specified constant.

### See also

- [“GetColumnDefaultType method” on page 246](#)

## ul\_column\_name\_type enumeration

Specifies values that control how a column name is retrieved when describing a result set.

### Syntax

```
public enum ul_column_name_type
```

### Members

Member name	Description
ul_name_type_sql	For SELECT statements, return the alias or correlation name. For tables, return the column name.
ul_name_type_sql_column_only	For SELECT statements, return the alias or correlation name and exclude any table names that were specified. For tables, return the column name.
ul_name_type_base_table	Return the underlying table name if it can be determined. If the table does not exist in the database schema, an empty string is returned.
ul_name_type_base_column	Return the underlying column name if it can be determined. If the column does not exist in the database schema, an empty string is returned.

### See also

- [“GetColumnName method” on page 232](#)

## ul\_index\_flag enumeration

Flags (bit fields) which identify properties of an index.

### Syntax

```
public enum ul_index_flag
```

### Members

Member name	Description
ul_index_flag_primary_key	The index is a primary key.
ul_index_flag_unique_key	The index is a primary key or index created for a unique constraint (nulls not allowed).
ul_index_flag_unique_index	The index was created with the UNIQUE flag (or is a primary key).
ul_index_flag_foreign_key	The index is a foreign key.

Member name	Description
ul_index_flag_foreign_key_nullable	The foreign key allows nulls.
ul_index_flag_foreign_key_check_on_commit	Referential integrity checks are performed on commit (rather than on insert/update).

**See also**

- [“GetIndexFlags method” on page 164](#)

## ul\_table\_sync\_type enumeration

Identifies a table synchronization type.

**Syntax**

```
public enum ul_table_sync_type
```

**Members**

Member name	Description
ul_table_sync_on	All changed rows are synchronized; default behavior. Corresponds to SYNCHRONIZE ON in CREATE TABLE statement.
ul_table_sync_off	Table is never synchronized. Corresponds to SYNCHRONIZE OFF in CREATE TABLE statement.
ul_table_sync_upload_all_rows	Always upload every row, even unchanged rows. Corresponds to SYNCHRONIZE ALL in CREATE TABLE statement.

**See also**

- [“GetTableSyncType method” on page 251](#)

## UL\_BLOB\_CONTINUE variable

Used when reading data with the `ULResultSet.GetStringChunk()` or `ULResultSet.GetByteChunk()` functions.

**Syntax**

```
#define UL_BLOB_CONTINUE (size_t)(-1)
```



**Remarks**

This value indicates that the chunk of data to be read should continue from where the last chunk was read.

**See also**

- [“GetStringChunk method” on page 206](#)
- [“GetByteChunk method” on page 191](#)

## ul\_index\_iter\_start variable

Used by the `GetNextIndex` method to initialize index iteration in a table.

**Syntax**

```
#define ul_index_iter_start 1
```

**See also**

- [“GetNextIndex method” on page 249](#)

## ul\_publication\_iter\_start variable

Used by the `GetNextPublication` method to initialize publication iteration in a database.

**Syntax**

```
#define ul_publication_iter_start 1
```

**See also**

- [“GetNextPublication method” on page 152](#)

## ul\_table\_iter\_start variable

Used by the `GetNextTable` method to initialize table iteration in a database.

**Syntax**

```
#define ul_table_iter_start 1
```

**See also**

- [“GetNextTable method” on page 153](#)

---

---

# UltraLite Embedded SQL API reference

This section lists functions that support UltraLite functionality in embedded SQL applications.

For general information about SQL statements that can be used, see [“Developing applications using embedded SQL” on page 29](#).

Use the EXEC SQL INCLUDE SQLCA command to include prototypes for the functions in this chapter.

## Header files

- `mfiletransfer.h`
- `ulprotos.h`

## db\_fini method

Frees resources used by the UltraLite runtime library.

### Syntax

```
unsigned short db_fini(SQLCA * sqlca);
```

### Returns

- 0 if an error occurs during processing. The error code is set in SQLCA.
- Non-zero if there are no errors.

### Remarks

You must not make any other UltraLite library call or execute any embedded SQL command after `db_fini` is called.

Call `db_fini` once for each SQLCA being used.

### See also

- [“db\\_init method” on page 259](#)

## db\_init method

Initializes the UltraLite runtime library.

### Syntax

```
unsigned short db_init(SQLCA * sqlca);
```

**Returns**

- 0 if an error occurs during processing (for example, during initialization of the persistent store). The error code is set in SQLCA.
- Non-zero if there are no errors. You can begin using embedded SQL commands and functions.

**Remarks**

You must call this function before you make any other UltraLite library call, and before you execute any embedded SQL command.

Usually you should only call this function once, passing the address of the global sqlca variable (as defined in the *sqlca.h* header file). If you have multiple execution paths in your application, you can use more than one db\_init call, as long as each one has a separate sqlca pointer. This separate SQLCA pointer can be a user-defined one, or could be a global SQLCA that has been freed using db\_fini.

In multi-threaded applications, each thread must call db\_init to obtain a separate SQLCA. Carry out subsequent connections and transactions that use this SQLCA on a single thread.

Initializing the SQLCA also resets any settings from previously called ULEnable functions. If you re-initialize a SQLCA, you must issue any ULEnable functions the application requires.

**See also**

- [“db\\_fini method” on page 259](#)

## MLFTEnableEccE2ee method

Enables the developer to specify the ECC end-to-end encryption feature.

**Syntax**

```
public void MLFTEnableEccE2ee(ml_file_transfer_info * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**See also**

- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLFTEnableEccE2eeW method

Enables the developer to specify the ECC end-to-end encryption feature.

**Syntax**

```
public void MLFTEnableEccE2eeW(ml_file_transfer_info_w * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks****Note**

This method prototype is used internally when you refer to `MLFTEnableEccE2ee` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

**See also**

- [“MLFTEnableEccE2ee method” on page 260](#)
- [“ml\\_file\\_transfer\\_info\\_w structure” on page 322](#)

## MLFTEnableEccEncryption method

Enables the developer to specify the ECC encryption feature.

**Syntax**

```
public void MLFTEnableEccEncryption(ml_file_transfer_info * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**See also**

- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLFTEnableEccEncryptionW method

Enables the developer to specify the ECC encryption feature.

**Syntax**

```
public void MLFTEnableEccEncryptionW(ml_file_transfer_info_w * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks****Note**

This method prototype is used internally when you refer to `MLFTEnableEccEncryption` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

### See also

- [“MLFTEnableEccEncryption method” on page 261](#)
- [“ml\\_file\\_transfer\\_info\\_w structure” on page 322](#)

## MLFTEnableRsaE2ee method

Enables the developer to specify the RSA end-to-end encryption feature.

### Syntax

```
public void MLFTEnableRsaE2ee(ml_file_transfer_info * info)
```

### Parameters

- **info** A structure containing the file transfer information.

### See also

- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLFTEnableRsaE2eeW method

Enables the developer to specify the RSA end-to-end encryption feature.

### Syntax

```
public void MLFTEnableRsaE2eeW(ml_file_transfer_info_w * info)
```

### Parameters

- **info** A structure containing the file transfer information.

### Remarks

**Note**

This method prototype is used internally when you refer to `MLFTEnableRsaE2ee` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

### See also

- [“MLFTEnableRsaE2ee method” on page 262](#)
- [“ml\\_file\\_transfer\\_info\\_w structure” on page 322](#)

## MLFTEnableRsaEncryption method

Enables the developer to specify the RSA encryption feature.

**Syntax**

```
public void MLFTEnableRsaEncryption(ml_file_transfer_info * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**See also**

- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLFTEnableRsaEncryptionW method

Enables the developer to specify the RSA encryption feature.

**Syntax**

```
public void MLFTEnableRsaEncryptionW(ml_file_transfer_info_w * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks****Note**

This method prototype is used internally when you refer to `MLFTEnableRsaEncryption` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

**See also**

- [“MLFTEnableRsaEncryption method” on page 262](#)
- [“ml\\_file\\_transfer\\_info\\_w structure” on page 322](#)

## MLFTEnableRsaFipsE2ee method

Enables the developer to specify the RSAFIPS end-to-end encryption feature.

**Syntax**

```
public void MLFTEnableRsaFipsE2ee(ml_file_transfer_info * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**See also**

- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLFTEnableRsaFipsE2eeW method

Enables the developer to specify the RSAFIPS end-to-end encryption feature.

### Syntax

```
public void MLFTEnableRsaFipsE2eeW(ml_file_transfer_info_w * info)
```

### Parameters

- **info** A structure containing the file transfer information.

### Remarks

#### Note

This method prototype is used internally when you refer to `MLFTEnableRsaFipsE2ee` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

### See also

- [“MLFTEnableRsaFipsE2ee method” on page 263](#)
- [“ml\\_file\\_transfer\\_info\\_w structure” on page 322](#)

## MLFTEnableRsaFipsEncryption method

Enables the developer to specify the RSAFIPS encryption feature.

### Syntax

```
public void MLFTEnableRsaFipsEncryption(ml_file_transfer_info * info)
```

### Parameters

- **info** A structure containing the file transfer information.

### See also

- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLFTEnableRsaFipsEncryptionW method

Enables the developer to specify the RSAFIPS encryption feature.

### Syntax

```
public void MLFTEnableRsaFipsEncryptionW(ml_file_transfer_info_w * info)
```



**Parameters**

- **info** A structure containing the file transfer information.

**Remarks****Note**

This method prototype is used internally when you refer to `MLFTEnableRsaFipsEncryption` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

**See also**

- [“MLFTEnableRsaFipsEncryption method” on page 264](#)
- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLFTEnableZlibCompression method

Enables the developer to specify the ZLIB compression feature.

**Syntax**

```
public void MLFTEnableZlibCompression(ml_file_transfer_info * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**See also**

- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLFTEnableZlibCompressionW method

Enables the developer to specify the ZLIB compression feature.

**Syntax**

```
public void MLFTEnableZlibCompressionW(ml_file_transfer_info_w * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks****Note**

This method prototype is used internally when you refer to `MLFTEnableZlibCompression` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

**See also**

- [“MLFTEnableZlibCompression method” on page 265](#)
- [“ml\\_file\\_transfer\\_info\\_w structure” on page 322](#)

## MLFileDownload method

Downloads a file from a MobiLink server with the MobiLink interface.

**Syntax**

```
public bool MLFileDownload(ml_file_transfer_info * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks**

You must set the source location of the file to be transferred. This location must be specified as a MobiLink user's directory on the MobiLink server (or in the default directory on that server). You can also set the intended target location and file name of the file.

For example, you can program your application to download a new or replacement database from the MobiLink server. You can customize the file for specific users, since the first location that is searched is a specific user's subdirectory. You can also maintain a default file in the root folder on the server, since that location is used if the specified file is not found in the user's folder.

**See also**

- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLFileDownloadW method

Downloads a file from a MobiLink server with the MobiLink interface.

**Syntax**

```
public bool MLFileDownloadW(ml_file_transfer_info_w * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks****Note**

This method prototype is used internally when you refer to `MLFileDownload` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

You must set the source location of the file to be transferred. This location must be specified as a MobiLink user's directory on the MobiLink server (or in the default directory on that server). You can also set the intended target location and file name of the file.

For example, you can program your application to download a new or replacement database from the MobiLink server. You can customize the file for specific users, since the first location that is searched is a specific user's subdirectory. You can also maintain a default file in the root folder on the server, since that location is used if the specified file is not found in the user's folder.

**See also**

- [“MLFileDownload method” on page 266](#)
- [“ml\\_file\\_transfer\\_info\\_w structure” on page 322](#)

## MLFileUpload method

Uploads a file from a MobiLink server with the MobiLink interface.

**Syntax**

```
public bool MLFileUpload(ml_file_transfer_info * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks**

You must set the source location of the file to be transferred. This location must be specified as a MobiLink user's directory on the MobiLink server (or in the default directory on that server). You can also set the intended target location and file name of the file.

For example, you can program your application to upload a new or replacement database from the MobiLink server. You can customize the file for specific users, since the first location that is searched is a specific user's subdirectory. You can also maintain a default file in the root folder on the server, since that location is used if the specified file is not found in the user's folder.

**See also**

- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLFileUploadW method

Uploads a file from a MobiLink server with the MobiLink interface.

**Syntax**

```
public bool MLFileUploadW(ml_file_transfer_info_w * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks****Note**

This method prototype is used internally when you refer to `MLFileUpload` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

You must set the source location of the file to be transferred. This location must be specified as a MobiLink user's directory on the MobiLink server (or in the default directory on that server). You can also set the intended target location and file name of the file.

For example, you can program your application to upload a new or replacement database from the MobiLink server. You can customize the file for specific users, since the first location that is searched is a specific user's subdirectory. You can also maintain a default file in the root folder on the server, since that location is used if the specified file is not found in the user's folder.

**See also**

- [“MLFileUpload method” on page 267](#)
- [“ml\\_file\\_transfer\\_info\\_w structure” on page 322](#)

## MLFinifileTransferInfo method

Finalizes any resources allocated in the `ml_file_transfer_info` structure when it is initialized.

**Syntax**

```
public void MLFinifileTransferInfo(ml_file_transfer_info * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks**

This method should be called after the file upload/download has completed.

**See also**

- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLFinifileTransferInfoW method

Finalizes any resources allocated in the `ml_file_transfer_info` structure when it is initialized.

**Syntax**

```
public void MLFiniFileTransferInfoW(ml_file_transfer_info_w * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks****Note**

This method prototype is used internally when you refer to `MLFiniFileTransferInfo` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

This method should be called after the file upload/download has completed.

**See also**

- [“MLFiniFileTransferInfo method” on page 268](#)
- [“ml\\_file\\_transfer\\_info\\_w structure” on page 322](#)

## MLInitFileTransferInfo method

Initializes the `ml_file_transfer_info` structure.

**Syntax**

```
public bool MLInitFileTransferInfo(ml_file_transfer_info * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks**

This should be called before starting the file upload/download.

**See also**

- [“ml\\_file\\_transfer\\_info structure” on page 320](#)

## MLInitFileTransferInfoW method

Initializes the `ml_file_transfer_info` structure.

**Syntax**

```
public bool MLInitFileTransferInfoW(ml_file_transfer_info_w * info)
```

**Parameters**

- **info** A structure containing the file transfer information.

**Remarks****Note**

This method prototype is used internally when you refer to `MlInitFileTransferInfo` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

This should be called before starting the file upload/download.

**See also**

- [“MlInitFileTransferInfo method” on page 269](#)
- [“ml\\_file\\_transfer\\_info\\_w structure” on page 322](#)

## ULCancelGetNotification method

Cancels any pending get-notification calls on all queues matching the given name.

**Syntax**

```
public ul_u_long ULCancelGetNotification(
    SQLCA * sqlca,
    char const * queue_name
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **queue\_name** The name of the queue.

**Returns**

The number of affected queues (not the number of blocked reads necessarily).

## ULCancelGetNotificationW method

Cancels any pending get-notification calls on all queues matching the given name.

**Syntax**

```
public ul_u_long ULCancelGetNotificationW(
    SQLCA * sqlca,
    ul_wchar const * queue_name
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **queue\_name** The name of the queue.

**Returns**

The number of affected queues (not the number of blocked reads necessarily).

**Remarks****Note**

This method prototype is used internally when you refer to `ULCancelGetNotification` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

## ULChangeEncryptionKey method

Changes the encryption key for an UltraLite database.

**Syntax**

```
public ul_bool ULChangeEncryptionKey(  
    SQLCA * sqlca,  
    char const * new_key  
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA
- **new\_key** The new encryption key

**Remarks**

Applications that call this function must first ensure that the user has either synchronized the database or created a reliable backup copy of the database. It is important to have a reliable backup of the database because `ULChangeEncryptionKey` is an operation that must run to completion. When the database encryption key is changed, every row in the database is first decrypted with the old key and then encrypted with the new key and rewritten. This operation is not recoverable. If the encryption change operation does not complete, the database is left in an invalid state and you cannot access it again.

## ULChangeEncryptionKeyW method

Changes the encryption key for an UltraLite database.

**Syntax**

```
public ul_bool ULChangeEncryptionKeyW(
    SQLCA * sqlca,
    ul_wchar const * new_key
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA
- **new\_key** The new encryption key

**Remarks****Note**

This method prototype is used internally when you refer to `ULChangeEncryptionKey` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

Applications that call this function must first ensure that the user has either synchronized the database or created a reliable backup copy of the database. It is important to have a reliable backup of the database because `ULChangeEncryptionKey` is an operation that must run to completion. When the database encryption key is changed, every row in the database is first decrypted with the old key and then encrypted with the new key and rewritten. This operation is not recoverable. If the encryption change operation does not complete, the database is left in an invalid state and you cannot access it again.

## ULCheckpoint method

Performs a checkpoint operation, flushing any pending committed transactions to the database.

**Syntax**

```
public ul_ret_void ULCheckpoint(SQLCA * sqlca)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.

**Remarks**

Any current transaction is not committed by calling `ULCheckpoint`. The `ULCheckpoint` function is used in conjunction with deferring automatic transaction checkpoints as a performance enhancement.

The `ULCheckpoint` function ensures that all pending committed transactions have been written to the database storage.

## ULCountUploadRows method

Counts the number of rows that need to be uploaded for synchronization.



**Syntax**

```
public ul_u_long ULCountUploadRows(
    SQLCA * sqlca,
    char const * pub_list,
    ul_u_long threshold
)
```

**Parameters**

- **sqlca** A pointer to the SQL.
- **pub\_list** A string containing a comma-separated list of publications to check. An empty string (UL\_SYNC\_ALL macro) implies all tables except tables marked as "no sync". A string containing just an asterisk (UL\_SYNC\_ALL\_PUBS macro) implies all tables referred to in any publication. Some tables may not be part of any publication and are not included if the pub-list string is "\*".
- **threshold** Determines the maximum number of rows to count, thereby limiting the amount of time taken by the call. A threshold of 0 corresponds to no limit (that is, count all rows that need to be synchronized) and A threshold of 1 can be used to quickly determine if any rows need to be synchronized.

**Returns**

The number of rows that need to be synchronized, either in a specified set of publications or in the whole database.

**Remarks**

Use this function to prompt users to synchronize.

The following call checks the entire database for the number of rows to be synchronized:

```
count = ULCountUploadRows( sqlca, UL_SYNC_ALL_PUBS, 0 );
```

The following call checks publications PUB1 and PUB2 for a maximum of 1000 rows:

```
count = ULCountUploadRows( sqlca, UL_TEXT("PUB1,PUB2"), 1000 );
```

The following call checks to see if any rows need to be synchronized in publications PUB1 and PUB2:

```
count = ULCountUploadRows( sqlca, UL_TEXT("PUB1,PUB2"), 1 );
```

## ULCountUploadRowsW method

Counts the number of rows that need to be uploaded for synchronization.

**Syntax**

```
public ul_u_long ULCountUploadRowsW(
    SQLCA * sqlca,
    ul_wchar const * pub_list,
    ul_u_long threshold
)
```

## Parameters

- **sqlca** A pointer to the SQL.
- **pub\_list** A string containing a comma-separated list of publications to check. An empty string (UL\_SYNC\_ALL macro) implies all tables except tables marked as "no sync". A string containing just an asterisk (UL\_SYNC\_ALL\_PUBS macro) implies all tables referred to in any publication. Some tables may not be part of any publication and are not included if the pub-list string is "\*".
- **threshold** Determines the maximum number of rows to count, thereby limiting the amount of time taken by the call. A threshold of 0 corresponds to no limit (that is, count all rows that need to be synchronized) and A threshold of 1 can be used to quickly determine if any rows need to be synchronized.

## Returns

The number of rows that need to be synchronized, either in a specified set of publications or in the whole database.

## Remarks

### Note

This method prototype is used internally when you refer to ULCountUploadRows and #define the UNICODE macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

Use this function to prompt users to synchronize.

The following call checks the entire database for the number of rows to be synchronized:

```
count = ULCountUploadRows( sqlca, UL_SYNC_ALL_PUBS, 0 );
```

The following call checks publications PUB1 and PUB2 for a maximum of 1000 rows:

```
count = ULCountUploadRows( sqlca, UL_TEXT("PUB1,PUB2"), 1000 );
```

The following call checks to see if any rows need to be synchronized in publications PUB1 and PUB2:

```
count = ULCountUploadRows( sqlca, UL_TEXT("PUB1,PUB2"), 1 );
```

# ULCreateDatabase method

Creates an UltraLite database.

## Syntax

```
public ul_bool ULCreateDatabase(  
    SQLCA * sqlca,  
    char const * connect_parms,  
    char const * create_parms,  
    void * reserved  
)
```

**Parameters**

- **sqlca** A pointer to the initialized SQLCA.
- **connect\_parms** A semicolon separated string of connection parameters, which are set as keyword value pairs. The connection string must include the name of the database. These parameters are the same set of parameters that may be specified when you connect to a database.
- **create\_parms** A semicolon separated string of database creation parameters, which are set as keyword value pairs. For example: `page_size=2048;obfuscate=yes`.
- **reserved** This parameter is reserved for future use.

**Returns**

ul\_true if database was successfully created or ul\_false if it the creation was unsuccessful. Typically ul\_false is caused by an invalid file name or access denial.

**Remarks**

The database is created with information provided in two sets of parameters.

The connection-params are standard connection parameters that are applicable whenever the database is accessed (for example, file name, user id, password, optional encryption key, and so on).

The creation-params are parameters that are only relevant when creating a database (for example, obfuscation,page-size, time and date format, and so one). Applications can call this function after initializing the SQLCA.

The following code illustrates using ULCreateDatabase to create an UltraLite database as the file C:\myfile.udb :

```
if( ULCreateDatabase(&sqlca
    ,UL_TEXT("DBF=C:\myfile.udb;uid=DBA;pwd=sql")
    ,ULGetCollation_1250LATIN2()
    ,UL_TEXT("obfuscate=1;page_size=8192")
    ,NULL)
{
    // success
};
```

## ULCreateDatabaseW method

Creates an UltraLite database.

**Syntax**

```
public ul_bool ULCreateDatabaseW(
    SQLCA * sqlca,
    ul_wchar const * connect_parms,
    ul_wchar const * create_parms,
    void * reserved
)
```

## Parameters

- **sqlca** A pointer to the initialized SQLCA.
- **connect\_parms** A semicolon separated string of connection parameters, which are set as keyword value pairs. The connection string must include the name of the database. These parameters are the same set of parameters that may be specified when you connect to a database.
- **create\_parms** A semicolon separated string of database creation parameters, which are set as keyword value pairs. For example: `page_size=2048;obfuscate=yes`.
- **reserved** This parameter is reserved for future use.

## Returns

`ul_true` if database was successfully created or `ul_false` if it the creation was unsuccessful. Typically `ul_false` is caused by an invalid file name or access denial.

## Remarks

### Note

This method prototype is used internally when you refer to `ULCreateDatabase` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

The database is created with information provided in two sets of parameters.

The connection-params are standard connection parameters that are applicable whenever the database is accessed (for example, file name, user id, password, optional encryption key, and so on).

The creation-params are parameters that are only relevant when creating a database (for example, obfuscation,page-size, time and date format, and so one). Applications can call this function after initializing the SQLCA.

The following code illustrates using `ULCreateDatabase` to create an UltraLite database as the file `C:\myfile.udb` :

```
if( ULCreateDatabase(&sqlca
,UL_TEXT("DBF=C:\myfile.udb;uid=DBA;pwd=sql")
,ULGetCollation_1250LATIN2()
,UL_TEXT("obfuscate=1;page_size=8192")
,NULL)
{
    // success
};
```

## ULCreateNotificationQueue method

Creates an event notification queue for this connection.

**Syntax**

```
public ul_bool ULCreateNotificationQueue(
    SQLCA * sqlca,
    char const * name,
    char const * parameters
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **name** Name for the new queue.
- **parameters** Creation parameters; currently unused, set to NULL.

**Returns**

True on success, otherwise false.

**Remarks**

Queue names are scoped per-connection, so different connections can create queues with the same name. When an event notification is sent, all queues in the database with a matching name receive (a separate instance of) the notification. Names are case insensitive. A default queue is created on demand for each connection when calling RegisterForEvent() if no queue is specified. This call fails with an error if the name already exists or isn't valid.

## ULCreateNotificationQueueW method

Creates an event notification queue for this connection.

**Syntax**

```
public ul_bool ULCreateNotificationQueueW(
    SQLCA * sqlca,
    ul_wchar const * name,
    ul_wchar const * parameters
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **name** Name for the new queue.
- **parameters** Creation parameters; currently unused, set to NULL.

**Returns**

True on success, otherwise false.

## Remarks

**Note**

This method prototype is used internally when you refer to `ULCreateNotificationQueue` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

Queue names are scoped per-connection, so different connections can create queues with the same name. When an event notification is sent, all queues in the database with a matching name receive (a separate instance of) the notification. Names are case insensitive. A default queue is created on demand for each connection when calling `RegisterForEvent()` if no queue is specified. This call fails with an error if the name already exists or isn't valid.

## ULDeclareEvent method

Declares an event which can then be registered for and triggered.

### Syntax

```
public ul_bool ULDeclareEvent(SQLCA * sqlca, char const * event_name)
```

### Parameters

- **sqlca** A pointer to the SQLCA
- **event\_name** The name for the new user-defined event.

### Returns

True if the event was declared successfully, false if the name is already used or not valid.

## ULDeclareEventW method

Declares an event which can then be registered for and triggered.

### Syntax

```
public ul_bool ULDeclareEventW(  
    SQLCA * sqlca,  
    ul_wchar const * event_name  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA
- **event\_name** The name for the new user-defined event.

**Returns**

True if the event was declared successfully, false if the name is already used or not valid.

**Remarks****Note**

This method prototype is used internally when you refer to `ULDeclareEvent` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

## ULDeleteAllRows method

Deletes all rows from table.

**Syntax**

```
public ul_ret_void ULDeleteAllRows(SQLCA * sqlca, ul_table_num number)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **number** The id of the table to truncate.

**Returns**

True on success. False on failure. For example, the table is not open, or there was a SQL error, and so on.

**Remarks**

In some applications, you may want to delete all rows from a table before downloading a new set of data into the table. If you set the stop sync property on the connection, the deleted rows are not synchronized.

Note: Any uncommitted inserts from other connections are not deleted. Also, any uncommitted deletes from other connections are not deleted, if the other connection does a rollback after it calls `DeleteAllRows()`.

If this table has been opened without an index then it is considered read-only and data cannot be deleted.

## ULDestroyNotificationQueue method

Destroys the given event notification queue.

**Syntax**

```
public ul_bool ULDestroyNotificationQueue(  
    SQLCA * sqlca,  
    char const * name  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **name** The name of the queue to destroy.

### Returns

True on success, otherwise false.

## ULDestroyNotificationQueueW method

Destroys the given event notification queue.

### Syntax

```
public ul_bool ULDestroyNotificationQueueW(  
    SQLCA * sqlca,  
    ul_wchar const * name  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **name** The name of the queue to destroy.

### Returns

True on success, otherwise false.

### Remarks

**Note**

This method prototype is used internally when you refer to `ULDestroyNotificationQueue` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

## ULECCLibraryVersion method

Returns the version number of the ECC encryption library.

### Syntax

```
public char const * ULECCLibraryVersion(ul_arg_void )
```

### Returns

The version number of the ECC encryption library.



## ULEnableAesDBEncryption method

Enables strong encryption.

### Syntax

```
public ul_ret_void ULEnableAesDBEncryption(SQLCA * sqlca)
```

### Parameters

- **sqlca** A pointer to the initialized SQLCA.

### Remarks

This was formerly called ULEnableStrongEncryption.

You can use this method in C++ API applications and embedded SQL applications. You must call this method before ULEnableDatabaseManager.

#### Note

Calling this method causes the encryption routines to be included in the application and increases the size of the application code.

## ULEnableAesFipsDBEncryption method

Enables FIPS-based strong encryption for the database.

### Syntax

```
public ul_ret_void ULEnableAesFipsDBEncryption(SQLCA * sqlca)
```

### Parameters

- **sqlca** A pointer to the initialized SQLCA.

### Remarks

Calling this function causes the appropriate encryption routines to be included in the application and increases the size of the application code. This was formerly called ULEnableFIPSStrongEncryption.

You can use this function in C++ API applications and embedded SQL applications. You must call this function before the Synchronize function. If you attempt to synchronize without a preceding call to enable the synchronization type, the error `SQLE_METHOD_CANNOT_BE_CALLED` occurs.

#### Separately licensed component required

ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See [“Separately licensed components” \[SQL Anywhere 12 - Introduction\]](#).

**See also**

- [“ULEnableAesDBEncryption method” on page 281](#)

## ULEnableEccE2ee method

Enables ECC end-to-end encryption.

**Syntax**

```
public ul_ret_void ULEnableEccE2ee(SQLCA * sqlca)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.

## ULEnableEccSyncEncryption method

Enables ECC encryption for SSL or TLS streams.

**Syntax**

```
public ul_ret_void ULEnableEccSyncEncryption(SQLCA * sqlca)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.

**Remarks**

This is required when set a stream parameter to TLS or HTTPS. In this case, you must also set the synchronization parameter `tls_type` as ECC.

You can use this function in C++ API applications and embedded SQL applications. You must call this function before the Synchronize function. If you attempt to synchronize without a preceding call to enable the synchronization type, the error `SQLE_METHOD_CANNOT_BE_CALLED` occurs.

**Separately licensed component required**

ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See [“Separately licensed components” \[SQL Anywhere 12 - Introduction\]](#).

**See also**

- [“ULEnableZlibSyncCompression method” on page 285](#)
- [“ULEnableRsaFipsSyncEncryption method” on page 283](#)

## ULEnableHttpSynchronization method

Enables HTTP synchronization.

### Syntax

```
public ul_ret_void ULEnableHttpSynchronization(SQLCA * sqlca)
```

### Parameters

- **sqlca** A pointer to the SQLCA.

### Remarks

You can use this function in C++ API applications and embedded SQL applications. You must call this function before the Synchronize function. If you attempt to synchronize without a preceding call to enable the synchronization type, the error `SQLE_METHOD_CANNOT_BE_CALLED` occurs.

## ULEnableRsaE2ee method

Enables RSA end-to-end encryption.

### Syntax

```
public ul_ret_void ULEnableRsaE2ee(SQLCA * sqlca)
```

### Parameters

- **sqlca** A pointer to the SQLCA.

## ULEnableRsaFipsE2ee method

Enables FIPS 140-2 certified RSA end-to-end encryption.

### Syntax

```
public ul_ret_void ULEnableRsaFipsE2ee(SQLCA * sqlca)
```

### Parameters

- **sqlca** A pointer to the SQLCA.

## ULEnableRsaFipsSyncEncryption method

Enables RSA FIPS encryption for SSL or TLS streams.

**Syntax**

```
public ul_ret_void UEnableRsaFipsSyncEncryption(SQLCA * sqlca)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.

**Remarks**

This is required when setting a stream parameter to TLS or HTTPS. In this case, you must also set the synchronization parameter `tls_type` as RSA.

You can use this function in C++ API applications and embedded SQL applications. You must call this function before the Synchronize function. If you attempt to synchronize without a preceding call to enable the synchronization type, the error `SQLE_METHOD_CANNOT_BE_CALLED` occurs.

**See also**

- [“UEnableRsaSyncEncryption method” on page 284](#)
- [“UEnableEccSyncEncryption method” on page 282](#)

## UEnableRsaSyncEncryption method

Enables RSA encryption for SSL or TLS streams.

**Syntax**

```
public ul_ret_void UEnableRsaSyncEncryption(SQLCA * sqlca)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.

**Remarks**

This is required when setting a stream parameter to TLS or HTTPS. In this case, you must also set the synchronization parameter `tls_type` as RSA.

You can use this function in C++ API applications and embedded SQL applications. You must call this function before the Synchronize function. If you attempt to synchronize without a preceding call to enable the synchronization type, the error `SQLE_METHOD_CANNOT_BE_CALLED` occurs.

**See also**

- [“UEnableEccSyncEncryption method” on page 282](#)
- [“UEnableRsaFipsSyncEncryption method” on page 283](#)

## UEnableTcpipSynchronization method

Enables TCP/IP synchronization.

**Syntax**

```
public ul_ret_void ULEnableTcpiP synchronization(SQLCA * sqlca)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.

**Remarks**

You can use this function in C++ API applications and embedded SQL applications. You must call this function before the Synchronize function. If you attempt to synchronize without a preceding call to enable the synchronization type, the error `SQLE_METHOD_CANNOT_BE_CALLED` occurs.

## ULEnableZlibSyncCompression method

Enables ZLIB compression for a synchronization stream.

**Syntax**

```
public ul_ret_void ULEnableZlibSyncCompression(SQLCA * sqlca)
```

**Parameters**

- **sqlca** A pointer to the initialized SQLCA.

**Remarks**

You can use this function in C++ API applications and embedded SQL applications. You must call this function before calling the Synchronize function. If you attempt to synchronize without a preceding call to enable the synchronization type, the error `SQLE_METHOD_CANNOT_BE_CALLED` occurs.

## ULErrorInfoInitFromSqlca method

Copies the error information from the SQLCA to the `ul_error_info` object.

**Syntax**

```
public void ULErrorInfoInitFromSqlca(  
    ul_error_info * errinf,  
    SQLCA const * sqlca  
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **errinf** The `ul_error_info` object.

## ULErrorInfoParameterAt method

Retrieves an error parameter by ordinal.

### Syntax

```
public size_t ULErrorInfoParameterAt(
    ul_error_info const * errinf,
    ul_u_short parmNo,
    char * buffer,
    size_t bufferSize
)
```

### Returns

The size (in bytes) required to store the parameter, or zero if the ordinal isn't valid. If the return value is larger than *bufferSize*, the parameter was truncated.

## ULErrorInfoParameterCount method

Retrieves the number of error parameters.

### Syntax

```
public ul_u_short ULErrorInfoParameterCount(
    ul_error_info const * errinf
)
```

### Parameters

- **errinf** The *ul\_error\_info* object.

### Returns

The number of error parameters.

## ULErrorInfoString method

Retrieves a description of the error.

### Syntax

```
public size_t ULErrorInfoString(
    ul_error_info const * errinf,
    char * buffer,
    size_t bufferSize
)
```

**Parameters**

- **errinf** The `ul_error_info` object.
- **buffer** The buffer to receive the error description.
- **bufferSize** The size, in bytes, of the buffer.

**Returns**

The size, in bytes, required to store the string. If the return value is larger than *len*, the string was truncated.

## ULErrorInfoURL method

Retrieves a URL to the documentation page for this error.

**Syntax**

```
public size_t ULErrorInfoURL(  
    ul_error_info const * errinf,  
    char * buffer,  
    size_t bufferSize,  
    char const * reserved  
)
```

**Parameters**

- **errinf** The `ul_error_info` object.
- **buffer** The buffer to receive the URL.
- **bufferSize** The size, in bytes, of the buffer.
- **reserved** Reserved for future use.

**Returns**

The size, in bytes, required to store the URL. If the return value is larger than *len*, the URL was truncated.

## ULGetDatabaseID method

Gets the current database ID.

**Syntax**

```
public ul_u_long ULGetDatabaseID(SQLCA * sqlca)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.

## Returns

The value set by the last call to SetDatabaseID or UL\_INVALID\_DATABASE\_ID if the ID was never set.

## Remarks

The current database ID used for global autoincrement.

# ULGetDatabaseProperty method

Obtains the value of a database property.

## Syntax

```
public void ULGetDatabaseProperty(  
    SQLCA * sqlca,  
    ul_database_property_id id,  
    char * dst,  
    size_t buffer_size,  
    ul_bool * null_indicator  
)
```

## Parameters

- **sqlca** A pointer to the SQLCA.
- **id** The identifier for the database property.
- **dst** A character array to store the value of the property.
- **buffer\_size** The size of the character array dst.
- **null\_indicator** An indicator that the database parameter is null.

# ULGetDatabasePropertyW method

Obtains the value of a database property.

## Syntax

```
public void ULGetDatabasePropertyW(  
    SQLCA * sqlca,  
    ul_database_property_id id,  
    ul_wchar * dst,  
    size_t buffer_size,  
    ul_bool * null_indicator  
)
```

## Parameters

- **sqlca** A pointer to the SQLCA.



- **id** The identifier for the database property.
- **dst** A character array to store the value of the property.
- **buffer\_size** The size of the character array *dst*.
- **null\_indicator** An indicator that the database parameter is null.

#### Remarks

**Note**

This method prototype is used internally when you refer to `ULGetDatabaseProperty` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

## ULGetErrorParameter method

Retrieve error parameter via ordinal.

#### Syntax

```
public size_t ULGetErrorParameter(
    SQLCA const * sqlca,
    ul_u_long parm_num,
    char * buffer,
    size_t size
)
```

#### Parameters

- **sqlca** A pointer to the SQLCA
- **parm\_num** The ordinal parameter number.
- **buffer** Pointer to a buffer to contain the error parameter.
- **size** The size in bytes of the buffer.

#### Returns

This function returns the number of characters copied to the supplied buffer.

#### See also

- [“ULGetErrorParameterCount method” on page 289](#)

## ULGetErrorParameterCount method

Obtain count of the number of error parameters.

**Syntax**

```
public ul_u_long ULGetErrorParameterCount(SQLCA const * sqlca)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.

**Returns**

This function returns the number of error parameters. Unless the result is zero, values from 1 through this result can be used to call `ULGetErrorParameter` to retrieve the corresponding error parameter value.

**See also**

- [“ULGetErrorParameter method” on page 289](#)

## ULGetErrorParameterW method

Retrieve error parameter via ordinal.

**Syntax**

```
public size_t ULGetErrorParameterW(  
    SQLCA const * sqlca,  
    ul_u_long parm_num,  
    ul_wchar * buffer,  
    size_t size  
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA
- **parm\_num** The ordinal parameter number.
- **buffer** Pointer to a buffer to contain the error parameter.
- **size** The size in bytes of the buffer.

**Returns**

This function returns the number of characters copied to the supplied buffer.

**Remarks****Note**

This method prototype is used internally when you refer to `ULGetErrorParameter` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

**See also**

- [“ULGetErrorParameterCount method” on page 289](#)

## ULGetIdentity method

Gets the @identity value.

**Syntax**

```
public ul_u_big ULGetIdentity(SQLCA * sqlca)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.

**Returns**

The last value inserted into an autoincrement or global autoincrement column.

## ULGetLastDownloadTime method

Obtains the last time a specified publication was downloaded.

**Syntax**

```
public ul_bool ULGetLastDownloadTime(  
    SQLCA * sqlca,  
    char const * pub_name,  
    DECL_DATETIME * value  
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **pub\_name** A string containing a publication name for which the last download time is retrieved.
- **value** A pointer to the DECL\_DATETIME structure to be populated. For example, value of January 1, 1990 indicates that the publication has yet to be synchronized.

**Returns**

True when the value is successfully populated by the last download time of the publication specified by pub-name. Otherwise, it returns false.

**Remarks**

The following call populates the dt structure with the date and time that publication UL\_PUB\_PUB1 was downloaded:

```
DECL_DATETIME dt;  
ret = ULGetLastDownloadTime( &sqlca, UL_TEXT("UL_PUB_PUB1"), &dt );
```

## ULGetLastDownloadTimeW method

Obtains the last time a specified publication was downloaded.

### Syntax

```
public ul_bool ULGetLastDownloadTimeW(  
    SQLCA * sqlca,  
    ul_wchar const * pub_name,  
    DECL_DATETIME * value  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **pub\_name** A string containing a publication name for which the last download time is retrieved.
- **value** A pointer to the DECL\_DATETIME structure to be populated. For example, value of January 1, 1990 indicates that the publication has yet to be synchronized.

### Returns

True when the value is successfully populated by the last download time of the publication specified by pub-name. Otherwise, it returns false.

### Remarks

#### Note

This method prototype is used internally when you refer to ULGetLastDownloadTime and #define the UNICODE macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

The following call populates the dt structure with the date and time that publication UL\_PUB\_PUB1 was downloaded:

```
DECL_DATETIME dt;  
ret = ULGetLastDownloadTime( &sqlca, UL_TEXT("UL_PUB_PUB1"), &dt );
```

## ULGetNotification method

Reads an event notification.

### Syntax

```
public ul_bool ULGetNotification(  
    SQLCA * sqlca,
```

```
    char const * queue_name,  
    char * event_name_buf,  
    ul_length event_name_buf_len,  
    ul_u_long wait_ms  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **queue\_name** The queue to read or NULL for default connection queue.
- **event\_name\_buf** A buffer to hold the name of the event.
- **event\_name\_buf\_len** The size of the buffer in bytes.
- **wait\_ms** The time to wait (block) before returning.

### Returns

True on success, false on failure.

### Remarks

This call blocks until a notification is received or until the given wait period expires. To wait indefinitely, pass `UL_READ_WAIT_INFINITE` for `wait_ms`. To cancel a wait, send another notification to the given queue or use `CancelGetNotification()`. After reading a notification, use `GetNotificationParameter()` to retrieve additional parameters by name.

## ULGetNotificationParameter method

Gets a parameter for the event notification just read by `GetNotification()`.

### Syntax

```
public ul_bool ULGetNotificationParameter(  
    SQLCA * sqlca,  
    char const * queue_name,  
    char const * parameter_name,  
    char * value_buf,  
    ul_length value_buf_len  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **queue\_name** The queue to read or NULL for default connection queue.
- **parameter\_name** The name of the parameter to read (or "\*").
- **value\_buf** A buffer to hold the parameter value.

- **value\_buf\_len** The size of the buffer in bytes.

### Returns

True on success, false on failure.

### Remarks

Only the parameters from the most-recently read notification on the given queue are available. Parameters are retrieved by name. A parameter name of "\*" retrieves the entire parameter string.

## ULGetNotificationParameterW method

Gets a parameter for the event notification just read by GetNotification().

### Syntax

```
public ul_bool ULGetNotificationParameterW(  
    SQLCA * sqlca,  
    ul_wchar const * queue_name,  
    ul_wchar const * parameter_name,  
    ul_wchar * value_buf,  
    ul_length value_buf_len  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **queue\_name** The queue to read or NULL for default connection queue.
- **parameter\_name** The name of the parameter to read (or "\*").
- **value\_buf** A buffer to hold the parameter value.
- **value\_buf\_len** The size of the buffer in ul\_wchars.

### Returns

True on success, false on failure.

### Remarks

**Note**

This method prototype is used internally when you refer to ULGetNotificationParameter and #define the UNICODE macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

Only the parameters from the most-recently read notification on the given queue are available. Parameters are retrieved by name. A parameter name of "\*" retrieves the entire parameter string.

## ULGetNotificationW method

Reads an event notification.

### Syntax

```
public ul_bool ULGetNotificationW(  
    SQLCA * sqlca,  
    ul_wchar const * queue_name,  
    ul_wchar * event_name_buf,  
    ul_length event_name_buf_len,  
    ul_u_long wait_ms  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **queue\_name** The queue to read or NULL for default connection queue.
- **event\_name\_buf** A buffer to hold the name of the event.
- **event\_name\_buf\_len** The size of the buffer in ul\_wchars.
- **wait\_ms** The time to wait (block) before returning.

### Returns

True on success, false on failure.

### Remarks

**Note**

This method prototype is used internally when you refer to ULGetNotification and #define the UNICODE macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

This call blocks until a notification is received or until the given wait period expires. To wait indefinitely, pass UL\_READ\_WAIT\_INFINITE for wait\_ms. To cancel a wait, send another notification to the given queue or use CancelGetNotification(). After reading a notification, use GetNotificationParameter() to retrieve additional parameters by name.

## ULGetSyncResult method

Gets the result of the last synchronization.

### Syntax

```
public ul_bool ULGetSyncResult(  
    SQLCA * sqlca,
```

```
    ul_sync_result * sync_result  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **sync\_result** A pointer to the ul\_sync\_result structure that holds the synchronization results.

### Returns

True on success, otherwise false.

### Remarks

For more information about the ul\_sync\_result structure, see [“ul\\_sync\\_result structure” on page 86](#).

## ULGlobalAutoincUsage method

Obtains the percent of the default values used in all the columns having global autoincrement defaults.

### Syntax

```
public ul_u_short ULGlobalAutoincUsage(SQLCA * sqlca)
```

### Parameters

- **sqlca** A pointer to the SQLCA.

### Returns

A short in the range 0-100.

### Remarks

If the database contains more than one column with this default, this value is calculated for all columns and the maximum is returned. For example, a return value of 99 indicates that very few default values remain for at least one of the columns.

### See also

- [“ULSetDatabaseID method” on page 306](#)

## ULGrantConnectTo method

Grants access to an UltraLite database for a new or existing user ID with the given password.

### Syntax

```
public ul_ret_void ULGrantConnectTo(  
    SQLCA * sqlca,  
    char const * uid,
```



```
        char const * pwd
    )
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **uid** A character array that holds the user ID. The maximum length is 16 characters.
- **pwd** A character array that holds the password for the user ID. The maximum length is 16 characters.

**Remarks**

If you specify an existing user ID, this function then updates the password for the user.

**See also**

- [“ULRevokeConnectFrom method” on page 303](#)

## ULGrantConnectToW method

Grants access to an UltraLite database for a new or existing user ID with the given password.

**Syntax**

```
public ul_ret_void ULGrantConnectToW(
    SQLCA * sqlca,
    ul_wchar const * uid,
    ul_wchar const * pwd
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **uid** A character array that holds the user ID. The maximum length is 16 characters.
- **pwd** A character array that holds the password for the user ID. The maximum length is 16 characters.

**Remarks****Note**

This method prototype is used internally when you refer to `ULGrantConnectTo` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

If you specify an existing user ID, this function then updates the password for the user.

**See also**

- [“ULRevokeConnectFrom method” on page 303](#)

## ULInitSyncInfo method

Initializes the synchronization information structure.

### Syntax

```
public ul_ret_void ULInitSyncInfo(ul_sync_info * info)
```

### Parameters

- **info** A synchronization structure.

## ULInitSyncInfoW method

Initializes the synchronization information structure.

### Syntax

```
public ul_ret_void ULInitSyncInfoW(ul_sync_info_w2 * info)
```

### Parameters

- **info** A synchronization structure.

### Remarks

**Note**

This method prototype is used internally when you refer to `ULInitSyncInfo` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

## ULIsSynchronizeMessage method

Checks a message to see if it is a synchronization message from the MobiLink provider for ActiveSync, so that code to handle such a message can be called.

### Syntax

```
public ul_bool ULIsSynchronizeMessage(ul_u_long number)
```

### Remarks

When the processing of a synchronization message is complete, the `ULSignalSyncIsComplete` function should be called.

You should include a call to this function in the `WindowProc` function of your application. This applies to Windows Mobile for ActiveSync.

The following code snippet illustrates how to use `ULIsSynchronizeMessage` to handle a synchronization message:

```
LRESULT CALLBACK WindowProc( HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam )
{
    if( ULIsSynchronizeMessage( uMsg ) ) {
        // execute synchronization code
        if( wParam == 1 ) DestroyWindow( hwnd );
        return 0;
    }

    switch( uMsg ) {

        // code to handle other windows messages

        default:
            return DefWindowProc( hwnd, uMsg, wParam, lParam );
    }
    return 0;
}
```

**See also**

- [“ULSignalSyncIsComplete method” on page 311](#)

## ULLibraryVersion method

Returns the version number of the UltraLite runtime library.

**Syntax**

```
public char const * ULLibraryVersion(ul_arg_void )
```

**Returns**

The version number of the UltraLite runtime library.

## ULRSALibraryVersion method

Returns the version number of the RSA encryption library.

**Syntax**

```
public char const * ULRSALibraryVersion(ul_arg_void )
```

**Returns**

The version number of the RSA encryption library.

## ULRegisterForEvent method

Registers or unregisters a queue to receive notifications of an event.

### Syntax

```
public ul_bool ULRegisterForEvent(  
    SQLCA * sqlca,  
    char const * event_name,  
    char const * object_name,  
    char const * queue_name,  
    ul_bool register_not_unreg  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **event\_name** The system or user-defined event to register for.
- **object\_name** The object to which the event applies (like table name).
- **queue\_name** NULL means default connection queue.
- **register\_not\_unreg** True to register, false to unregister.

### Returns

True if the registration succeeded, false if the queue or event does not exist.

### Remarks

If no queue name is supplied, the default connection queue is implied, and created if required. Certain system events allow specification of an object name to which the event applies. For example, the TableModified event can specify the table name. Unlike SendNotification(), only the specific queue registered receives notifications of the event. Other queues with the same name on different connections do not, unless they are also explicit registered.

The predefined system events are:

TableModified - Triggered when rows in a table are inserted, updated, or deleted. One notification is sent per request, no matter how many rows were affected by the request. The *object\_name* parameter specifies the table to monitor. A value of "\*" means all tables in the database. This event has a parameter named 'table\_name' whose value is the name of the modified table.

Commit - Triggered after any commit completes. This event has no parameters.

SyncComplete - Triggered after synchronization completes. This event has no parameters.

## ULRegisterForEventW method

Registers or unregisters a queue to receive notifications of an event.

### Syntax

```
public ul_bool ULRegisterForEventW(
    SQLCA * sqlca,
    ul_wchar const * event_name,
    ul_wchar const * object_name,
    ul_wchar const * queue_name,
    ul_bool register_not_unreg
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **event\_name** The system or user-defined event to register for.
- **object\_name** The object to which the event applies (like table name).
- **queue\_name** NULL means default connection queue.
- **register\_not\_unreg** True to register, false to unregister.

### Returns

True if the registration succeeded, false if the queue or event does not exist.

### Remarks

#### Note

This method prototype is used internally when you refer to `ULRegisterForEvent` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

If no queue name is supplied, the default connection queue is implied, and created if required. Certain system events allow specification of an object name to which the event applies. For example, the `TableModified` event can specify the table name. Unlike `SendNotification()`, only the specific queue registered receives notifications of the event. Other queues with the same name on different connections do not, unless they are also explicit registered.

The predefined system events are:

`TableModified` - Triggered when rows in a table are inserted, updated, or deleted. One notification is sent per request, no matter how many rows were affected by the request. The *object\_name* parameter specifies the table to monitor. A value of "\*" means all tables in the database. This event has a parameter named 'table\_name' whose value is the name of the modified table.

`Commit` - Triggered after any commit completes. This event has no parameters.

`SyncComplete` - Triggered after synchronization completes. This event has no parameters.

## ULResetLastDownloadTime method

Resets the last download time of a publication so that the application resynchronizes previously downloaded data.

### Syntax

```
public ul_ret_void ULResetLastDownloadTime(  
    SQLCA * sqlca,  
    char const * pub_list  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA
- **pub\_list** A string containing a comma-separated list of publications to reset. An empty string implies all tables except tables marked as "no sync". A string containing just an asterisk ("\*") implies all publications. Some tables may not be part of any publication and are not included if the pub-list string is "\*".

### Remarks

The following function call resets the last download time for all tables:

```
ULResetLastDownloadTime( &sqlca, UL_TEXT("") );
```

## ULResetLastDownloadTimeW method

Resets the last download time of a publication so that the application resynchronizes previously downloaded data.

### Syntax

```
public ul_ret_void ULResetLastDownloadTimeW(  
    SQLCA * sqlca,  
    ul_wchar const * pub_list  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA
- **pub\_list** A string containing a comma-separated list of publications to reset. An empty string implies all tables except tables marked as "no sync". A string containing just an asterisk ("\*") implies all publications. Some tables may not be part of any publication and are not included if the pub-list string is "\*".

## Remarks

**Note**

This method prototype is used internally when you refer to `ULResetLastDownloadTime` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

The following function call resets the last download time for all tables:

```
ULResetLastDownloadTime( &sqlca, UL_TEXT("") );
```

## ULRevokeConnectFrom method

Revokes access from an UltraLite database for a user ID.

### Syntax

```
public ul_ret_void ULRevokeConnectFrom(SQLCA * sqlca, char const * uid)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **uid** A character array holding the user ID to be excluded from database access. The maximum length is 16 characters.

## ULRevokeConnectFromW method

Revokes access from an UltraLite database for a user ID.

### Syntax

```
public ul_ret_void ULRevokeConnectFromW(  
    SQLCA * sqlca,  
    ul_wchar const * uid  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **uid** A character array holding the user ID to be excluded from database access. The maximum length is 16 characters.

## Remarks

**Note**

This method prototype is used internally when you refer to `ULRevokeConnectFrom` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

## ULRollbackPartialDownload method

Rolls back the changes from a failed synchronization.

### Syntax

```
public ul_ret_void ULRollbackPartialDownload(SQLCA * sqlca)
```

### Parameters

- **sqlca** A pointer to the SQLCA.

### Remarks

When a communication error occurs during the download phase of synchronization, UltraLite can apply the downloaded changes, so that the application can resume the synchronization from the place it was interrupted. If the download changes are not needed (the user or application does not want to resume the download at this point), `ULRollbackPartialDownload` rolls back the failed download transaction.

## ULSendNotification method

Sends a notification to all queues matching the given name.

### Syntax

```
public ul_u_long ULSendNotification(  
    SQLCA * sqlca,  
    char const * queue_name,  
    char const * event_name,  
    char const * parameters  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **queue\_name** NULL means default connection queue.
- **event\_name** The system or user-defined event to register for.
- **parameters** Creation parameters; currently unused, set to NULL.



## Returns

The number of notifications sent (the number of matching queues).

## Remarks

This includes any such queue on the current connection. This call does not block. Use the special queue name "\*" to send to all queues. The given event name does not need to correspond to any system or user-defined event; it is simply passed through to identify the notification when read and has meaning only to the sender and receiver. The *parameters* argument specifies a semi-colon delimited name=value pairs option list. After the notification is read, the parameter values are read with GetNotificationParameter().

# ULSendNotificationW method

Sends a notification to all queues matching the given name.

## Syntax

```
public ul_u_long ULSendNotificationW(  
    SQLCA * sqlca,  
    ul_wchar const * queue_name,  
    ul_wchar const * event_name,  
    ul_wchar const * parameters  
)
```

## Parameters

- **sqlca** A pointer to the SQLCA.
- **queue\_name** NULL means default connection queue.
- **event\_name** The system or user-defined event to register for.
- **parameters** Creation parameters; currently unused, set to NULL.

## Returns

The number of notifications sent (the number of matching queues).

## Remarks

### Note

This method prototype is used internally when you refer to ULSendNotification and #define the UNICODE macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

This includes any such queue on the current connection. This call does not block. Use the special queue name "\*" to send to all queues. The given event name does not need to correspond to any system or user-defined event; it is simply passed through to identify the notification when read and has meaning only to the sender and receiver. The *parameters* argument specifies a semi-colon delimited name=value pairs option list. After the notification is read, the parameter values are read with GetNotificationParameter().

## ULSetDatabaseID method

Sets the database identification number.

### Syntax

```
public ul_ret_void ULSetDatabaseID(SQLCA * sqlca, ul_u_long value)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **value** A positive integer that uniquely identifies a particular database in a replication or synchronization setup.

### See also

- [“ULGlobalAutoincUsage method” on page 296](#)

## ULSetDatabaseOptionString method

Sets a database option from a string value.

### Syntax

```
public void ULSetDatabaseOptionString(  
    SQLCA * sqlca,  
    ul_database_option_id id,  
    char const * value  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **id** The identifier for the database option to be set.
- **value** The value of the database option.

## ULSetDatabaseOptionStringW method

Sets a database option from a string value.

### Syntax

```
public void ULSetDatabaseOptionStringW(  
    SQLCA * sqlca,  
    ul_database_option_id id,  
    ul_wchar const * value  
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **id** The identifier for the database option to be set.
- **value** The value of the database option.

**Remarks****Note**

This method prototype is used internally when you refer to `ULSetDatabaseOptionString` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

## ULSetDatabaseOptionULong method

Sets a numeric database option.

**Syntax**

```
public void ULSetDatabaseOptionULong(
    SQLCA * sqlca,
    ul_database_option_id id,
    ul_u_long value
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **id** The identifier for the database option to be set.
- **value** The value of the database option.

## ULSetErrorCallback method

Sets the callback to be invoked when an error occurs.

**Syntax**

```
public ul_ret_void ULSetErrorCallback(
    SQLCA * sqlca,
    ul_error_callback_fn_a callback,
    ul_void * user_data,
    char * buffer,
    size_t len
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **callback** The callback function
- **user\_data** User context information passed to the callback.
- **buffer** A user-supplied buffer that contains the error parameters when the callback is invoked.
- **len** The size of the buffer in bytes.

## ULSetErrorCallbackW method

Sets the callback to be invoked when an error occurs.

**Syntax**

```
public ul_ret_void ULSetErrorCallbackW(  
    SQLCA * sqlca,  
    ul_error_callback_fn_w2 callback,  
    ul_void * user_data,  
    ul_wchar * buffer,  
    size_t len  
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **callback** The callback function
- **user\_data** User context information passed to the callback.
- **buffer** A user-supplied buffer that contains the error parameters when the callback is invoked.
- **len** The size of the buffer in bytes.

**Remarks****Note**

This method prototype is used internally when you refer to `ULSetErrorCallback` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

## ULSetSyncInfo method

Creates a synchronization profile using the given name based on the given `ul_sync_info` structure.

**Syntax**

```
public ul_bool ULSetSyncInfo(
    SQLCA * sqlca,
    char const * profile_name,
    ul_sync_info * sync_info
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **profile\_name** The name of the synchronization profile.
- **sync\_info** A pointer to the ul\_sync\_info structure that holds the synchronization parameters.

**Returns**

True on success, otherwise false.

**Remarks**

This sync profile replaces any previous sync profile with the same name. Specifying a null pointer for the ul\_sync\_info deletes the named profile.

## ULSetSyncInfoW method

Creates a synchronization profile using the given name based on the given ul\_sync\_info structure.

**Syntax**

```
public ul_bool ULSetSyncInfoW(
    SQLCA * sqlca,
    ul_wchar const * profile_name,
    ul_sync_info_w2 * sync_info
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **profile\_name** The name of the synchronization profile.
- **sync\_info** A pointer to the ul\_sync\_info structure that holds the synchronization parameters.

**Returns**

True on success, otherwise false.

## Remarks

**Note**

This method prototype is used internally when you refer to `ULSetSyncInfo` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

This sync profile replaces any previous sync profile with the same name. Specifying a null pointer for the `ul_sync_info` deletes the named profile.

## ULSetSynchronizationCallback method

Sets the callback to be invoked while performing a synchronization.

### Syntax

```
public ul_ret_void ULSetSynchronizationCallback(
    SQLCA * sqlca,
    ul_sync_observer_fn callback,
    ul_void * user_data
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **callback** The callback.
- **user\_data** User context information passed to the callback.

## ULSetSynchronizationCallbackEx method

Sets the callback to be invoked while performing a synchronization.

### Syntax

```
public ul_ret_void ULSetSynchronizationCallbackEx(
    SQLCA * sqlca,
    ul_sync_observer_fn callback,
    ul_void * user_data
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **callback** The callback.
- **user\_data** User context information passed to the callback.

## ULSignalSyncIsComplete method

This function is called to indicate that processing a synchronization message is complete.

### Syntax

```
public ul_ret_void ULSignalSyncIsComplete()
```

### Remarks

Applications which are registered with the ActiveSync provider need to call this method in their WNDPROC when processing a synchronization message is complete.

## ULStartSynchronizationDelete method

Sets START SYNCHRONIZATION DELETE for this connection.

### Syntax

```
public ul_ret_void ULStartSynchronizationDelete(SQLCA * sqlca)
```

### Parameters

- **sqlca** A pointer to the SQLCA.

### Returns

True on success, otherwise false.

## ULStaticFini method

Performs finalization of the UltraLite runtime for embedded SQL applications.

### Syntax

```
public void ULStaticFini()
```

### Remarks

This function should be called once and only once per application, after which no other UltraLite functions should be called.

## ULStaticInit method

Performs initialization of the UltraLite runtime for embedded SQL applications.

### Syntax

```
public void ULStaticInit()
```

### Remarks

This function should be called once and only once per application, before any other UltraLite functions have been called.

## ULStopSynchronizationDelete method

Sets STOP SYNCHRONIZATION DELETE for this connection.

### Syntax

```
public ul_bool ULStopSynchronizationDelete(SQLCA * sqlca)
```

### Parameters

- **sqlca** A pointer to the SQLCA

### Returns

True on success, otherwise false.

## ULSynchronize method

Initiates synchronization in an UltraLite application.

### Syntax

```
public ul_ret_void ULSynchronize(SQLCA * sqlca, ul_sync_info * info)
```

### Parameters

- **sqlca** A pointer to the SQLCA
- **info** A synchronization structure.

### Remarks

For TCP/IP or HTTP synchronization, the ULSynchronize function initiates synchronization. Errors during synchronization that are not handled by the handle\_error script are reported as SQL errors. Application programs should test the SQLCODE return value of this function.

## ULSynchronizeFromProfile method

Synchronize the database using the given profile and merge parameters.



**Syntax**

```
public ul_ret_void ULSynchronizeFromProfile(  
    SQLCA * sqlca,  
    char const * profile_name,  
    char const * merge_parms,  
    ul_sync_observer_fn observer,  
    ul_void * user_data  
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **profile\_name** Name of the profile to synchronize.
- **merge\_parms** Merge parameters for the synchronization.
- **observer** Observer callback to send status updates to.
- **user\_data** User context data passed to callback.

## ULSynchronizeFromProfileW method

Synchronize the database using the given profile and merge parameters.

**Syntax**

```
public ul_ret_void ULSynchronizeFromProfileW(  
    SQLCA * sqlca,  
    ul_wchar const * profile_name,  
    ul_wchar const * merge_parms,  
    ul_sync_observer_fn observer,  
    ul_void * user_data  
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **profile\_name** Name of the profile to synchronize.
- **merge\_parms** Merge parameters for the synchronization.
- **observer** Observer callback to send status updates to.
- **user\_data** User context data passed to callback.

**Remarks****Note**

This method prototype is used internally when you refer to `ULSynchronizeFromProfile` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

## ULSynchronizeW method

Initiates synchronization in an UltraLite application.

**Syntax**

```
public ul_ret_void ULSynchronizeW(SQLCA * sqlca, ul_sync_info_w2 * info)
```

**Parameters**

- **sqlca** A pointer to the SQLCA
- **info** A synchronization structure.

**Remarks****Note**

This method prototype is used internally when you refer to `ULSynchronize` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

For TCP/IP or HTTP synchronization, the `ULSynchronize` function initiates synchronization. Errors during synchronization that are not handled by the `handle_error` script are reported as SQL errors. Application programs should test the `SQLCODE` return value of this function.

## ULTriggerEvent method

Trigger a user-defined event (and send notification to all registered queues).

**Syntax**

```
public ul_u_long ULTriggerEvent(  
    SQLCA * sqlca,  
    char const * event_name,  
    char const * parameters  
)
```

**Parameters**

- **sqlca** A pointer for the SQLCA.

- **event\_name** The system or user-defined event to register for.
- **parameters** Creation parameters; currently unused, set to NULL.

### Returns

The number of event notifications sent.

### Remarks

The *parameters* value specifies a semi-colon delimited name=value pairs option list. After the notification is read, the parameter values are read with GetNotificationParameter().

## ULTriggerEventW method

Trigger a user-defined event (and send notification to all registered queues).

### Syntax

```
public ul_u_long ULTriggerEventW(
    SQLCA * sqlca,
    ul_wchar const * event_name,
    ul_wchar const * parameters
)
```

### Parameters

- **sqlca** A pointer for the SQLCA.
- **event\_name** The system or user-defined event to register for.
- **parameters** Creation parameters; currently unused, set to NULL.

### Returns

The number of event notifications sent.

### Remarks

#### Note

This method prototype is used internally when you refer to ULTriggerEvent and #define the UNICODE macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

The *parameters* value specifies a semi-colon delimited name=value pairs option list. After the notification is read, the parameter values are read with GetNotificationParameter().

## ULTruncateTable method

Truncates the table and temporarily activates STOP SYNCHRONIZATION DELETE.

**Syntax**

```
public ul_ret_void ULTruncateTable(SQLCA * sqlca, ul_table_num number)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **number** The id of the table to truncate.

**Returns**

True on success, otherwise false.

## ULValidateDatabase method

Validates the database on this connection.

**Syntax**

```
public ul_bool ULValidateDatabase(  
    SQLCA * sqlca,  
    char const * start_parms,  
    ul_table_num table_id,  
    ul_u_short flags,  
    ul_validate_callback_fn callback_fn,  
    void * user_data  
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **start\_parms** The parameter used to start the database.
- **table\_id** The id of a specific table to validate.
- **flags** Flags controlling the type of validation.
- **callback\_fn** The function to receive validation progress information.
- **user\_data** User data to send back to the caller via the callback.

**Returns**

True on success, otherwise false.

**Remarks**

Depending on the flags passed to this routine, the low level store and/or the indexes can be validated. To receive information during the validation, implement a callback function and pass the address to this routine. To limit the validation to a specific table, pass in the table name or ID as the last parameter.

The flags parameter is combination of the following values:

- **#ULVF\_TABLE** For more information, see “[ULVF\\_TABLE variable](#)” on page 90.
- **#ULVF\_INDEX** For more information, see “[ULVF\\_INDEX variable](#)” on page 90.
- **#ULVF\_DATABASE** For more information, see “[ULVF\\_DATABASE variable](#)” on page 89.
- **#ULVF\_EXPRESS** For more information, see “[ULVF\\_EXPRESS variable](#)” on page 89.
- **#ULVF\_FULL\_VALIDATE** For more information, see “[ULVF\\_FULL\\_VALIDATE variable](#)” on page 89.

## ULValidateDatabaseTableName method

Validates the database on this connection.

### Syntax

```
public ul_bool ULValidateDatabaseTableName(
    SQLCA * sqlca,
    char const * start_parms,
    char const * table_name,
    ul_u_short flags,
    ul_validate_callback_fn callback_fn,
    void * user_data
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **start\_parms** The parameter used to start the database.
- **table\_name** The name of a specific table to validate.
- **flags** Flags controlling the type of validation.
- **callback\_fn** The function to receive validation progress information.
- **user\_data** User data to send back to the caller via the callback.

### Returns

True on success, otherwise false.

### Remarks

Depending on the flags passed to this routine, the low level store and/or the indexes can be validated. To receive information during the validation, implement a callback function and pass the address to this routine. To limit the validation to a specific table, pass in the table name or ID as the last parameter.

The flags parameter is combination of the following values:

- **#ULVF\_TABLE** For more information, see “[ULVF\\_TABLE variable](#)” on page 90.
- **#ULVF\_INDEX** For more information, see “[ULVF\\_INDEX variable](#)” on page 90.
- **#ULVF\_DATABASE** For more information, see “[ULVF\\_DATABASE variable](#)” on page 89.
- **#ULVF\_EXPRESS** For more information, see “[ULVF\\_EXPRESS variable](#)” on page 89.
- **#ULVF\_FULL\_VALIDATE** For more information, see “[ULVF\\_FULL\\_VALIDATE variable](#)” on page 89.

## ULValidateDatabaseTableNameW method

Validates the database on this connection.

### Syntax

```
public ul_bool ULValidateDatabaseTableNameW(  
    SQLCA * sqlca,  
    ul_wchar const * start_parms,  
    ul_wchar const * table_name,  
    ul_u_short flags,  
    ul_validate_callback_fn callback_fn,  
    void * user_data  
)
```

### Parameters

- **sqlca** A pointer to the SQLCA.
- **start\_parms** The parameter used to start the database.
- **table\_name** The name of a specific table to validate.
- **flags** Flags controlling the type of validation.
- **callback\_fn** The function to receive validation progress information.
- **user\_data** User data to send back to the caller via the callback.

### Returns

True on success, otherwise false.

### Remarks

Depending on the flags passed to this routine, the low level store and/or the indexes can be validated. To receive information during the validation, implement a callback function and pass the address to this routine. To limit the validation to a specific table, pass in the table name or ID as the last parameter.

The flags parameter is combination of the following values:

- **#ULVF\_TABLE** For more information, see [“ULVF\\_TABLE variable” on page 90](#).
- **#ULVF\_INDEX** For more information, see [“ULVF\\_INDEX variable” on page 90](#).
- **#ULVF\_DATABASE** For more information, see [“ULVF\\_DATABASE variable” on page 89](#).
- **#ULVF\_EXPRESS** For more information, see [“ULVF\\_EXPRESS variable” on page 89](#).
- **#ULVF\_FULL\_VALIDATE** For more information, see [“ULVF\\_FULL\\_VALIDATE variable” on page 89](#).

**Note**

This method prototype is used internally when you refer to `ULValidateDatabaseTableName` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

## ULValidateDatabaseW method

Validates the database on this connection.

**Syntax**

```
public ul_bool ULValidateDatabaseW(
    SQLCA * sqlca,
    ul_wchar const * start_parms,
    ul_table_num table_id,
    ul_u_short flags,
    ul_validate_callback_fn callback_fn,
    void * user_data
)
```

**Parameters**

- **sqlca** A pointer to the SQLCA.
- **start\_parms** The parameter used to start the database.
- **table\_id** The id of a specific table to validate.
- **flags** Flags controlling the type of validation.
- **callback\_fn** The function to receive validation progress information.
- **user\_data** User data to send back to the caller via the callback.

**Returns**

True on success, otherwise false.

**Remarks****Note**

This method prototype is used internally when you refer to `ULValidateDatabase` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

Depending on the flags passed to this routine, the low level store and/or the indexes can be validated. To receive information during the validation, implement a callback function and pass the address to this routine. To limit the validation to a specific table, pass in the table name or ID as the last parameter.

The flags parameter is combination of the following values:

- **#ULVF\_TABLE** For more information, see [“ULVF\\_TABLE variable” on page 90](#).
- **#ULVF\_INDEX** For more information, see [“ULVF\\_INDEX variable” on page 90](#).
- **#ULVF\_DATABASE** For more information, see [“ULVF\\_DATABASE variable” on page 89](#).
- **#ULVF\_EXPRESS** For more information, see [“ULVF\\_EXPRESS variable” on page 89](#).
- **#ULVF\_FULL\_VALIDATE** For more information, see [“ULVF\\_FULL\\_VALIDATE variable” on page 89](#).

## ml\_file\_transfer\_info structure

A structure containing the parameters to the file upload/download.

**Syntax**

```
public struct ml_file_transfer_info
```

**Members**

Member name	Type	Description
auth_parms	char **	Supplies parameters to authentication parameters in MobiLink events.  For more information, see <a href="#">“Additional Parameters synchronization parameter” [UltraLite - Database Management and Reference]</a> .



Member name	Type	Description
auth_status	asa_uint16	Supplies parameters to authentication parameters in MobiLink events.  For more information, see <a href="#">“Additional Parameters synchronization parameter”</a> [ <i>UltraLite - Database Management and Reference</i> ].
auth_value	asa_uint16	Reports results of a custom MobiLink user authentication script. The MobiLink server provides this information to the client.  For more information, see <a href="#">“Authentication Value synchronization parameter”</a> [ <i>UltraLite - Database Management and Reference</i> ].
enable_resume	bool	If set to true, MLFileDownload resumes a previous download that was interrupted because of a communications error or because it was canceled by the user. If the file on the server is newer than the partial local file, the partial file is discarded and the new version is downloaded from the beginning. The default is true.
error	mlft_stream_error	Contains information about any error that occurs.
file_auth_code	asa_uint16	Contains the return code of the optional authenticate_file_transfer script on the server.
filename	char *	The file name to be transferred from the server running MobiLink. MobiLink searches the username subdirectory first, before defaulting to the root directory.  For more information, see <a href="#">“-ftr mlsrv12 option”</a> [ <i>MobiLink - Server Administration</i> ].
local_filename	char *	The local name for the downloaded file. If this parameter is empty, the value in file name is used.
local_path	char *	The local path to store the downloaded file. If this parameter is empty (the default), the downloaded file is stored in the current directory.  On Windows Mobile, if dest_path is empty, the file is stored in the root (\) directory of the device.  On the desktop, if the dest_path is empty, the file is stored in the user's current directory.

Member name	Type	Description
num_auth_parms	asa_uint8	The number of authentication parameters being passed to authentication parameters in MobiLink events.  For more information, see <a href="#">“Number of Authentication Parameters parameter”</a> [ <i>UltraLite - Database Management and Reference</i> ].
observer	ml_file_transfer_observer_fn	A callback can be provided to observe file download progress through the 'observer' field. For more details, see description of Callback Function that follows.
password	char *	The password for the MobiLink user name.
remote_key	char *	The MobiLink remote key.
stream	const char *	Required. The protocol can be one of: TCPIP, TLS, HTTP, or HTTPS.  For more information, see <a href="#">“Stream Type synchronization parameter”</a> [ <i>UltraLite - Database Management and Reference</i> ].
stream_parms	char *	The protocol options for a given stream.  For more information, see <a href="#">“Network protocol options for UltraLite synchronization streams”</a> [ <i>UltraLite - Database Management and Reference</i> ].
transferred_file	asa_uint16	1 if the file was successfully transferred and 0 if an error occurs. An error occurs if the file is already up-to-date when MLFileUpload is invoked. In this case, the function returns true rather than false.
user_data	void *	The application-specific information made available to the synchronization observer.  For more information, see <a href="#">“User Data synchronization parameter”</a> [ <i>UltraLite - Database Management and Reference</i> ].
username	char *	Required. MobiLink user name.
version	char *	Required. The MobiLink script version.

## ml\_file\_transfer\_info\_w structure

A structure containing the parameters to the file upload/download.

**Syntax**

```
public struct ml_file_transfer_info_w
```

**Members**

Member name	Type	Description
auth_parms	wchar_t **	Supplies parameters to authentication parameters in MobiLink events.  For more information, see <a href="#">“Additional Parameters synchronization parameter”</a> [ <i>UltraLite - Database Management and Reference</i> ].
auth_status	asa_uint16	Supplies parameters to authentication parameters in MobiLink events.  For more information, see <a href="#">“Additional Parameters synchronization parameter”</a> [ <i>UltraLite - Database Management and Reference</i> ].
auth_value	asa_uint16	Reports results of a custom MobiLink user authentication script. The MobiLink server provides this information to the client.  For more information, see <a href="#">“Authentication Value synchronization parameter”</a> [ <i>UltraLite - Database Management and Reference</i> ].
enable_resume	bool	If set to true, MLFileDownload resumes a previous download that was interrupted because of a communications error or because it was canceled by the user. If the file on the server is newer than the partial local file, the partial file is discarded and the new version is downloaded from the beginning. The default is true.
error	mlft_stream_error_w	Contains information about any error that occurs.
file_auth_code	asa_uint16	Contains the return code of the optional authenticate_file_transfer script on the server.
filename	wchar_t *	The file name to be transferred from the server running MobiLink. MobiLink searches the username subdirectory first, before defaulting to the root directory.  For more information, see <a href="#">“-ftr mlsrv12 option”</a> [ <i>MobiLink - Server Administration</i> ].

Member name	Type	Description
local_filename	wchar_t *	The local name for the downloaded file. If this parameter is empty, the value in file name is used.
local_path	wchar_t *	The local path to store the downloaded file. If this parameter is empty (the default), the downloaded file is stored in the current directory.  On Windows Mobile, if dest_path is empty, the file is stored in the root (\) directory of the device.  On the desktop, if the dest_path is empty, the file is stored in the user's current directory.
num_auth_parms	asa_uint8	The number of authentication parameters being passed to authentication parameters in MobiLink events.  For more information, see <a href="#">“Number of Authentication Parameters parameter” [UltraLite - Database Management and Reference]</a> .
observer	ml_file_transfer_observer_fn	A callback can be provided to observe file download progress through the 'observer' field. For more details, see description of Callback Function that follows.
password	wchar_t *	The password for the MobiLink user name.
remote_key	wchar_t *	The MobiLink remote key.
stream	const char *	Required. The protocol can be one of: TCPIP, TLS, HTTP, or HTTPS.  For more information, see <a href="#">“Stream Type synchronization parameter” [UltraLite - Database Management and Reference]</a> .
stream_parms	wchar_t *	The protocol options for a given stream.  For more information, see <a href="#">“Network protocol options for UltraLite synchronization streams” [UltraLite - Database Management and Reference]</a> .
transferred_file	asa_uint16	1 if the file was successfully transferred and 0 if an error occurs. An error occurs if the file is already up-to-date when MLFileUpload is invoked. In this case, the function returns true rather than false.

Member name	Type	Description
user_data	void *	The application-specific information made available to the synchronization observer.  For more information, see “ <a href="#">User Data synchronization parameter</a> ” [ <i>UltraLite - Database Management and Reference</i> ].
username	wchar_t *	Required. MobiLink user name.
version	wchar_t *	Required. The MobiLink script version.

### Remarks

#### Note

This structure prototype is used internally when you refer to ml\_file\_transfer\_info and #define the UNICODE macro on Win32 platforms. Typically, you would not reference this method directly when creating an UltraLite application.

## ml\_file\_transfer\_status structure

A structure containing status/progress information while the file upload/download is in progress.

### Syntax

```
public struct ml_file_transfer_status
```

### Members

Member name	Type	Description
bytes_transferred	asa_uint64	Indicates how much of the file has been downloaded so far, including previous synchronizations, if the download is resumed.
file_size	asa_uint64	The total size in bytes of the file being downloaded.
flags	asa_uint16	Provides additional information. The value MLFT_STATUS_FLAG_IS_BLOCKING is set when MLFileDownload is blocking on a network call and the download status has not changed since the last time the observer function was called.
info	ml_file_transfer_info *	Points to the info object passed to MLFileDownload. You can access the user_data parameter through this pointer.
resumed_at_size	asa_uint64	Used with download resumption and indicates at what point the current download resumed.

Member name	Type	Description
stop	asa_uint8	May be set to true to cancel the current download. You can resume the download in a subsequent call to MLFileDownload, but only if you have set the enable_resume parameter.

## mlft\_stream\_error structure

A structure containing status/progress information while the file upload/download is in progress.

### Syntax

```
public struct mlft_stream_error
```

### Members

Member name	Type	Description
error_string	char	The parameter to the stream error if one exists.
stream_error_code	ss_error_code	The specific stream error. See the ss_error_code enumeration for possible values.
system_error_code	asa_int32	A system-specific error code.

## mlft\_stream\_error\_w structure

A structure containing status/progress information while the file upload/download is in progress.

### Syntax

```
public struct mlft_stream_error_w
```

### Members

Member name	Type	Description
error_string	wchar_t	The parameter to the stream error if one exists.
stream_error_code	ss_error_code	The specific stream error. See the ss_error_code enumeration for possible values.
system_error_code	asa_int32	A system-specific error code.

**Remarks****Note**

This structure prototype is used internally when you refer to `mlft_stream_error` and `#define` the `UNICODE` macro on Win32 platforms. Typically, you would not reference this structure directly when creating an UltraLite application.

**See also**

- [“mlft\\_stream\\_error structure” on page 326](#)

## MLFT\_STATUS\_FLAG\_IS\_BLOCKING variable

Defines a bit set in the `ml_file_transfer_status.flags` field to indicate that the file transfer is blocked awaiting a response from the MobiLink server.

**Syntax**

```
#define MLFT_STATUS_FLAG_IS_BLOCKING 1
```

---



---

# Tutorial: Build an application using the C++ API

This tutorial guides you through the process of building an UltraLite C++ application. The application is built for Windows desktop operating systems, and runs at a command prompt.

This tutorial is based on development using Microsoft Visual C++, although you can also use any C++ development environment.

The tutorial takes about 30 minutes if you copy and paste the code. The final section of this tutorial contains the full source code of the program described in this tutorial.

## Competencies and experience

This tutorial assumes:

- You are familiar with the C++ programming language
- You have a C++ compiler installed on your computer
- You know how to create an UltraLite database with the **Create Database Wizard**.

For more information, see [“Create a database with the Create Database Wizard” \[UltraLite - Database Management and Reference\]](#).

The goal for the tutorial is to gain competence with the process of developing an UltraLite C++ application.

## Lesson 1: Create database and connect to database

In the first procedure, you create a local UltraLite database. You then write, compile, and run a C++ application that accesses the database you created.

### To create an UltraLite database

1. Set the **VCINSTALLDIR** environment variable to the root directory of your Visual C++ installation if the variable does not already exist.
2. Add `%VCINSTALLDIR%\VC\atlmfc\src\atl` to your **INCLUDE** environment variable.
3. Create a directory to contain the files you will create in this tutorial.

The remainder of this tutorial assumes that this directory is `C:\tutorial\cpp\`. If you create a directory with a different name, use that directory instead of `C:\tutorial\cpp\`.

4. Using UltraLite in Sybase Central, create a database named `ULCustomer.udb` in your new directory with the default characteristics.

For more information about using UltraLite in Sybase Central, see [“Create a database with the Create Database Wizard” \[UltraLite - Database Management and Reference\]](#).

5. Add a table named **ULCustomer** to the database. Use the following specifications for the ULCustomer table:

Column name	Data type (size)	Column allows NULL values?	Default value	Primary Key
cust_id	integer	No	autoincrement	ascending
cust_name	varchar(30)	No	None	

6. Disconnect from the database in Sybase Central, otherwise your executable will not be able to connect.

### To connect to the UltraLite database

1. In Microsoft Visual C++, choose **File » New**.
2. On the **Files** tab, choose **C++ Source File**.
3. Save the file as *customer.cpp* in your tutorial directory.
4. Include the UltraLite libraries.

Copy the code below into *customer.cpp*:

```
#include <tchar.h>
#include <stdio.h>
#include "ulcpp.h"
#define MAX_NAME_LEN 100
```

5. Define connection parameters to connect to the database.

In this code fragment, the connection parameters are hard coded. In a real application, the locations might be specified at runtime.

Copy the code below into *customer.cpp*.

```
static ul_char const * ConnectionParms =
    "UID=DBA;PWD=sql;DBF=C:\\\\tutorial\\cpp\\ULCustomer.udb";
```

For more information about connection parameters, see [“UltraLite connection parameters” \[UltraLite - Database Management and Reference\]](#).

#### Special Character handling

A backslash character that appears in the file name location string must be escaped by a preceding backslash character.

6. Define a method for handling database errors in the application.

UltraLite provides a callback mechanism to notify the application of errors. In a development environment this method can be useful as a mechanism to handle errors that were not anticipated. A

production application typically includes code to handle all common error situations. An application can check for errors after every call to an UltraLite method or can choose to use an error callback function.

The following code is a sample callback function:

```

ul_error_action UL_CALLBACK_FN MyErrorCallBack(
    const ULError *  error,
    ul_void *        user_data )
{
    ul_error_action rc;
    an_sql_code code = error->GetSQLCode();

    (void) user_data;

    switch( code ){
        // The following error is used for flow control - don't report it
        here
        case SQLE_NOTFOUND:
            rc = UL_ERROR_ACTION_CONTINUE;
            break;

        default:
            if (code >= 0) { // warning or success
                rc = UL_ERROR_ACTION_DEFAULT;
            } else { // negative is real error
                _tprintf( "Error %ld: %s\n", code, error->GetString() );
                rc = UL_ERROR_ACTION_CANCEL;
            }
            break;
    }
    return rc;
}

```

In UltraLite, the error `SQLE_NOTFOUND` is often used to control application flow. That error is signaled to mark the end of a loop over a result set. The generic error handler coded above does not output a message for this error condition.

For more information about error handling, see [“Handling errors” on page 24](#).

#### 7. Define a method to open a connection to a database.

If the database file does not exist, an error message is displayed, otherwise a connection is established.

```

static ULConnection * open_conn( void )
{
    ULConnection * conn =
    ULDatabaseManager::OpenConnection( ConnectionParms );
    if( conn == UL_NULL ) {
        _tprintf("Unable to open existing database.\n");
    }
    return conn;
}

```

#### 8. Implement the main method to carry out the following tasks:

- Registers the error handling method.
- Opens a connection to the database.
- Closes the connection and finalizes the database manager.

```
int main() {
    ul_char      buffer[ MAX_NAME_LEN ];
    ULConnection * conn;

    ULDatabaseManager::Init();
    ULDatabaseManager::SetErrorCallback( MyErrorCallBack, NULL );

    conn = open_conn();
    if ( conn == UL_NULL ) {
        ULDatabaseManager::Fini();
        return 1;
    }

    // Main processing code goes here ...

    conn->Close();
    ULDatabaseManager::Fini();
    return 0;
}
```

### 9. Compile and link the source file.

The method you use to compile the source file depends on your compiler. The following instructions are for the Microsoft Visual C++ command line compiler using a makefile:

- Open a command prompt and change to your tutorial directory.
- Create a makefile named *makefile*.
- In the makefile, add directories to your include path.

```
IncludeFolders=/I"${SQLANY12}\SDK\Include"
```

- In the makefile, add directories to your libraries path.

```
LibraryFolders=/LIBPATH:"${SQLANY12}\UltraLite\win32\386\Lib\vs8"
```

- In the makefile, add libraries to your linker options.

```
Libraries=ulimp.lib
```

The UltraLite runtime library is named *ulimp.lib*.

- In the makefile, set compiler options. You must enter these options on a single line.

```
CompileOptions=/c /nologo /W3 /Od /Zi /DWIN32 /DUL_USE_DLL
```

- In the makefile, add an instruction for linking the application.

```
customer.exe: customer.obj
    link /NOLOGO /DEBUG customer.obj $(LibraryFolders) $(Libraries)
```

- In the makefile, add an instruction for compiling the application.

```
customer.obj: customer.cpp
    cl $(CompileOptions) $(IncludeFolders) customer.cpp
```

- Run *vsvars32.bat*.

```
%VCINSTALLDIR%\Tools\vsvars32.bat
```

- Run the makefile.

```
nmake
```

This creates an executable named *customer.exe*.

10. Run the application.

At the command prompt, enter **customer**.

## Lesson 2: Insert data into the database

The following procedure demonstrates how to add data to a database.

### To add rows to your database

1. Add the method below to *customer.cpp*, immediately before the main method:

```
static bool do_insert( ULConnection * conn )
{
    ULTable * table = conn->OpenTable( "ULCustomer" );
    if( table == UL_NULL ) {
        _tprintf( "Table not found: ULCustomer\n" );
        return false;
    }
    if( table->GetRowCount() == 0 ) {
        _tprintf( "Inserting one row.\n" );
        table->InsertBegin();
        table->SetString( "cust_name", "New Customer" );
        table->Insert();
        conn->Commit();
    } else {
        _tprintf( "The table has %lu rows\n", table->GetRowCount() );
    }
    table->Close();
    return true;
}
```

This method performs the following tasks.

- Opens the table using the `connection->OpenTable()` method. You must open a Table object to carry out operations on the table.
  - If the table is empty, adds a row to the table. To insert a row, the code changes to insert mode using the `InsertBegin` method, sets values for each required column, and executes an insert to add the row to the database.
  - If the table is not empty, reports the number of rows in the table.
  - Closes the Table object to free resources associated with it.
  - Returns a boolean indicating whether the operation was successful.
2. Call the `do_insert` method you have created.

Add the following line to the `main()` method, immediately before the call to `conn->Close`.

```
do_insert( conn );
```

3. Compile your application by running `nmake`.

4. Run your application by typing **customer** at the command prompt.

## Lesson 3: Select and list rows from the table

The following procedure retrieves rows from the table and prints them on the command line.

### To list rows in the table

1. Add the method below to *customer.cpp*. This method carries out the following tasks:

- Opens the Table object.
- Retrieves the column identifiers.
- Sets the current position before the first row of the table.

Any operations on a table are carried out at the current position. The position may be before the first row, on one of the rows of the table, or after the last row. By default, as in this case, the rows are ordered by their primary key value (*cust\_id*). To order rows in a different way, you can add an index to an UltraLite database and open a table using that index.

- For each row, the *cust\_id* and *cust\_name* values are written out. The loop carries on until the Next method returns false, which occurs after the final row.
- Closes the Table object.

```
static bool do_select( ULConnection * conn )
{
    ULTable * table = conn->OpenTable( "ULCustomer" );
    if( table == UL_NULL ) {
        return false;
    }
    ULTableSchema * schema = table->GetTableSchema();
    if( schema == UL_NULL ) {
        table->Close();
        return false;
    }
    ul_column_num id_cid =
        schema->GetColumnID( "cust_id" );
    ul_column_num cname_cid =
        schema->GetColumnID( "cust_name" );
    schema->Close();

    _tprintf( "\n\nTable 'ULCustomer' row contents:\n" );
    while( table->Next() ) {
        ul_char cname[ MAX_NAME_LEN ];
        table->GetString( cname_cid, cname, MAX_NAME_LEN );
        _tprintf( "id=%d, name=%s \n", (int)table->GetInt(id_cid),
cname );
    }
    table->Close();
    return true;
}
```

2. Add the following line to the main method immediately after the call to the insert method:

```
do_select(conn);
```

3. Compile your application by running *nmake*.
4. Run your application by typing *customer* at the command prompt.

## Lesson 4: Add synchronization to your application

This lesson synchronizes your application with a consolidated database running on your computer.

The following procedures add synchronization code to your application, start the MobiLink server, and run your application to synchronize.

The UltraLite database you created in the previous lessons synchronizes with the UltraLite 12 Sample database. The UltraLite 12 Sample database has a ULCustomer table whose columns include those in the customer table of your local UltraLite database.

This lesson assumes that you are familiar with MobiLink synchronization.

### To add synchronization to your application

1. Add the method below to *customer.cpp*. This method carries out the following tasks:
  - Enables TCP/IP communications by invoking `EnableTcpipSynchronization`. Synchronization can also be carried out over HTTP, HTTPS, and TLS. See “[UltraLite clients](#)” [[UltraLite - Database Management and Reference](#)].
  - Sets the script version. MobiLink synchronization is controlled by scripts stored in the consolidated database. The script version identifies which set of scripts to use.
  - Sets the MobiLink user name. This value is used for authentication at the MobiLink server. It is distinct from the UltraLite database user ID, although in some applications you may want to give them the same value.
  - Sets the `download_only` parameter to true. By default, MobiLink synchronization is two-way. This application uses download-only synchronization so that the rows in your table do not get uploaded to the sample database.

```
static bool do_sync(ULConnection * conn )
{
    ul_sync_info info;
    ul_stream_error * se = &info.stream_error;

    ULDatabaseManager::EnableTcpipSynchronization();
    conn->InitSyncInfo( &info );
    info.stream = "TCPIP";
    info.version = "custdb 12.0";
    info.user_name = "50";
    info.download_only = true;
    if( !conn->Synchronize( &info ) ) {
        _tprintf( "Synchronization error \n" );
        _tprintf( "  stream_error_code is '%lu'\n", se-
>stream_error_code );
        _tprintf( "  system_error_code is '%ld'\n", se-
>system_error_code );
        _tprintf( "  error_string is ' ' );
    }
}
```

```
        _tprintf( "%s", se->error_string );
        _tprintf( "'\n" );
        return false;
    }
    return true;
}
```

2. Add the following line to the main method, immediately after the call to the insert method and before the call to the select method:

```
do_sync(conn);
```

3. Compile your application by running *nmake*.

### Synchronize data

1. Start the MobiLink server.

From a command prompt, run the following command:

```
mlsrv12 -c "dsn=SQL Anywhere 12 CustDB;uid=ml_server;pwd=sql" -v -vr -vs -zu+ -o custdbASA.log
```

The `-zu+` option provides automatic addition of users. The `-v+` option turns on verbose logging for all messages.

For more information about this option, see [“MobiLink server options”](#) [*MobiLink - Server Administration*].

2. Run your application by typing *customer* at the command prompt.

The MobiLink server messages window displays status messages indicating the synchronization progress. If synchronization is successful, the final message displays `Synchronization complete`.

## Code listing for tutorial

The following is the complete code for the tutorial program described in the preceding sections.

```
#include <tchar.h>
#include <stdio.h>

#include "ulcpp.h"

#define MAX_NAME_LEN 100

static ul_char const * ConnectionParms =
    "UID=DBA;PWD=sql;DBF=c:\\tutorial\\cpp\\ULCustomer.udb";

ul_error_action UL_CALLBACK_FN MyErrorCallBack(
    const ULError * error,
    ul_void * user_data )
{
    ul_error_action rc;
    an_sql_code code = error->GetSQLCode();
}
```



```

        (void) user_data;

        switch( code ){
            // The following error is used for flow control - don't report it
here        case SQLE_NOTFOUND:
                rc = UL_ERROR_ACTION_CONTINUE;
                break;

            default:
                if (code >= 0) { // warning or success
                    rc = UL_ERROR_ACTION_DEFAULT;
                } else { // negative is real error
                    _tprintf( "Error %ld: %s\n", code, error->GetString() );
                    rc = UL_ERROR_ACTION_CANCEL;
                }
                break;
        }
        return rc;
    }

    static ULConnection * open_conn( void ) {
        ULConnection * conn =
        ULDatabaseManager::OpenConnection( ConnectionParms );
        if( conn == UL_NULL ) {
            _tprintf( "Unable to open existing database.\n" );
        }
        return conn;
    }

    static bool do_insert( ULConnection * conn ) {
        ULTable * table = conn->OpenTable( "ULCustomer" );
        if( table == UL_NULL ) {
            _tprintf( "Table not found: ULCustomer\n" );
            return false;
        }
        if( table->GetRowCount() == 0 ) {
            _tprintf( "Inserting one row.\n" );
            table->InsertBegin();
            table->SetString( "cust_name", "New Customer" );
            table->Insert();
            conn->Commit();
        } else {
            _tprintf( "The table has %lu rows\n",
                table->GetRowCount() );
        }
        table->Close();
        return true;
    }

    static bool do_select( ULConnection * conn )
    {
        ULTable * table = conn->OpenTable( "ULCustomer" );
        if( table == UL_NULL ) {
            return false;
        }
        ULTableSchema * schema = table->GetTableSchema();
        if( schema == UL_NULL ) {
            table->Close();
            return false;
        }
        ul_column_num id_cid =
            schema->GetColumnID( "cust_id" );
    }

```

```

        ul_column_num cname_cid =
            schema->GetColumnID( "cust_name" );

        schema->Close();

        _tprintf( "\n\nTable 'ULCustomer' row contents:\n" );

        while( table->Next() ) {
            ul_char cname[ MAX_NAME_LEN ];

            table->GetString( cname_cid, cname, MAX_NAME_LEN );

            _tprintf( "id=%d, name=%s \n", (int)table->GetInt(id_cid), cname );
        }
        table->Close();
        return true;
    }

static bool do_sync( ULConnection * conn )
{
    ul_sync_info info;
    ul_stream_error * se = &info.stream_error;

    ULDatabaseManager::EnableTcpipSynchronization();
    conn->InitSyncInfo( &info );
    info.stream = "TCPIP";
    info.version = "custdb 12.0";
    info.user_name = "50";
    info.download_only = true;
    if( !conn->Synchronize( &info ) ) {
        _tprintf( "Synchronization error \n" );
        _tprintf( "  stream_error_code is '%lu'\n", se->stream_error_code );
        _tprintf( "  system_error_code is '%ld'\n", se->system_error_code );
        _tprintf( "  error_string is '" );
        _tprintf( "%s", se->error_string );
        _tprintf( "'\n" );
        return false;
    }
    return true;
}

int main()
{
    ul_char          buffer[ MAX_NAME_LEN ];
    ULConnection *   conn;

    ULDatabaseManager::Init();
    ULDatabaseManager::SetErrorCallback( MyErrorCallBack, NULL );

    conn = open_conn();
    if( conn == UL_NULL ){
        ULDatabaseManager::Fini();
        return 1;
    }

    // Main processing code goes here ...

    conn->Close();
    ULDatabaseManager::Fini();
    return 0;
}

```

---

# Tutorial: Build an iPhone application using the C++ API

This tutorial guides you through the process of building an UltraLite C++ application for the iPhone using the Apple's development tools. Wherever possible, this tutorial provides links to more detailed information.

This tutorial is based on development using Microsoft Visual C++, although you can use any C++ development environment.

## Competencies and experience

This tutorial assumes:

- You are familiar with Objective-C and C++ programming languages
- You have a C++ compiler installed on your computer
- You have the iPhone SDK installed on your computer
- You have enrolled in the iPhone Developer Program (This is only required to run the program on a physical device. Running on the iPhone Simulator does not require enrollment.)

The goal for the tutorial is to gain competence with the process of developing an UltraLite C++ iPhone application.

## Compiling the UltraLite iPhone library

In order to use UltraLite on the iPhone, you must first compile the library.

### To compile the library for the iPhone simulator

1. Extract the UltraLite for iPhone tarball by double-clicking it. This will create a *sqlany* folder in the same location as the tarball.
2. Navigate to the *sqlany/UltraLite/iPhone/x86* directory you just extracted.
3. Extract the *ulrt.tar.gz* tarball by double-clicking it. This will extract many source files, along with an *ulrt.make* makefile.
4. Open the makefile using **TextEdit** or any editor of your choice.
5. Change the **ISYSROOT** variable so that it refers to your version of the iPhone SDK. If you are unsure of what version you have, navigate to: */Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/*.

6. If you installed the iPhone SDK to a non-default location, adjust the other makefile variables to match the location of the installation.
7. Open a Terminal and `cd` to the directory that has the makefile.
8. Run the command: `make -f ulrt.make` and wait for compilation to finish. If you receive any errors, go back and verify that the variables in the makefile point to the correct location of the iPhone SDK.
9. Once compilation is complete, verify that the `ulrt.a` library archive has been created and move it so that it resides in the `sqlany/UltraLite/iPhone/x86` directory since the example projects expect it at that location. Extracting the tar file using the **Archive Utility** might have created an additional `ulrt` folder.

## Creating an UltraLite application on iPhone

The first part of this tutorial walks you through the process of creating an iPhone application that maintains a list of names in a simple, single-table UltraLite database. The second part of the tutorial adds synchronization with a SQL Anywhere database to the application (see [“Lesson 6: Adding synchronization” on page 352](#)).

## Lesson 1: Creating a new iPhone application project

Before you can start doing anything, you must first create an Xcode project.

### To create an xCode project

1. Launch Xcode.
2. Under the **File** menu, choose **New Project...**
3. Choose **iPhone OS Application** from the left-hand side selection.
4. Choose a **Navigation-based Application**, leaving the **Use Core Data for Storage.** option unchecked.
5. Click the **Choose...** button and save the project **names** in the location of your choice.

The Navigation-based project automatically creates an application with a `UITableViewController` inside of a `UINavigationController`, as well as a `UIApplicationDelegate` which places the controllers in the window when the application is launched. So far, the application only has an empty navigation bar at the top, and an empty table view. To fill the table with names, the application will request the data from an UltraLite database on the device.

### Setting up the project for an UltraLite library.

#### To set up the project for the UltraLite library

1. In order to use the UltraLite library you must configure the Xcode project.

2. Control-click the names project on the left-hand side of the project window and select **Get Info**.
3. Under the **Build** tab of the Info window, search for the **User Header Search Paths** setting using the search field and double-click the setting.
4. In the **Finder**, navigate to the SQL Anywhere *sdk* folder.
5. Click and drag the *Include* folder into the modal window that opened when you double-clicked the setting in the info window.
6. Click **OK**.

### Setting up the UltraLite preprocessor macro

The UltraLite headers can now be used in this project. However, the library requires that the `UNDER_IPHONE` preprocessor macro be defined to function correctly. On some versions of the SDK the `GCC_PREPROCESSOR_DEFINITIONS` is already defined. If this is the case, simply set its value to `UNDER_IPHONE`.

Otherwise, to create it:

#### To set the UltraLite preprocessor macro

1. Click the cog button in the bottom-left corner of the project info window and select *Add User-Defined Setting*.
2. Name the new setting `GCC_PREPROCESSOR_DEFINITIONS` and set the value to `UNDER_IPHONE`.

### Compiling the project as C++

This tutorial uses the UltraLite C++ API. In order to eliminate the need to cast to C types, compile the source as C++:

#### To compile the project as C++

1. In the **Search in Build Settings** box, enter **Compile Sources As**.
2. Choose **Objective-C++** from the options in the **Value** field.

**Note**

On some older versions of Xcode, this setting is only visible if a device target is selected for this project.

3. Close the project info window.

### Adding the UltraLite library to the project

The only step left to be able to use UltraLite in the application is to add the library itself to the project.

### To add the UltraLite library to the project

1. In the **Finder**, navigate to the *libulrt.a* library you compiled earlier. It should be located in *ultralite/iphone* in your SQL Anywhere folder.
2. Click and drag the *libulrt.a* library from the **Finder** onto the **Names** project in the project window. (The same element you control-clicked to get the project info window.)
3. You should now have *libulrt.a* listed in the project files with a little yellow toolbox icon.

### Adding frameworks required by UltraLite

UltraLite requires the *CFNetwork* and *Security* frameworks in addition to the default frameworks.

### To add the frameworks required by UltraLite

1. Control-click the **Frameworks** folder under the **names** project in the project window and select **Existing Frameworks** from the **Add** menu.
2. Browse to the **Security** framework and choose **Add**. (*/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS3.1.3.sdk/System/Library/Frameworks/Security.framework*) for example.
3. Choose **Reference Type** of **Relative to Current SDK**.
4. Repeat with the *CFNetwork* framework.

## Lesson 2: Adding a database to the application

In order to interact with the database, the application follows the MVC pattern prescribed by the Apple framework and defines a Model class.

### Data Model Class

The Data Access class will be the only class which interacts with the database. This way, if a schema change or other database-related change is required, there is only one place where updates are required.

### To set up the Data Model Class

1. Control-click the *Classes* folder in the **Project** window. In the **File** menu, select **New File** from the **Add** menu.
2. Choose **iPhone OS Cocoa Touch Class** in the left-hand selection.
3. Select a new Objective-C class.
4. Make sure the **Subclass** of selection is **NSObject**.
5. Click **Next**.

6. Name the file *DataAccess.mm*. The .mm extension is important as it signals to Xcode that the file contains both Objective-C and C++.
7. Make sure the **Also create DataAccess.h** box is checked.
8. For **Location**, choose the *Names/Classes* subfolder.
9. Click **Finish**.

### The **DataAccess** singleton

When the application launches, it needs to initialize the UltraLite database manager and connect to the local database. To do this, the class creates a singleton instance of this data access object which is used by the **RootViewController** to display and manage the list of names.

In the *DataAccess.mm* file, add the following code in the implementation:

```
static DataAccess *      sharedInstance = nil;

+ (DataAccess *)sharedInstance {
    // Create a new instance if none was created yet
    if (sharedInstance == nil) {
        sharedInstance = [[super alloc] init];
        [sharedInstance openConnection];
    }

    // Otherwise, just return the existing instance
    return sharedInstance;
}
```

Before being able to use any UltraLite classes and methods, you have to import the *ulcpp* header file. Add the following to the existing imports in *DataAccess.h*:

```
#import "ulcpp.h"
```

### Open the connection to the database

Before the connection can be opened, the database managers *Init* method is used to initialize the UltraLite runtime. Once initialized, an attempt to connect to the database will indicate whether the database exists. Add the following instance variable to the *DataAccess* header:

```
ULConnection *      connection;
```

In the *DataAccess.mm* file, add the following code in the implementation:

```
- (void)openConnection {
    NSLog(@"Connect to database.");
    if (ULDatabaseManager::Init()) {
        NSArray *      paths = NSSearchPathForDirectoriesInDomains(
            NSDocumentDirectory,
            NSUserDomainMask,
            YES);
        NSString *      documentsDirectory = [paths objectAtIndex:0];
        NSString *      writableDBPath = [documentsDirectory
            stringByAppendingPathComponent:
            @"Names.udb"];
    }
}
```

```
ULConnection *    conn = nil;
const char *      connectionParms;
ULError          error;

connectionParms = [[NSString stringWithFormat:@"DBF=%@",
        writableDBPath]
        UTF8String];

// Attempt connection to the database
conn = ULDatabaseManager::OpenConnection(
        connectionParms,
        &error);
// If database file not found, create it and create the schema
if (error.GetSQLCode() == SQLE_ULTRALITE_DATABASE_NOT_FOUND) {
    conn = [self createDatabase:connectionParms];
}
connection = conn;
} else {
    NSLog(@"UL Database Manager initialization failed.");
    connection = nil;
}
}
```

### Database access

The database schema of the application is composed of a single table with two columns. The Names table has an ID column which uses UUIDs and a name column which stores the names as VARCHARs. The ID column uses UUIDs to easily support row insertions from remote databases using MobiLink. This is discussed in [“Lesson 6: Adding synchronization” on page 352](#).

Displaying data in a table view requires that each row be accessible using a 1-based index. To do this, the database uses an ascending index on the name column. Each row's index will be equal to its position in the alphabetical list of names.

In the *DataAccess.mm* file, add the following code in the implementation:

```
- (ULConnection *)createDatabase:(const char *)connectionParms {
    const char *    CREATE_TABLE =
        "CREATE TABLE Names ( "
        "id UNIQUEIDENTIFIER DEFAULT NEWID() PRIMARY KEY,"
        "name VARCHAR(254) NOT NULL)";
    const char *    CREATE_INDEX =
        "CREATE UNIQUE INDEX namesIndex ON Names(name ASC)";
    const char *    createParms =
        "page_size=4k;utf8_encoding=true;collation=UTF8BIN";
    ULError          error;
    ULConnection *    conn;

    conn = ULDatabaseManager::CreateDatabase(
        connectionParms,
        createParms,
        &error);

    if (!conn) {
        NSLog(@"Error code creating the database: %ld",
            error.GetSQLCode());
    } else {
        NSLog(@"Creating Schema.");
        conn->ExecuteStatement(CREATE_TABLE);
        conn->ExecuteStatement(CREATE_INDEX);
    }
}
```



```
    return conn;
}
```

**Note**

*openConnection* uses *createDatabase*, so either *createDatabase* must come before *openConnection* or its method signature must be added to the header file.

Add a dealloc method to finalize the UltraLite runtime,

```
- (void)dealloc {
    NSLog(@"Finalizing DB Manager.");
    connection->Close();
    ULDatabaseManager::Fini();

    [super dealloc];
}
```

And a fini method to release the instance,

```
+ (void)fini {
    [sharedInstance release];
}
```

Add the method signatures to the header file after the interface curly brace block:

```
// Release objects.
- (void)dealloc;

// Singleton instance of the DataAccess class.
+ (DataAccess*)sharedInstance;

// Finalize the Database Manager when done with the DB.
+ (void)fini;
```

Call the fini method from the **NamesAppDelegate's applicationWillTerminate** method:

```
- (void)applicationWillTerminate:(UIApplication *)application {
    // Save data if appropriate
    [DataAccess fini];
}
```

Also, since the application delegate is calling fini, it must:

```
#import "DataAccess.h"
```

## Lesson 3: Adding data to the database

Now that the application has a database on the device with its schema initialized, it can start adding names to the database. To do this, the application will use a new screen with a text field.

### View controller for new names

To let the user input new names you will create a new view controller:

### To set up the view controller for new names

1. In the **File** menu, select **New File...**
2. With the **iPhone OS Cocoa Touch Class** selected on the left-hand side selection, choose **UIViewController** subclass.
3. Do not check the **UITableViewController** subclass option and select (check) the **With XIB for user interface** option.
4. Click **Next**.
5. Name the file *NewNameViewController.m*.
6. Select the *Names/Classes/* location.
7. Click **Finished**.

This will create three files: the **NewNameViewController** header and implementation file, as well as a XIB file that can be edited in Interface Builder. To group all the XIB files together, you can move it to the *Resources* folder where two XIB files already exist (the **MainWindow** and **RootViewController**'s XIB files).

Before editing the XIB file to contain the text field, you must first add an **Outlet** property to the header so that you can tie in the text field from the XIB file into the code as well as an action to perform when the user has finished entering the name.

```
@interface NewNameViewController : UIViewController {
    UITextField *newNameField;
}
@property (retain) IBOutlet UITextField *newNameField;
- (IBAction)doneAdding:(id)sender;
@end
```

Synthesize the property in the implementation file and release the text field in the dealloc method. You will define the action later, once the **DataAccess** object can insert new names. For now, simply create an empty method stub to avoid compilation warnings.

```
@synthesize newNameField;
- (IBAction)doneAdding:(id)sender {}
```

The **IBAction** keyword lets **Interface Builder** know that the method should be made available for events to call. Make sure to save the header so that **Interface Builder** knows about the method.

Now that the outlet is set up, double-click the **NewNameViewController** XIB file to edit it in **Interface Builder**. In **Interface Builder**, you should see an empty view with a status bar showing the battery charging. Since this view also has a navigation bar showing, simulate showing it by doing the following:

1. In the **Document Window** (Command-0), select the **View**.
2. In the **Attribute Inspection Window** (Command-1), under the **Simulated Interface Elements**, set the **Top Bar** to **Navigation Bar**.

The view should now show an empty navigation bar just below the status bar.

To allow the user to input a new name, add a text field to the view:

1. In the **Library** window (Command-Shift-L), under **Inputs & Values**, click-and- drag a text field into the view.
2. Position the text field in the upper half of the view, so that the keyboard in the lower half won't hide it.
3. Make the text field a little wider so that a name would fit comfortably within it. A width of about 230 pixels should work. To see the size and placement options, use the **Size Inspector** (Command-3) while the text field is selected.

To make the text field more user-friendly, you can change a few properties:

1. Select the text field in the view.
2. With the text field selected, open the **Attribute Inspection Window** (Command-1).
3. Set the **Placeholder** to **Name**.
4. Make the font size 18 points.
5. Set **Capitalize** to **Words**.
6. Set the **Return Key** to **Done**.

In order for the controller to be made aware of when the user is done entering the name, you must make action connections in **Interface Builder**.

1. Open the **Connections Inspector** (Command-2).
2. Click and drag from the **Did End on Exit** event circle to the **File's Owner** in the **Document Window**, and select **done Adding** to make the connection. The **File's Owner** of this XIB file is the *NewItemViewController* class, as the **File's Owner's** type indicates. If dragging and dropping over the **File's Owner** doesn't provide a selection, make sure the *NewItemViewController* header is saved and has the **IBOutlet doneAdding**.

In order to refer to the text field in the code, you need to connect this text field to the **IBOutlet** property you defined in the header earlier.

1. Select the **File's Owner** in the **Document Window**. Open the **Connections Inspector** (Command-2).
2. Under **Outlets**, look for **newNameField**.
3. Click and drag from the **newNameField** circle to the actual name field in the view.
4. The view is now complete. Save the XIB file in **Interface Builder** and return to Xcode.

To finish the view, you need to set a few more properties in code. Uncomment the **viewDidLoad** method template in the **NewNameViewController** implementation and add the following code:

```
- (void)viewDidLoad {
    [super viewDidLoad];

    // Set the title to display in the nav bar
    self.title = @"Add Name";

    // Set the text field to the first responder to display the keyboard.
    // Without this the user needs to tap on the text field.
    [newNameField becomeFirstResponder];
}
```

### Root view controller setup

To display the view controller you just created, the **RootViewController** needs to be configured. Import the **NewNameViewController** and **DataAccess** headers in the **RootViewController**'s implementation file and add the following method signature to its header:

```
- (void)showAddNameScreen;
```

Implement the method with the following code:

```
- (void)showAddNameScreen {
    NewNameViewController * addNameScreen = [[NewNameViewController alloc]
    initWithNibName:@"NewNameViewController" bundle:nil];
    [self.navigationController pushViewController:addNameScreen
    animated:YES];
}
```

Similar to how you set the title for the **NewNameViewController**, set the title of the **RootViewController**, as well as add a plus-sign button to the right side of the navigation bar that calls **showAddNameScreen**.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    // The Navigation Controller uses this to display the title in the nav
    bar.
    self.title = @"Names";
    // Little button with the + sign on the right in the nav bar
    self.navigationItem.rightBarButtonItem =
    [[UIBarButtonItem alloc]
    initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
    target:self
    action:@selector(showAddNameScreen)];
}
```

### Inserting new names into the database

In order to insert the new names into the database, you will add functionality to the **DataAccess** object. Add the following method signature to the **DataAccess** header:

```
// Adds the given name to the database.
- (void)addName:(NSString *)name;
```

To perform the insertion, the application will use the following SQL statement:

```
const char * INSERT = "INSERT INTO Names(name) VALUES(?)";
```

The **addName** method can now be implemented:

```

- (void)addName:(NSString *)name {
    const char *      INSERT = "INSERT INTO Names(name) VALUES(?)";
    ULPreparedStatement *  prepStmt = connection->PrepareStatement(INSERT);

    if (prepStmt) {
        // Convert the NSString to a C-Style string using UTF8 Collation
        prepStmt->SetParameterString(1, [name UTF8String], [name length]);
        prepStmt->ExecuteStatement();
        prepStmt->Close();
        connection->Commit();
    } else {
        NSLog(@"Could not prepare INSERT statement.");
    }
}

```

Now that the **addName** method is complete, the **NewNameViewController**'s **doneAdding** method can be implemented:

```

- (IBAction)doneAdding:(id)sender {
    if (newNameField.text > 0) {
        [[DataAccess sharedInstance] addName:newNameField.text];
    }
    [self.navigationController popViewControllerAnimated:YES];
}

```

You should now import the **DataAccess** header. The keyboard's **done** button now adds the name to the database and returns to the table view. However, the table view is not yet configured to display what is in the database.

## Lesson 4: Displaying data from the database

By default, the data source of a table view is its controller. In the case of this application, the **RootViewController** is currently the data source for the table view. In this lesson, you will implement the **UITableViewDataSource** protocol in the **DataAccess** class. This way, the **DataAccess** class will be responsible for providing the table view with the data it requires.

To begin, add the protocol to the **DataAccess** header.

```

@interface DataAccess : NSObject <UITableViewDataSource> {
    ULConnection *      connection;
}

```

### Getting the number of names in the database

The **UITableViewDataSource** protocol has two required methods. This section will implement one of those methods, namely, the **tableView:numberOfRowsInSection:** method. The table view has a single section: a list of names. Therefore, this method will simply return the count of the rows in the database. Use the following SQL statement:

```

SELECT COUNT (*) FROM Names

```

Add the following method to get the count:

```

- (NSInteger)tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)section {

```

```
const char *          COUNT = "SELECT COUNT (*) FROM Names";
ULPreparedStatement *  prepStmt = connection->PrepareStatement(COUNT);

if (prepStmt) {
    ULResultSet      *resultSet = prepStmt->ExecuteQuery();
    int              numberOfNames;

    resultSet->First();
    numberOfNames = resultSet->GetInt(1);
    resultSet->Close();
    prepStmt->Close();
    return numberOfNames;
} else {
    NSLog(@"Couldn't prepare COUNT.");
}

return 0;
}
```

### Creating the table cells

The second required method in the protocol is the **tableView:cellForRowAtIndexPath:** method. This method is responsible for creating the actual cell objects that the table view displays. The **RootViewController** defines a good template to reuse cells. Table cells are usually recycled using the pattern in the template to save memory and allow for fast scrolling. The only identifier the table view provides is a 0-based integer index. However, the database does not normally keep an ordered list of the names. In order to simulate an ordered list, the database uses an ordered index on the names and the application will use a select statement with a row limitation and an order by clause to select a given index. Although this method doesn't require the FOR UPDATE clause, this same query will be used to delete names in [“Lesson 5: Deleting data from the database” on page 351](#). Add the following to *DataAccess.mm* just below the import statements:

```
#define SELECT_STMT @"SELECT TOP 1 START AT %d name FROM Names ORDER BY name FOR UPDATE"
```

To use this query in the method do the following in the *DataAccess* implementation:

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *    CellIdentifier = @"Cell";
    UITableViewCell *    cell =
        [tableView dequeueReusableCellWithIdentifier: CellIdentifier];
    ULPreparedStatement *  prepStmt;

    if (cell == nil) {
        cell = [[[UITableViewCell alloc]
            initWithStyle:UITableViewCellStyleDefault
            reuseIdentifier:CellIdentifier]
            autorelease];
    }

    // Make it so the cell cannot be selected.
    cell.selectionStyle = UITableViewCellSelectionStyleNone;

    // +1 to the index since the DB uses a 1-based index rather than 0-based
    prepStmt = connection->PrepareStatement(
        [NSString stringWithFormat:SELECT_STMT, indexPath.row + 1]
        UTF8String);
    if (prepStmt) {
```

```

        ULResultSet *    resultSet = prepStmt->ExecuteQuery();
        char            name[255];

        resultSet->First();
        resultSet->GetString("name", name, 255);
        resultSet->Close();
        prepStmt->Close();
        cell.textLabel.text = [NSString stringWithUTF8String:name];
    } else {
        NSLog(@"Couldn't prepare SELECT with index.");
    }

    return cell;
}

```

### Setting the `DataAccess` object as the data source

Now that the `DataAccess` object is configured as a data source, you need to set the data source of the table. To do this, add the following at the end of the `viewDidLoad` method of the `RootViewController` class:

```

// Set the data source.
[self.tableView setDataSource:[DataAccess sharedInstance]];

```

You also need to add the following to the `viewWillAppear`. Without it, changes are not picked up until the rows scroll out of view and back in:

```

[self.tableView reloadData];

```

## Lesson 5: Deleting data from the database

The application is now able to display names from the database, and the user is able to add new names using a custom view. It is not yet possible for the user to remove a name. This lesson adds the deletion functionality using the swipe-to-delete functionality available in many iPhone applications.

### Deleting the name from the database

To delete the name from the database, use the same SQL statement as the select, but open the result set for update so that you can delete the selected row. Add the following method to the `DataAccess` class:

```

- (void)removeNameAtIndexPath:(NSIndexPath *)indexPath {
    // +1 to the index since the DB uses a 1-based index rather than 0-based
    ULPreparedStatement *prepStmt =
        connection->PrepareStatement(
            [[NSString stringWithFormat:SELECT_STMT, indexPath.row + 1]
             UTF8String]);

    if (prepStmt) {
        ULResultSet            *resultSet = prepStmt->ExecuteQuery();

        resultSet->First();
        resultSet->Delete();
        resultSet->Close();
        prepStmt->Close();
        connection->Commit(); // Commit the deletion.
    } else {
        NSLog(@"Couldn't prepare SELECT with index for delete.");
    }
}

```

```
}  
}
```

Also add the method signature to the header.

### Enabling the swipe-to-delete

In order to enable the delete functionality, simply copy the comment template of the **tableView:commitEditingStyle:forRowAtIndexPath** method from **RootViewController** into the **DataAccess** class and call the **removeNameAtIndexPath** method you just created.

```
- (void)tableView:(UITableView *)tableView commitEditingStyle:  
(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath  
*)indexPath {  
  
    if (editingStyle == UITableViewCellEditingStyleDelete) {  
        // Delete the row from the data source.  
        [[DataAccess sharedInstance] removeNameAtIndexPath:indexPath];  
        [tableView deleteRowsAtIndexPaths:[NSArray  
arrayWithObject:indexPath]  
         withRowAnimation:UITableViewRowAnimationFade];  
    } else if (editingStyle == UITableViewCellEditingStyleInsert) {  
        // Create a new instance of the appropriate class, insert it into the  
        array, and add a new row to the table view.  
    }  
}
```

The application is now complete. The table view displays the list of names in alphabetical order, new names can be added using the **NewNameViewController** and names can be removed using the familiar swipe-to-delete gesture.

## Lesson 6: Adding synchronization

Now that the Names application can add and remove names from an UltraLite database, you will add synchronization to a consolidated database server on your computer. In order to complete this lesson you will need to have SQL Anywhere installed on your computer. If you receive any errors or warnings during this lesson, make sure you have configured your installation correctly and have the proper environment variables set.

### Creating the consolidated database

1. Open Sybase Central.
2. Under the **Tools** menu, choose **SQL Anywhere 12 -> Create Database...**
3. Click **Next** on the **Welcome** screen and the **Select a Location** screen.
4. Choose a location to save the database file and name it *Names.db* and click **Finish**. The rest of the options can remain as their defaults
5. When the database has been created, close the popup window.



### Create the consolidated table

In order to synchronize, you will first need to create a table on the consolidated database:

1. If the left-hand selection is **Tasks** or **Search**, change it to **Folders**, by selecting **Folders** from the **View** menu.
2. In the **Folders** pane, Control-click the **Tables** element under the **Names** database and select **New -> Table**.
3. Name the table "Names" and click **Finish**. This will create the table and ready the cursor to name the primary key column of the table.
4. Name the primary key of the table "**id**" with a **uniqueidentifier** type.
5. Click the [...] button under the **Value** heading and set the default value to the **User-defined value: NEWID()**.
6. Make sure the **Literal string** checkmark is not checked and click **OK**.
7. Add another column by clicking the **New Column** button on the toolbar.
8. Name the new column "name" of type VARCHAR and size 254.
9. Uncheck the **Null** checkmark for the name column.
10. Check the **Unique** checkmark for the name column.
11. Click the **Save** button on the toolbar.
12. Disconnect from the database.

### Create an ODBC data source

In order to set up MobiLink, the system must have an ODBC data source for the consolidated database. Before you can setup the ODBC data source, you must first install the SQL Anywhere ODBC driver:

1. Open a Terminal.
2. Type the following command to create the ODBC data name source:

```
dbdsn -w "Names" -c "UID=dba;PWD=sql;DBF=/Users/user/Names.db"
```

If the location of your database is different than this, make sure to update the DBF option.

### Creating a MobiLink synchronization model

In order for MobiLink to perform the synchronization, you need to configure the synchronization scripts. To make configuration easier, Sybase Central provides script templates for many common forms of synchronization.

In Sybase Central:

1. Under the **Tools** menu, select **MobiLink 12 -> New Project**.
2. Name the project **NamesProject** and click **Next**.
3. Check **Add a consolidated database to the project**. Provide a **Database display name** of **NamesCondb**. Provide a connection string of 'UID=dba;PWD=sql;DSN=Names'. Note that you can also build this string by clicking the **Edit** button. Click **Next**.
4. Select **Create a new model** and click **Next**.
5. Check **Add a remote schema name to the project**, enter a name, and, with **UltraLite schema** selected, click **Finish**. This remote schema is not actually used, but since the remote schema is on the iPhone, this option is the simplest.
6. A message appears saying that MobiLink has not yet been installed and asks if it should be installed now. Click **Yes**. This will create the tables required by MobiLink in the database, as well as some stored procedures.
7. Enter **NamesModel** as the model name and click **Next**.
8. Check all three checkboxes to acknowledge the MobiLink requirements, and click **Next**. In order for synchronization to function correctly, MobiLink assumes a few things about the primary keys of the tables. Since the Names application already abides by these assumptions, no changes are required.
9. Choose the **NamesCondb** consolidated database and click **Next**.
10. Choose **No, create a new remote database schema** and click **Next**.
11. Check the **Names** table from the list of tables and click **Next**. Any other tables listed are used by MobiLink behind the scenes and can safely be ignored for synchronization.
12. For the **Download type**, choose **Timestamp-based download**. This option provides a good default implementation of synchronization by only synchronizing changes since the last sync. This option will save bandwidth for the iPhone since no unnecessary data will be transferred.
13. Since all other settings should be kept as their defaults, click **Finish**.

If you would like to see the other options available as MobiLink templates, you can click through all of the steps in the wizard.

### Deploying the MobiLink synchronization model

Now that the synchronization model has been created, it should be visible in the left- hand side **Folders** view. If the **Folders** view is not showing, make sure the menu **View -> Folders** is checked.

To deploy the synchronization model:

1. Control-click the **NamesModel** in the **Folder** view and select **Deploy....**
2. Uncheck **Remote database and synchronization client** leaving **Consolidated database** and **MobiLink server** checked and click **Next**.

3. Check **Save changes to the following SQL file** and **Connect to the consolidated database to directly apply the changes** with **NamesCondb** selected in the list. Click **Next**.
4. Accept the creation of the new directory if prompted.
5. Enter a MobiLink user and password. Note that MobiLink users are distinct from SQL Anywhere database users. You should use a different username than **dba**. Suggest 'user' and 'password' respectively. These will be used later in the tutorial.
6. Ensure that **Register this user in the consolidated database for MobiLink authentication** is checked and click **Finish**.
7. Accept the creation of the new directory if prompted.
8. Once the deployment window shows the deployment completing, dismiss it by clicking **Close**.

### Starting the MobiLink server

Now that the synchronization model has been deployed, the consolidated database contains all the information required for MobiLink to function. The deployment also created scripts for launching the MobiLink server:

1. Open a Terminal.
2. Navigate the Terminal session to where deployment saved the launch script. By default this should be `~/NamesProject/NamesModel/mlsrv`.
3. Make the script executable:

```
chmod u+x NamesModel_mlsrv.sh
```

4. Start the MobiLink server:

```
./NamesModel_mlsrv.sh "DSN=Names"
```

### Adding synchronization to the iPhone application

You now have a consolidated database with a MobiLink server running. In order for the application to be able to synchronize to it, you first have to enable synchronization. In the **openConnection** method in the **DataAccess** class, after the **if-block** that contains the database creation add the following line:

```
ULDatabaseManager::EnableTcpipSynchronization();
```

Now that the database is expecting to be synchronized, you can add the following method to the **DataAccess** class. Also add the method signature to the header file.

```
- (void)synchronize {
    NSString *          result = nil;
    ul_sync_info        info;

    // Initialize the sync info struct
    connection->InitSyncInfo(&info);

    // Set the sync parameters
    info.user_name = (char*)"user";    // Set to your username
```

```
info.password = (char*)"password"; // Set to your password
info.version = (char*)"NamesModel";
info.stream = "tcpip";
info.stream_parms = (char*)"host=localhost";

// Display the network activity indicator in the status bar
[[UIApplication sharedApplication]
 setNetworkActivityIndicatorVisible:YES];

// Sync and get the result
if (connection->Synchronize(&info)) {
    result = @"Sync was successful.";
} else {
    // Get the error message and log it.
    char    errorMsg[80];

    connection->GetLastError()->GetString(errorMsg, 80);
    NSLog(@"Sync failed: %s", errorMsg);
    result = [NSString stringWithFormat:@"Sync failed: %s", errorMsg];
}

// Stop showing the activity indicator
[[UIApplication sharedApplication]
 setNetworkActivityIndicatorVisible:NO];

[[[UIAlertView alloc]
 initWithTitle:@"Synchronization"
 message:result
 delegate:nil
 cancelButtonTitle:nil
 otherButtonTitles:@"OK", nil] autorelease] show];
}
```

This is the simplest synchronization a client can perform. In an enterprise application, you would likely want to set a callback method to get the progress of the synchronization and perform the synchronization on a separate thread to not block the application.

Also, in the current implementation, synchronization is done on the main event thread. It is not recommended to block the main thread this way. In the next lesson you will use a separate thread to do the synchronization and add a callback method to observe the sync and display the progress.

To let the user choose when to synchronize, add a button to the navigation bar. Pressing the button will call the following method in **RootViewController**:

```
- (void)sync {
    [[DataAccess sharedInstance] synchronize];
    [self.tableView reloadData];
}
```

To create the button, add the following to the **RootViewController**'s **viewDidLoad** method:

```
// Little button with the refresh sign on the left in the nav bar
self.navigationItem.leftBarButtonItem =
[[UIBarButtonItem alloc]
 initWithBarButtonSystemItem:UIBarButtonSystemItemRefresh
 target:self
 action:@selector(sync)];
```

You can now build and run the application. Whenever you press the refresh button, the database on the iPhone is synchronized with the consolidated SQL Anywhere database.

## Lesson 7: Adding a progress display

In the previous lesson, you added rudimentary synchronization which was done on the main thread. Blocking the main thread in such a way is not recommended. In this lesson you will move the synchronization to a background thread, and add a synchronization observing method to update a progress display.

### Creating the progress toolbar

To show the progress of the sync, you will use a progress bar with a label that will be placed in the bottom navigation toolbar, similar to the look of the **Mail** application when downloading new messages. To recreate this look, you must create a custom view:

1. Control-click the **Classes** folder in the **Groups & Files** pane. Select **New File** from the **Add** menu.
2. Under **Cocoa Touch Class**, choose **UIViewController subclass**.
3. Uncheck **UITableViewController subclass**.
4. Check **With XIB for user interface**.
5. Click **Next**.
6. Name the file: *ProgressToolbarViewController.m* and place it in the *Classes* subdirectory. Check the box to have a header file created.
7. Click **Finish**.
8. Move the *ProgressToolbarViewController.xib* file to the *Resources* directory.

Before creating the layout in **Interface Builder**, add the following instance variables to the **ProgressToolbarViewController**:

```
@interface ProgressToolbarViewController : UIViewController {
    IBOutlet UILabel      *label;
    IBOutlet UIProgressView *progressBar;
}

@property (readonly) IBOutlet UILabel      *label;
@property (readonly) IBOutlet UIProgressView *progressBar;
@end
```

Synthesize the properties in the implementation and add a release call in the **dealloc** method. Open the **ProgressToolbarViewController** by double-clicking *ProgressToolbarViewController.xib* in the Xcode *Resources* folder. Since this view will be displayed in a toolbar, you must size it appropriately, and set its background properties:

1. In the **Document** window (Command-0), select the **View** object.
2. In the **Attribute Inspector** (Command-1), set the simulated status bar to **None**.
3. Set the **Background opacity** to 0%
4. Uncheck the **Opaque** setting.
5. In the **Size Inspector** (Command-3), set the width to 232 and the height to 44.

#### To add the progress view

1. Click and drag a **UIProgressView** from the **Library** to the **View**.
2. In the **Size Inspector** (Command-3), set the position of the progress view to 26, 29.
3. Set the width of the progress view to 186.
4. In the **Attributes Inspector** (Command-1), set the style to **Bar** and the progress to zero.

#### To add the label

1. Click and drag a **UILabel** from the library to the view.
2. In the **Size Inspector** (Command-3), set the position of the label to 14, 5.
3. Set the size of the label to 210, 16.
4. In the **Attribute Inspector** (Command-1), set the text to **Sync Progress**.
5. Set the layout alignment to center.
6. Set the font to Helvetica bold, size 12.
7. Set the text color to white.
8. Set the shadow color to RGB: (103, 114, 130), with 100% opacity.

#### Connect the new label and progress view to the outlets

1. Select the **File's Owner** in the **Document** window.
2. In the **Connectors Inspector** (Command-2), link the label outlet to the **UILabel** you created in the last steps.
3. Link the progress outlet to the **UIProgressView** you created in the last steps.
4. Save the XIB file and close **Interface Builder**.

This view will be added to the **RootViewController**'s toolbar. However, the **DataAccess** object will also use a reference to it to display the progress of the synchronization. Add the following instance variable:

```
ProgressToolBarViewController * progressToolBar;
```

and add the property to the **DataAccess** class:

```
@property (retain, readwrite) IBOutlet ProgressToolBarViewController *
progressToolBar;
```

Import the **ProgressToolBarViewController** header and to synthesize the property in the implementation. To actually add the progress view to the toolbar, add the following to the **RootViewController**'s **viewDidLoad** method:

```
// Create progress display
ProgressToolBarViewController * progress =
[[ProgressToolBarViewController alloc]
 initWithNibName:@"ProgressToolBarViewController"
 bundle:nil];

// Register the toolbar with the DataAccess
[[DataAccess sharedInstance] setProgressToolBar:progress];

// Setup UIBarButtonItems
UIBarButtonItem * space =
[[UIBarButtonItem alloc]
 initWithBarButtonSystemItem:UIBarButtonSystemItemFlexibleSpace
 target:nil
 action:nil];
UIBarButtonItem * progressButtonItem =
[[UIBarButtonItem alloc] initWithCustomView:progress.view];

// Put them in the toolbar
self.toolbarItems =
[NSArray arrayWithObjects:space, progressButtonItem, space, nil];
[space release];
[progressButtonItem release];
```

The **RootViewController** now has a progress view in its toolbar, even though the toolbar is hidden. In the next section, you will move synchronization to a background thread and display the progress toolbar during the sync.

### Performing synchronization in a background thread

So far, everything the application does is performed on the main thread of the application. Since this is also the thread used to draw the interface and process user event such as touches, blocking it is not recommended.

The flow of the application during the sync will be as follows:

1. The user clicks the sync button in **RootViewController**. The toolbar progress view is displayed.
2. **RootViewController** initiates the sync on a detached thread, passing itself as an argument to the `synchronize` function. The main thread continues with the sync running in another thread.
3. The synchronization function updates the user interface using **performSelectorOnMainThread** to update the progress display throughout the synchronization.

4. When synchronization is complete, the background thread displays an alert box showing the result of the sync, with **RootViewController** set as the delegate to the alert so that it knows when the alert is dismissed.
5. Once **RootViewController** is told the alert box is dismissed, the toolbar progress view is hidden.

Since the **RootViewController** will need to implement the **UIAlertViewDelegate** protocol to handle the alert view dismissing, change its header file to the following:

```
@interface RootViewController : UITableViewController <UIAlertViewDelegate> {  
}  
  
// Displays the screen to add a name.  
- (void)showAddNameScreen;  
  
@end
```

The only method from the **UIAlertViewDelegate** protocol that the **RootViewController** needs to implement is the **alertView:clickedButtonAtIndex:** method. This method is called when the user presses a button of the **UIAlertView**. Since the view only has a single button, you don't need to inspect which button was pressed.

```
- (void)alertView:(UIAlertView *)alertView  
clickedButtonAtIndex:(NSInteger)buttonIndex {  
    NSLog(@"User dismissed sync alert, refreshing table.");  
    [self.tableView reloadData];  
    [self.navigationController setToolbarHidden:YES animated: YES];  
}
```

Change the sync method to do the synchronization call on a separate thread by using **detachNewThreadSelector:**

```
- (void)sync {  
    [self.navigationController setToolbarHidden:NO animated: YES];  
    [NSThread detachNewThreadSelector:@selector(synchronize:  
        toTarget:[DataAccess sharedInstance]  
        withObject:self];  
}
```

Now that the synchronization is on a separate thread, you need to change a few things. As you might have noticed in the **RootViewController**'s sync method, **DataAccess**' synchronize method is being passed a parameter. Change its signature to the following:

```
- (void)synchronize:(id<UIAlertViewDelegate>)sender;
```

Since the synchronization method uses auto-released object instances, we also need to create an **NSAutoreleasePool** and release it at the end of the synchronization:

```
- (void)synchronize:(id<UIAlertViewDelegate>)sender {  
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];  
    // Rest of synchronize method...  
  
    [pool release];  
}
```



Still in the `synchronize` method, you also need to set the passed-in **RootViewController** as the alert delegate. To do this update the previously create **UIAlertView** and change the delegate from `nil` to `sender`:

```
// Set RootViewController as the alert delegate
if (sender != nil) {
    NSLog(@"Showing Alert with sync result.");
    [[[UIAlertView alloc]
     initWithTitle:@"Synchronization"
     message:result
     delegate:sender // changed from nil to sender
     cancelButtonTitle:nil
     otherButtonTitles:@"OK", nil] autorelease] show];
} else {
    NSLog(@"Not showing alert since sender was nil.");
}
```

You now need to create the synchronization callback which will update the progress bar to the appropriate completion and display the correct message.

However, since the synchronization callback is executed on a non-main thread (the synchronization thread created by the **RootViewController**) you will have to use **performSelectorOnMainThread** to update the user interface. This method only allows the passing of a single parameter, so in order to pass the progress, as well as the message, create a plain object named **UpdateInfo** with two properties, a float, and an **NSString**:

```
@interface UpdateInfo : NSObject {
    NSString *    message;
    float        progress;
}

// Message to display in the progress view
@property (readonly) NSString *    message;

// Progress of the sync [0.0-1.0]
@property (readonly) float        progress;

// Preferred initializer
- (id)initWithMessage:(NSString*) message andProgress:(float)progress;

@end
```

The implementation is simply a single constructor that takes both parameters:

```
@implementation UpdateInfo

@synthesize message;
@synthesize progress;

- (id)initWithMessage:(NSString*) msg andProgress:(float) syncProgress {
    if (self = [super init]) {
        message = msg;
        progress = syncProgress;
    }

    return self;
}

@end
```

Now that you can bundle all the information needed by the user interface updating method, you can define it in the **DataAccess** class:

```
- (void)updateSyncProgress:(UpdateInfo *)info {
    progressBar.label.text = info.message;
    progressBar.progressBar.progress = info.progress;
}
```

Now import *UpdateInfo.h* in *DataAccess.h*. You can also define the callback method in *DataAccess.mm*:

```
static void UL_CALLBACK_FN progressCallback(ul_synch_status * status) {
    // Sync information for the GUI
    float percentDone = 0.0;
    NSString * message;

    // Note: percentDone is approximate.
    switch (status->state) {
        case UL_SYNCH_STATE_STARTING:
            percentDone = 0;
            message = @"Starting Sync";
            break;
        case UL_SYNCH_STATE_CONNECTING:
            percentDone = 5;
            message = @"Connecting to Server";
            break;
        case UL_SYNCH_STATE_SENDING_HEADER:
            percentDone = 10;
            message = @"Sending Sync Header";
            break;
        case UL_SYNCH_STATE_SENDING_TABLE:
            percentDone =
                10 + 25 *
                (status->sync_table_index / status->sync_table_count);
            message =
                [NSString stringWithFormat:@"Sending Table %s: %d of %d",
                 status->table_name,
                 status->sync_table_index,
                 status->sync_table_count];
            break;
        case UL_SYNCH_STATE_SENDING_DATA:
            percentDone =
                10 + 25 *
                (status->sync_table_index / status->sync_table_count);
            message = @"Sending Name Changes";
            break;
        case UL_SYNCH_STATE_FINISHING_UPLOAD:
        case UL_SYNCH_STATE_RECEIVING_UPLOAD_ACK:
            percentDone = 50;
            message = @"Finishing Upload";
            break;
        case UL_SYNCH_STATE_RECEIVING_TABLE:
        case UL_SYNCH_STATE_RECEIVING_DATA:
            percentDone =
                50 + 25 *
                (status->sync_table_index / status->sync_table_count);
            message =
                [NSString
                 stringWithFormat:@"Receiving Table %s: %d of %d",
                 status->table_name,
                 status->sync_table_index,
                 status->sync_table_count];
            break;
        case UL_SYNCH_STATE_COMMITTING_DOWNLOAD:
    }
```

```

    case UL_SYNCH_STATE_SENDING_DOWNLOAD_ACK:
        percentDone = 80;
        message = @"Committing Downloaded Updates";
        break;
    case UL_SYNCH_STATE_DISCONNECTING:
        percentDone = 90;
        message = @"Disconnecting from Server";
        break;
    case UL_SYNCH_STATE_DONE:
        percentDone = 100;
        message = @"Finished Sync";
        break;
    case UL_SYNCH_STATE_ERROR:
        percentDone = 95;
        message = @"Error During Sync";
        break;
    case UL_SYNCH_STATE_ROLLING_BACK_DOWNLOAD:
        percentDone = 100;
        message = @"Rolling Back due to Error";
        break;
    default:
        percentDone = 100;
        NSLog(@"Unknown sync state: '%d'", status->state);
        break;
}

// Wrap the GUI info in an object and have the main thread update
UpdateInfo * info = [[[UpdateInfo alloc]
                    initWithMessage:message
                    andProgress:percentDone / 100]
                    autorelease];
[[DataAccess sharedInstance]
 performSelectorOnMainThread:@selector(updateSyncProgress:)
 withObject:info waitUntilDone:YES

```

With everything in place, you can now set the sync observer to the callback method in the synchronize method:

```

// Set the sync parameters
info.user_name = (char*)"user"; // Set to your username
info.password = (char*)"password"; // Set to your password
info.version = (char*)"NamesModel";
info.stream = "tcpip";
info.stream_parms = (char*)"host=localhost";
info.observer = progressCallback; // Add this line

```

## Conclusion

The Names application is now a fully functioning iPhone application with an UltraLite database which can monitor synchronization with a MobiLink server. Using this example application as a starting point, you can use the concepts shown to make applications that make use of UltraLite and MobiLink technologies.

---

---

# Index

## Symbols

#define

UltraLite applications, 69

16-bit signed integer UltraLite embedded SQL data type  
about, 34

32-bit signed integer UltraLite embedded SQL data type  
about, 34

4-byte floating-point UltraLite embedded SQL data  
type  
about, 35

8-byte floating-point UltraLite embedded SQL data  
type  
about, 35

## A

accessing data

UltraLite C++ API, 17

ActiveSync

class names, 61

UltraLite MFC requirements, 64

UltraLite synchronization for Windows Mobile, 63

UltraLite versions for Windows Mobile, 63

UltraLite Windows Mobile applications, 63

WindowProc function, 64

AES encryption algorithm

UltraLite embedded SQL databases, 48

AfterLast method

ULResultSet class [UltraLite C/C++ API], 181

AppendByteChunk method

ULResultSet class [UltraLite C/C++ API], 182

AppendParameterByteChunk method

ULPreparedStatement class [UltraLite C/C++  
API], 169

AppendParameterStringChunk method

ULPreparedStatement class [UltraLite C/C++  
API], 169

AppendStringChunk method

ULResultSet class [UltraLite C/C++ API], 183

applications

building UltraLite embedded SQL, 56

compiling UltraLite embedded SQL, 56

preprocessing UltraLite embedded SQL, 56

writing UltraLite embedded SQL, 29

AutoCommit mode

UltraLite C++ development, 22

## B

BeforeFirst method

ULResultSet class [UltraLite C/C++ API], 186

binary UltraLite embedded SQL data type  
about, 36

bugs

providing feedback, xiv

build processes

embedded SQL applications, 56

UltraLite embedded SQL applications, 56

building

UltraLite embedded SQL applications, 56

## C

C++ APIs

(*see also* UltraLite C/C++ API)

C++ applications

(*see also* UltraLite C/C++)

CancelGetNotification method

ULConnection class [UltraLite C/C++ API], 95

canceling monitoring

in UltraLite embedded SQL, 52

canceling synchronization

in UltraLite embedded SQL, 52

casting

UltraLite C++ API data types in, 19

ChangeEncryptionKey method

ULConnection class [UltraLite C/C++ API], 96

changeEncryptionKey method

UltraLite embedded SQL, 49

character string UltraLite embedded SQL data type

fixed length, 35

variable length, 35

Checkpoint method

ULConnection class [UltraLite C/C++ API], 97

class names

ActiveSync synchronization, 61

Clear method

ULError class [UltraLite C/C++ API], 156

Close method

ULConnection class [UltraLite C/C++ API], 98

ULDatabaseSchema class [UltraLite C/C++ API],  
151

ULIndexSchema class [UltraLite C/C++ API], 162

- ULPreparedStatement class [UltraLite C/C++ API], 171
  - ULResultSet class [UltraLite C/C++ API], 186
  - ULTableSchema class [UltraLite C/C++ API], 245
  - CLOSE statement
    - UltraLite embedded SQL, 44
  - columns
    - UltraLite C++ API modification of values , 19
    - UltraLite C++ API retrieval of values , 19
  - command prompts
    - conventions, xiii
    - curly braces, xiii
    - environment variables, xiii
    - parentheses, xiii
    - quotes, xiii
    - semicolons, xiii
  - command shells
    - conventions, xiii
    - curly braces, xiii
    - environment variables, xiii
    - parentheses, xiii
    - quotes, xiii
  - Commit method
    - ULConnection class [UltraLite C/C++ API], 98
  - commit method
    - UltraLite C++ transactions, 22
  - committing
    - UltraLite C++ transactions, 22
    - UltraLite changes with embedded SQL, 41
  - communications errors
    - UltraLite embedded SQL, 52
  - compile options
    - UltraLite applications for Windows Mobile, 59
  - compiler directives
    - UltraLite applications, 69
    - UNDER\_CE, 70
  - compiler options
    - UltraLite C++ development, 26
  - compilers
    - UltraLite applications for Windows Mobile, 59
  - compiling
    - UltraLite applications for Windows Mobile, 59
    - UltraLite embedded SQL applications, 56
  - configuring
    - development tools for UltraLite embedded SQL, 57
  - CONNECT statement
    - UltraLite embedded SQL, 32
  - connecting
    - UltraLite C++ databases, 8
  - Connection object
    - UltraLite C++, 8
  - connections
    - UltraLite embedded SQL, 32
  - conventions
    - command prompts, xiii
    - command shells, xiii
    - documentation, xi
    - file names in documentation, xii
    - operating systems, xi
    - Unix , xi
    - Windows, xi
    - Windows CE, xi
    - Windows Mobile, xi
  - CountUploadRows method
    - ULConnection class [UltraLite C/C++ API], 98
  - CreateDatabase method
    - ULDatabaseManager class [UltraLite C/C++ API], 141
  - CreateNotificationQueue method
    - ULConnection class [UltraLite C/C++ API], 99
  - cursors
    - UltraLite fetching multiple rows, 44
    - UltraLite order of rows, 46
    - UltraLite positioning, 45
    - UltraLite positioning after updates, 47
    - UltraLite repositioning, 46
  - CustDB application
    - UltraLite building for Windows Mobile, 59
- ## D
- data manipulation
    - UltraLite C++ API, 17
    - UltraLite C++ API with SQL, 10
  - data types
    - UltraLite C++ API accessing and casting, 19
    - UltraLite embedded SQL, 34
  - database files
    - UltraLite encrypting and obfuscating (embedded SQL), 48
    - UltraLite for Windows Mobile, 60
  - database schemas
    - UltraLite C++ API access, 23
  - DatabaseManager object
    - UltraLite C++, 8
  - db\_fini method [UltraLite Embedded SQL API]

---

- description, 259
- db\_init method [UltraLite Embedded SQL API]
  - description, 259
- DCX
  - about, xi
- decimal UltraLite embedded SQL data type
  - about, 35
- DECL\_BINARY macro
  - UltraLite embedded SQL, 34
- DECL\_DATETIME macro
  - UltraLite embedded SQL, 34
- DECL\_DECIMAL macro
  - UltraLite embedded SQL, 34
- DECL\_FIXCHAR macro
  - UltraLite embedded SQL, 34
- DECL\_VARCHAR macro
  - UltraLite embedded SQL, 34
- declaration section
  - UltraLite embedded SQL declaration of, 33
- DECLARE statement
  - UltraLite embedded SQL, 44
- DeclareEvent method
  - ULConnection class [UltraLite C/C++ API], 101
- declaring
  - UltraLite host variables, 33
- Delete method
  - ULResultSet class [UltraLite C/C++ API], 187
- DeleteAllRows method
  - ULTable class [UltraLite C/C++ API], 238
- DeleteNamed method
  - ULResultSet class [UltraLite C/C++ API], 187
- deleting
  - UltraLite C++ API table rows, 22
- dependencies
  - UltraLite embedded SQL, 57
- deploying
  - UltraLite for Windows Mobile, 61
- DestroyNotificationQueue method
  - ULConnection class [UltraLite C/C++ API], 102
- developer centers
  - finding out more and requesting technical support, xv
- developer community
  - newsgroups, xiv
- development
  - UltraLite C++, 7
- development platforms
  - UltraLite C++, 2
- development process
  - UltraLite embedded SQL, 1
- development tools
  - UltraLite embedded SQL, 57
- directives
  - UltraLite applications, 69
- DML
  - UltraLite C++, 11
- DocCommentXchange (DCX)
  - about, xi
- documentation
  - conventions, xi
  - SQL Anywhere, xi
- DropDatabase method
  - ULDatabaseManager class [UltraLite C/C++ API], 142
- DT\_LONGBINARY UltraLite embedded SQL data type
  - about, 37
- DT\_LONGVARCHAR UltraLite embedded SQL data type
  - about, 36
- dynamic libraries
  - UltraLite C++ applications, 26

**E**

- embedded SQL
  - (*see also* UltraLite embedded SQL)
- embedded SQL library functions (*see* UltraLite embedded SQL library functions)
- emulator
  - UltraLite for Windows Mobile, 61
- EnableAesDBEncryption method
  - ULDatabaseManager class [UltraLite C/C++ API], 143
- EnableAesFipsDBEncryption method
  - ULDatabaseManager class [UltraLite C/C++ API], 143
- EnableEccE2ee method
  - ULDatabaseManager class [UltraLite C/C++ API], 144
- EnableEccSyncEncryption method
  - ULDatabaseManager class [UltraLite C/C++ API], 144
- EnableHttpsSynchronization method
  - ULDatabaseManager class [UltraLite C/C++ API], 144

---

- EnableHttpSynchronization method
    - ULDatabaseManager class [UltraLite C/C++ API], 144
  - EnableRsaE2ee method
    - ULDatabaseManager class [UltraLite C/C++ API], 144
  - EnableRsaFipsE2ee method
    - ULDatabaseManager class [UltraLite C/C++ API], 144
  - EnableRsaFipsSyncEncryption method
    - ULDatabaseManager class [UltraLite C/C++ API], 145
  - EnableRsaSyncEncryption method
    - ULDatabaseManager class [UltraLite C/C++ API], 145
  - EnableTcpipSynchronization method
    - ULDatabaseManager class [UltraLite C/C++ API], 145
  - EnableTlsSynchronization method
    - ULDatabaseManager class [UltraLite C/C++ API], 145
  - EnableZlibSyncCompression method
    - ULDatabaseManager class [UltraLite C/C++ API], 145
  - encryption
    - changing keys in UltraLite embedded SQL, 49
    - UltraLite C++ development, 25
    - UltraLite databases using embedded SQL, 48
    - UltraLite embedded SQL databases, 48
  - environment variables
    - command prompts, xiii
    - command shells, xiii
    - INCLUDE, 329
  - error handling
    - UltraLite C++, 24
  - errors
    - UltraLite C++ API handling , 24
    - UltraLite codes, 31
    - UltraLite embedded SQL communications errors, 52
    - UltraLite SQLCODE, 31
    - UltraLite sqlcode SQLCA field, 31
  - EXEC SQL
    - UltraLite embedded SQL development, 30
  - ExecuteQuery method
    - ULPreparedStatement class [UltraLite C/C++ API], 171
  - ExecuteScalar method
    - ULConnection class [UltraLite C/C++ API], 103
  - ExecuteScalarV method
    - ULConnection class [UltraLite C/C++ API], 106
  - ExecuteStatement method
    - ULConnection class [UltraLite C/C++ API], 109
    - ULPreparedStatement class [UltraLite C/C++ API], 171
- ## F
- feedback
    - documentation, xiv
    - providing, xiv
    - reporting an error, xiv
    - requesting an update, xiv
  - FETCH statement
    - UltraLite embedded SQL multi-row queries, 44
    - UltraLite embedded SQL single-row queries, 43
  - fetching
    - UltraLite embedded SQL, 43
  - Find method
    - ULTable class [UltraLite C/C++ API], 239
  - find methods
    - UltraLite C++, 20
  - find mode
    - UltraLite C++, 18
  - FindBegin method
    - ULTable class [UltraLite C/C++ API], 239
  - FindFirst method
    - ULTable class [UltraLite C/C++ API], 239
  - finding out more and requesting technical assistance
    - technical support, xiv
  - FindLast method
    - ULTable class [UltraLite C/C++ API], 240
  - FindNext method
    - ULTable class [UltraLite C/C++ API], 240
  - FindPrevious method
    - ULTable class [UltraLite C/C++ API], 240
  - Fini method
    - ULDatabaseManager class [UltraLite C/C++ API], 146
  - First method
    - ULResultSet class [UltraLite C/C++ API], 188
  - functions
    - UltraLite embedded SQL, 259
- ## G
- GetBinary method



---

ULResultSet class [UltraLite C/C++ API], 188  
 GetBinaryLength method  
     ULResultSet class [UltraLite C/C++ API], 190  
 GetByteChunk method  
     ULResultSet class [UltraLite C/C++ API], 191  
 GetChildObjectCount method  
     ULConnection class [UltraLite C/C++ API], 110  
 GetColumnCount method  
     ULIndexSchema class [UltraLite C/C++ API], 162  
     ULResultSetSchema class [UltraLite C/C++ API], 230  
 GetColumnDefault method  
     ULTableSchema class [UltraLite C/C++ API], 245  
 GetColumnDefaultType method  
     ULTableSchema class [UltraLite C/C++ API], 246  
 GetColumnDefaultW2 method  
     ULTableSchema class [UltraLite C/C++ API], 246  
 GetColumnID method  
     ULResultSetSchema class [UltraLite C/C++ API], 231  
 GetColumnName method  
     ULIndexSchema class [UltraLite C/C++ API], 163  
     ULResultSetSchema class [UltraLite C/C++ API], 232  
 GetColumnNameW2 method  
     ULIndexSchema class [UltraLite C/C++ API], 163  
     ULResultSetSchema class [UltraLite C/C++ API], 232  
 GetColumnPrecision method  
     ULResultSetSchema class [UltraLite C/C++ API], 233  
 GetColumnScale method  
     ULResultSetSchema class [UltraLite C/C++ API], 233  
 GetColumnSize method  
     ULResultSetSchema class [UltraLite C/C++ API], 233  
 GetColumnSQLType method  
     ULResultSetSchema class [UltraLite C/C++ API], 234  
 GetColumnType method  
     ULResultSetSchema class [UltraLite C/C++ API], 234  
 GetConnection method  
     ULDatabaseSchema class [UltraLite C/C++ API], 152  
     ULIndexSchema class [UltraLite C/C++ API], 163  
     ULPreparedStatement class [UltraLite C/C++ API], 171  
     ULResultSet class [UltraLite C/C++ API], 193  
     ULResultSetSchema class [UltraLite C/C++ API], 235  
 GetDatabaseProperty method  
     ULConnection class [UltraLite C/C++ API], 110  
 GetDatabasePropertyInt method  
     ULConnection class [UltraLite C/C++ API], 111  
 GetDatabaseSchema method  
     ULConnection class [UltraLite C/C++ API], 112  
 GetDateTime method  
     ULResultSet class [UltraLite C/C++ API], 193  
 GetDouble method  
     ULResultSet class [UltraLite C/C++ API], 195  
 GetErrorInfo method  
     ULError class [UltraLite C/C++ API], 156  
 GetFloat method  
     ULResultSet class [UltraLite C/C++ API], 196  
 GetGlobalAutoincPartitionSize method  
     ULTableSchema class [UltraLite C/C++ API], 246  
 GetGuid method  
     ULResultSet class [UltraLite C/C++ API], 197  
 GetIndexColumnID method  
     ULIndexSchema class [UltraLite C/C++ API], 164  
 GetIndexCount method  
     ULTableSchema class [UltraLite C/C++ API], 247  
 GetIndexFlags method  
     ULIndexSchema class [UltraLite C/C++ API], 164  
 GetIndexSchema method  
     ULTableSchema class [UltraLite C/C++ API], 247  
 GetInt method  
     ULResultSet class [UltraLite C/C++ API], 199  
 GetIntWithType method  
     ULResultSet class [UltraLite C/C++ API], 200  
 GetLastDownloadTime method  
     ULConnection class [UltraLite C/C++ API], 113  
 GetLastError method  
     ULConnection class [UltraLite C/C++ API], 114  
 GetLastIdentity method  
     ULConnection class [UltraLite C/C++ API], 114  
 GetName method  
     ULIndexSchema class [UltraLite C/C++ API], 165  
     ULTableSchema class [UltraLite C/C++ API], 248  
 GetNameW2 method  
     ULIndexSchema class [UltraLite C/C++ API], 165  
     ULTableSchema class [UltraLite C/C++ API], 248  
 GetNextIndex method

- ULTableSchema class [UltraLite C/C++ API], 249
- GetNextPublication method
  - ULDatabaseSchema class [UltraLite C/C++ API], 152
- GetNextPublicationW2 method
  - ULDatabaseSchema class [UltraLite C/C++ API], 152
- GetNextTable method
  - ULDatabaseSchema class [UltraLite C/C++ API], 153
- GetNotification method
  - ULConnection class [UltraLite C/C++ API], 114
- GetNotificationParameter method
  - ULConnection class [UltraLite C/C++ API], 116
- GetOptimalIndex method
  - ULTableSchema class [UltraLite C/C++ API], 249
- GetOptimalIndexW2 method
  - ULTableSchema class [UltraLite C/C++ API], 249
- GetParameter method
  - ULError class [UltraLite C/C++ API], 157
- GetParameterCount method
  - ULError class [UltraLite C/C++ API], 158
  - ULPreparedStatement class [UltraLite C/C++ API], 171
- GetParameterType method
  - ULPreparedStatement class [UltraLite C/C++ API], 172
- GetPlan method
  - ULPreparedStatement class [UltraLite C/C++ API], 172
- GetPrimaryKey method
  - ULTableSchema class [UltraLite C/C++ API], 250
- GetPublicationCount method
  - ULDatabaseSchema class [UltraLite C/C++ API], 153
- GetPublicationPredicate method
  - ULTableSchema class [UltraLite C/C++ API], 250
- GetReferencedIndexName method
  - ULIndexSchema class [UltraLite C/C++ API], 165
- GetReferencedIndexNameW2 method
  - ULIndexSchema class [UltraLite C/C++ API], 166
- GetReferencedTableName method
  - ULIndexSchema class [UltraLite C/C++ API], 166
- GetReferencedTableNameW2 method
  - ULIndexSchema class [UltraLite C/C++ API], 166
- GetResultSetSchema method
  - ULPreparedStatement class [UltraLite C/C++ API], 174
- ULResultSet class [UltraLite C/C++ API], 203
- GetRowCount method
  - ULResultSet class [UltraLite C/C++ API], 203
- GetRowsAffectedCount method
  - ULPreparedStatement class [UltraLite C/C++ API], 174
- GetSqlca method
  - ULConnection class [UltraLite C/C++ API], 117
- GetSQLCode method
  - ULError class [UltraLite C/C++ API], 158
- GetSQLCount method
  - ULError class [UltraLite C/C++ API], 158
- GetState method
  - ULResultSet class [UltraLite C/C++ API], 203
- GetString method
  - ULError class [UltraLite C/C++ API], 159
  - ULResultSet class [UltraLite C/C++ API], 203
- GetStringChunk method
  - ULResultSet class [UltraLite C/C++ API], 206
- GetStringLength method
  - ULResultSet class [UltraLite C/C++ API], 209
- GetSyncResult method
  - ULConnection class [UltraLite C/C++ API], 118
- GetTableCount method
  - ULDatabaseSchema class [UltraLite C/C++ API], 154
- GetTableName method
  - ULIndexSchema class [UltraLite C/C++ API], 167
- GetTableNameW2 method
  - ULIndexSchema class [UltraLite C/C++ API], 167
- GetTableSchema method
  - ULDatabaseSchema class [UltraLite C/C++ API], 154
  - ULTable class [UltraLite C/C++ API], 241
- GetTableSyncType method
  - ULTableSchema class [UltraLite C/C++ API], 251
- getting help
  - technical support, xiv
- GetURL method
  - ULError class [UltraLite C/C++ API], 160
- GlobalAutoincUsage method
  - ULConnection class [UltraLite C/C++ API], 118
- GrantConnectTo method
  - ULConnection class [UltraLite C/C++ API], 118
- grantConnectTo method
  - UltraLite C++ development, 24

---

## H

### HasResultSet method

- ULPreparedStatement class [UltraLite C/C++ API], 174

### help

- technical support, xiv

### host variables

- UltraLite embedded SQL, 33
- UltraLite embedded SQL expressions, 39
- UltraLite scope, 37
- UltraLite usage, 37

## I

### iAnywhere developer community

- newsgroups, xiv

### import libraries

- UltraLite C++, 26

### INCLUDE statement

- UltraLite SQLCA, 31

### indexes

- UltraLite C++ API schema information in , 23

### indicator variables

- UltraLite embedded SQL, 41
- UltraLite NULL, 42

### Init method

- ULDatabaseManager class [UltraLite C/C++ API], 146

### InitSyncInfo method

- ULConnection class [UltraLite C/C++ API], 119

### InPublication method

- ULTableSchema class [UltraLite C/C++ API], 252

### Insert method

- ULTable class [UltraLite C/C++ API], 241

### insert mode

- UltraLite C++, 18

### InsertBegin method

- ULTable class [UltraLite C/C++ API], 241

### inserting

- UltraLite C++ API table rows, 21

### install-dir

- documentation usage, xii

### installing

- UltraLite Windows Mobile applications, 59

### iPhone

- access databases, 342
- adding data to databases, 345
- adding databases to applications, 342

- compiling UltraLite library for, 339

- creating an application project, 340

- data model class, 342

- dataaccess singleton, 342

- deleting data from databases, 351

- displaying data, 349

- frameworks, 342

- interface builder, 345

- MobiLink synchronization model, 352

- ODBC data source, 352

- open connection, 342

- root view controller setup, 345

- swipe-to-delete, 351

- synchronization, 352

- thread management, 357

- tutorial, 339

- UltraLite C++ development, 26

- UltraLite preprocessor macro, 341

- UltraLite runtime libraries for, 28

- view controller, 345

### iPhone build settings

- UltraLite C++, 7

### iPhone include files

- UltraLite C++, 7

### IsAliased method

- ULResultSetSchema class [UltraLite C/C++ API], 235

### IsColumnDescending method

- ULIndexSchema class [UltraLite C/C++ API], 167

### IsColumnInIndex method

- ULTableSchema class [UltraLite C/C++ API], 252

### IsColumnNullable method

- ULTableSchema class [UltraLite C/C++ API], 254

### IsNull method

- ULResultSet class [UltraLite C/C++ API], 211

### IsOK method

- ULError class [UltraLite C/C++ API], 161

## L

### Last method

- ULResultSet class [UltraLite C/C++ API], 212

### libraries

- UltraLite applications for Windows Mobile, 59

- UltraLite compiling and linking in C++, 26

- UltraLite DLL for Windows Mobile, 61

- UltraLite linking example in C++, 330

### library functions

- UltraLite embedded SQL, 259
  - libulbase.lib
    - UltraLite C++ development, 27
  - libulecc.lib
    - UltraLite C++ development, 27
  - libulrsa.lib
    - UltraLite C++ development, 27
  - libulrt.lib
    - UltraLite C++ development, 27
  - libulrtc.lib
    - UltraLite C++ development, 27
  - linking
    - UltraLite applications for Windows Mobile, 59
    - UltraLite C++ applications, 26
  - Linux
    - UltraLite runtime libraries for, 27
  - Lookup method
    - ULTable class [UltraLite C/C++ API], 242
  - lookup methods
    - UltraLite C++, 20
  - lookup mode
    - UltraLite C++, 18
  - LookupBackward method
    - ULTable class [UltraLite C/C++ API], 242
  - LookupBegin method
    - ULTable class [UltraLite C/C++ API], 242
  - LookupForward method
    - ULTable class [UltraLite C/C++ API], 243
- M**
- Mac OS X
    - UltraLite C++ development, 26
    - UltraLite runtime libraries for, 28
  - Mac OS X include files
    - UltraLite C++, 7
  - macros
    - UL\_TEXT, 69
    - UL\_USE\_DLL, 69
    - UltraLite applications, 69
  - makefiles
    - UltraLite embedded SQL, 57
  - managing
    - UltraLite C++ transactions, 22
  - MFC
    - UltraLite applications ActiveSync requirements, 64
  - MFC applications
    - UltraLite for Windows Mobile, 61
    - ml\_file\_transfer\_info structure [UltraLite Embedded SQL API]
      - description, 320
    - ml\_file\_transfer\_info\_w structure [UltraLite Embedded SQL API]
      - description, 322
    - ml\_file\_transfer\_status structure [UltraLite Embedded SQL API]
      - description, 325
    - MLFileDownload method [UltraLite Embedded SQL API]
      - description, 266
    - MLFileDownloadW method [UltraLite Embedded SQL API]
      - description, 266
    - mlfiletransfer.h header file
      - UltraLite Embedded SQL API, 259
    - MLFileUpload method [UltraLite Embedded SQL API]
      - description, 267
    - MLFileUploadW method [UltraLite Embedded SQL API]
      - description, 267
    - MLFiniFileTransferInfo method [UltraLite Embedded SQL API]
      - description, 268
    - MLFiniFileTransferInfoW method [UltraLite Embedded SQL API]
      - description, 268
    - mlft\_stream\_error structure [UltraLite Embedded SQL API]
      - description, 326
    - mlft\_stream\_error\_w structure [UltraLite Embedded SQL API]
      - description, 326
    - MLFTEnableEccE2ee method [UltraLite Embedded SQL API]
      - description, 260
    - MLFTEnableEccE2eeW method [UltraLite Embedded SQL API]
      - description, 260
    - MLFTEnableEccEncryption method [UltraLite Embedded SQL API]
      - description, 261
    - MLFTEnableEccEncryptionW method [UltraLite Embedded SQL API]
      - description, 261
    - MLFTEnableRsaE2ee method [UltraLite Embedded SQL API]

- description, 262
- MLFTEnableRsaE2eeW method [UltraLite Embedded SQL API]
  - description, 262
- MLFTEnableRsaEncryption method [UltraLite Embedded SQL API]
  - description, 262
- MLFTEnableRsaEncryptionW method [UltraLite Embedded SQL API]
  - description, 263
- MLFTEnableRsaFipsE2ee method [UltraLite Embedded SQL API]
  - description, 263
- MLFTEnableRsaFipsE2eeW method [UltraLite Embedded SQL API]
  - description, 264
- MLFTEnableRsaFipsEncryption method [UltraLite Embedded SQL API]
  - description, 264
- MLFTEnableRsaFipsEncryptionW method [UltraLite Embedded SQL API]
  - description, 264
- MLFTEnableZlibCompression method [UltraLite Embedded SQL API]
  - description, 265
- MLFTEnableZlibCompressionW method [UltraLite Embedded SQL API]
  - description, 265
- MlInitFileTransferInfo method [UltraLite Embedded SQL API]
  - description, 269
- MlInitFileTransferInfoW method [UltraLite Embedded SQL API]
  - description, 269
- MobiLink synchronization
  - iPhone, 352
- modes
  - UltraLite C++, 18
- multi-row queries
  - UltraLite cursors, 44
- multi-threaded applications
  - UltraLite C++, 10
  - UltraLite embedded SQL, 32

## N

- namespaces
  - UltraLite C++ example, 330

- navigating
  - UltraLite C++ API, 17
- navigating SQL result sets
  - UltraLite C++, 16
- network protocols
  - UltraLite for Windows Mobile, 66
- newsgroups
  - technical support, xiv
- Next method
  - ULResultSet class [UltraLite C/C++ API], 212
- NULL
  - UltraLite indicator variables, 41
- Null terminated string UltraLite embedded SQL data type
  - about, 35
- null-terminated TCHAR character string UltraLite SQL data type
  - about, 35
- null-terminated UNICODE character string UltraLite SQL data type
  - about, 35
- null-terminated WCHAR character string UltraLite SQL data type
  - about, 35
- null-terminated wide character string UltraLite SQL data type
  - about, 35

## O

- obfuscating
  - UltraLite embedded SQL databases, 48
- obfuscation
  - UltraLite C++ development, 25
  - UltraLite databases using embedded SQL, 49
  - UltraLite embedded SQL databases, 48
- observer synchronization parameter
  - UltraLite embedded SQL example, 55
- ODBC data source
  - iPhone, 352
- offsets
  - UltraLite C++ relative, 17
- online books
  - PDF, xi
- OPEN statement
  - UltraLite embedded SQL, 44
- OpenConnection method

- ULDatabaseManager class [UltraLite C/C++ API], 146
- OpenTable method
  - ULConnection class [UltraLite C/C++ API], 120
- operating systems
  - Unix, xi
  - Windows, xi
  - Windows CE, xi
  - Windows Mobile, xi

## P

- packed decimal UltraLite embedded SQL data type
  - about, 35
- passwords
  - UltraLite C++ API authentication in , 24
- PDF
  - documentation, xi
- performance
  - UltraLite using DLL for economical memory use, 59
  - UltraLite using INSERT statements , 52
- permissions
  - UltraLite embedded SQL, 31
- persistent storage
  - UltraLite for Windows Mobile, 60
- platform requirements
  - UltraLite for Windows Mobile, 59
- platforms
  - supported in UltraLite C++, 2
- prepared statements
  - UltraLite C++, 11
- PrepareStatement method
  - ULConnection class [UltraLite C/C++ API], 121
- preprocessing
  - UltraLite embedded SQL applications, 56
  - UltraLite embedded SQL development tool settings, 57
- Previous method
  - ULResultSet class [UltraLite C/C++ API], 213
- program structure
  - UltraLite embedded SQL, 30
- protocols
  - UltraLite for Windows Mobile, 66

## Q

- queries
  - UltraLite embedded SQL multi-row queries, 44

- UltraLite embedded SQL single-row queries, 43

## R

- RegisterForEvent method
  - ULConnection class [UltraLite C/C++ API], 122
- Relative method
  - ULResultSet class [UltraLite C/C++ API], 213
- relative offset
  - UltraLite C++ API, 17
- ResetLastDownloadTime method
  - ULConnection class [UltraLite C/C++ API], 125
- result set schemas
  - UltraLite C++, 16
- result sets
  - UltraLite C++ API schema information in , 23
  - UltraLite C++ navigation of, 16
- RevokeConnectFrom method
  - ULConnection class [UltraLite C/C++ API], 126
- revokeConnectionFrom method
  - UltraLite C++ development, 24
- Rollback method
  - ULConnection class [UltraLite C/C++ API], 126
- rollback method
  - UltraLite C++ transactions, 22
- RollbackPartialDownload method
  - ULConnection class [UltraLite C/C++ API], 127
- rollbacks
  - UltraLite C++ transactions, 22
- rows
  - accessing in UltraLite C++ API tutorial, 334
  - UltraLite C++ table access of current, 19
  - UltraLite C++ table navigation, 17
  - UltraLite deletions with C++ API, 22
  - UltraLite insertions with C++ API, 21
  - UltraLite updates with C++ API, 21
- runtime libraries
  - UltraLite applications for Windows Mobile, 59
  - UltraLite C++, 26
  - UltraLite for C++, 26
- runtime library
  - Windows Mobile, 69

## S

- sample application
  - UltraLite building for Windows Mobile, 59
- samples-dir
  - documentation usage, xii

---

schemas  
  UltraLite C++ API access, 23

scrolling  
  UltraLite C++ API, 17

searching  
  UltraLite rows with C++, 20

security  
  changing the encryption key in UltraLite embedded SQL, 49  
  obfuscation in UltraLite embedded SQL, 48  
  UltraLite database encryption, 48

SELECT statement  
  UltraLite C++ data retrieval example, 15  
  UltraLite embedded SQL single row, 43

SendNotification method  
  ULConnection class [UltraLite C/C++ API], 127

SET CONNECTION statement  
  multiple connections in UltraLite embedded SQL, 33

SetBinary method  
  ULResultSet class [UltraLite C/C++ API], 213

SetDatabaseOption method  
  ULConnection class [UltraLite C/C++ API], 129

SetDatabaseOptionInt method  
  ULConnection class [UltraLite C/C++ API], 130

SetDateTime method  
  ULResultSet class [UltraLite C/C++ API], 214

SetDefault method  
  ULResultSet class [UltraLite C/C++ API], 216

SetDouble method  
  ULResultSet class [UltraLite C/C++ API], 217

SetErrorCallback method  
  ULDatabaseManager class [UltraLite C/C++ API], 148

SetFloat method  
  ULResultSet class [UltraLite C/C++ API], 219

SetGuid method  
  ULResultSet class [UltraLite C/C++ API], 220

SetInt method  
  ULResultSet class [UltraLite C/C++ API], 221

SetIntWithType method  
  ULResultSet class [UltraLite C/C++ API], 223

SetNull method  
  ULResultSet class [UltraLite C/C++ API], 225

SetParameterBinary method  
  ULPreparedStatement class [UltraLite C/C++ API], 174

SetParameterDateTime method  
  ULPreparedStatement class [UltraLite C/C++ API], 175

SetParameterDouble method  
  ULPreparedStatement class [UltraLite C/C++ API], 175

SetParameterFloat method  
  ULPreparedStatement class [UltraLite C/C++ API], 176

SetParameterGuid method  
  ULPreparedStatement class [UltraLite C/C++ API], 176

SetParameterInt method  
  ULPreparedStatement class [UltraLite C/C++ API], 176

SetParameterIntWithType method  
  ULPreparedStatement class [UltraLite C/C++ API], 177

SetParameterNull method  
  ULPreparedStatement class [UltraLite C/C++ API], 178

SetParameterString method  
  ULPreparedStatement class [UltraLite C/C++ API], 178

SetString method  
  ULResultSet class [UltraLite C/C++ API], 227

SetSynchronizationCallback method  
  ULConnection class [UltraLite C/C++ API], 131

SetSyncInfo method  
  ULConnection class [UltraLite C/C++ API], 131

simple encryption  
  UltraLite database simple encryption, 49

SQL Anywhere  
  documentation, xi

SQL Anywhere Developer Centers  
  finding out more and requesting technical support, xv

SQL Anywhere Tech Corner  
  finding out more and requesting technical support, xv

SQL Communications Area  
  UltraLite embedded SQL, 31

SQL preprocessor utility (sqlpp)  
  UltraLite embedded SQL applications, 56

SQLCA  
  UltraLite embedded SQL, 31  
  UltraLite embedded SQL multiple SQLCA, 32  
  UltraLite fields, 31

sqlcabc SQLCA field

- UltraLite embedded SQL, 31
- sqlcaid SQLCA field
  - UltraLite embedded SQL, 31
- SQLCODE
  - UltraLite C++ error handling, 24
- sqlcode SQLCA field
  - UltraLite embedded SQL, 31
- sqlerrd SQLCA field
  - UltraLite embedded SQL, 32
- sqlerrmc SQLCA field
  - UltraLite embedded SQL, 32
- sqlerrml SQLCA field
  - UltraLite embedded SQL, 32
- sqlerrp SQLCA field
  - UltraLite embedded SQL, 32
- sqlpp utility
  - UlgraLite usage, 56
  - UltraLite embedded SQL applications, 56
- sqlstate SQLCA field
  - UltraLite embedded SQL, 32
- sqlwarn SQLCA field
  - UltraLite embedded SQL, 32
- StartSynchronizationDelete method
  - ULConnection class [UltraLite C/C++ API], 132
- static libraries
  - UltraLite C++ applications, 26
- StopSynchronizationDelete method
  - ULConnection class [UltraLite C/C++ API], 133
- string UltraLite embedded SQL data type
  - about, 35
  - fixed length, 35
  - variable length, 35
- strings
  - UL\_TEXT macro, 69
- strong encryption
  - UltraLite embedded SQL, 48
- support
  - newsgroups, xiv
- supported platforms
  - UltraLite C++, 2
- swipe-to-delete
  - iPhone, 351
- synchronization
  - adding to UltraLite embedded SQL applications, 49
  - canceling in UltraLite embedded SQL, 52
  - for UltraLite for Windows Mobile menu control, 66

- initial in UltraLite embedded SQL, 51
- invoking in UltraLite embedded SQL, 51
- iPhone, 352
- monitoring in UltraLite embedded SQL, 52
- UltraLite C++ API tutorial, 335
- UltraLite embedded SQL, 49
- UltraLite embedded SQL committing changes in , 51
- UltraLite embedded SQL example, 50
- UltraLite for Windows Mobile introduction, 63
- synchronization errors
  - UltraLite embedded SQL communications errors , 52
- Synchronize method
  - ULConnection class [UltraLite C/C++ API], 133
- SynchronizeFromProfile method
  - ULConnection class [UltraLite C/C++ API], 134
- synchronizing
  - iPhone, 352
  - UltraLite C/C++ applications, 26
- synchronizing data
  - UltraLite about, 26

## T

- tables
  - UltraLite C++ API schema information in , 23
- target platforms
  - UltraLite C++, 2
- tech corners
  - finding out more and requesting technical support, xv
- technical support
  - newsgroups, xiv
- threads
  - UltraLite C++ API multi-threaded applications, 10
  - UltraLite embedded SQL, 32
- timestamp structure UltraLite embedded SQL data type
  - about, 36
- tips
  - UltraLite development, 51
- transaction processing
  - UltraLite C++ management of, 22
- transactions
  - committing in UltraLite with embedded SQL, 51
  - UltraLite C++ management of, 22
- TriggerEvent method
  - ULConnection class [UltraLite C/C++ API], 136



---

troubleshooting  
  newsgroups, xiv  
  UltraLite C++ handling errors, 24  
  UltraLite development, 51  
  UltraLite synchronization with embedded SQL, 51  
  UltraLite using reference expressions in SQL  
  preprocessor, 39  
TruncateTable method  
  ULTable class [UltraLite C/C++ API], 243  
truncation  
  UltraLite FETCH, 43  
tutorials  
  UltraLite C++ API, 329, 339

## U

ul\_binary structure [UltraLite C and Embedded SQL datatypes]  
  description, 80  
ul\_column\_default\_type enumeration [UltraLite C++ API]  
  description, 254  
ul\_column\_name\_type enumeration [UltraLite C++ API]  
  description, 255  
ul\_column\_sql\_type enumeration [UltraLite C and Embedded SQL datatypes]  
  description, 70  
ul\_column\_storage\_type enumeration [UltraLite C and Embedded SQL datatypes]  
  description, 72  
ul\_database\_option\_id enumeration [UltraLite C and Embedded SQL datatypes]  
  description, 73  
ul\_database\_property\_id enumeration [UltraLite C and Embedded SQL datatypes]  
  description, 74  
ul\_error\_action enumeration [UltraLite C and Embedded SQL datatypes]  
  description, 76  
ul\_error\_info structure [UltraLite C and Embedded SQL datatypes]  
  description, 80  
ul\_index\_flag enumeration [UltraLite C++ API]  
  description, 255  
UL\_RS\_STATE enumeration [UltraLite C and Embedded SQL datatypes]  
  description, 70  
ul\_stream\_error structure [UltraLite C and Embedded SQL datatypes]  
  description, 81  
UL\_SYNC\_ALL variable [UltraLite C and Embedded SQL datatypes]  
  description, 91  
UL\_SYNC\_ALL\_PUBS variable [UltraLite C and Embedded SQL datatypes]  
  description, 91  
ul\_sync\_info structure  
  about, 50  
ul\_sync\_info structure [UltraLite C and Embedded SQL datatypes]  
  description, 81  
ul\_sync\_info\_w2 structure [UltraLite C and Embedded SQL datatypes]  
  description, 83  
ul\_sync\_result structure [UltraLite C and Embedded SQL datatypes]  
  description, 86  
ul\_sync\_state enumeration [UltraLite C and Embedded SQL datatypes]  
  description, 76  
ul\_sync\_stats structure [UltraLite C and Embedded SQL datatypes]  
  description, 87  
ul\_sync\_status structure  
  UltraLite embedded SQL, 53  
ul\_sync\_status structure [UltraLite C and Embedded SQL datatypes]  
  description, 87  
UL\_SYNC\_STATUS\_FLAG\_IS\_BLOCKING variable [UltraLite C and Embedded SQL datatypes]  
  description, 91  
ul\_table\_sync\_type enumeration [UltraLite C++ API]  
  description, 256  
UL\_TEXT macro  
  about, 69  
UL\_USE\_DLL macro  
  about, 69  
UL\_VALID\_IS\_ERROR variable [UltraLite C and Embedded SQL datatypes]  
  description, 91  
UL\_VALID\_IS\_INFO variable [UltraLite C and Embedded SQL datatypes]  
  description, 92  
ul\_validate\_data structure [UltraLite C and Embedded SQL datatypes]

- description, 88
- ul\_validate\_status\_id enumeration [UltraLite C and Embedded SQL datatypes]
  - description, 78
- ULActiveSyncStream function
  - Windows Mobile usage, 63
- ulbase.lib
  - UltraLite C++ development, 26
- ULCancelGetNotification method [UltraLite Embedded SQL API]
  - description, 270
- ULCancelGetNotificationW method [UltraLite Embedded SQL API]
  - description, 270
- ULChangeEncryptionKey function [UL ESQL]
  - using, 49
- ULChangeEncryptionKey method [UltraLite Embedded SQL API]
  - description, 271
- ULChangeEncryptionKeyW method [UltraLite Embedded SQL API]
  - description, 271
- ULCheckpoint method [UltraLite Embedded SQL API]
  - description, 272
- ULConnection class [UltraLite C++ API]
  - CancelGetNotification method, 95
  - ChangeEncryptionKey method, 96
  - Checkpoint method, 97
  - Close method, 98
  - Commit method, 98
  - CountUploadRows method, 98
  - CreateNotificationQueue method, 99
  - DeclareEvent method, 101
  - description, 93
  - DestroyNotificationQueue method, 102
  - ExecuteScalar method, 103
  - ExecuteScalarV method, 106
  - ExecuteStatement method, 109
  - GetChildObjectCount method, 110
  - GetDatabaseProperty method, 110
  - GetDatabasePropertyInt method, 111
  - GetDatabaseSchema method, 112
  - GetLastDownloadTime method, 113
  - GetLastError method, 114
  - GetLastIdentity method, 114
  - GetNotification method, 114
  - GetNotificationParameter method, 116
  - GetSqlca method, 117
  - GetSyncResult method, 118
  - GlobalAutoincUsage method, 118
  - GrantConnectTo method, 118
  - InitSyncInfo method, 119
  - OpenTable method, 120
  - PrepareStatement method, 121
  - RegisterForEvent method, 122
  - ResetLastDownloadTime method, 125
  - RevokeConnectFrom method, 126
  - Rollback method, 126
  - RollbackPartialDownload method, 127
  - SendNotification method, 127
  - SetDatabaseOption method, 129
  - SetDatabaseOptionInt method, 130
  - SetSynchronizationCallback method, 131
  - SetSyncInfo method, 131
  - StartSynchronizationDelete method, 132
  - StopSynchronizationDelete method, 133
  - Synchronize method, 133
  - SynchronizeFromProfile method, 134
  - TriggerEvent method, 136
  - ValidateDatabase method, 137
- ULCountUploadRows method [UltraLite Embedded SQL API]
  - description, 272
- ULCountUploadRowsW method [UltraLite Embedded SQL API]
  - description, 273
- ulcpp.h header file
  - UltraLite C/C++ API reference, 93
- ULCreateDatabase method [UltraLite Embedded SQL API]
  - description, 274
- ULCreateDatabaseW method [UltraLite Embedded SQL API]
  - description, 275
- ULCreateNotificationQueue method [UltraLite Embedded SQL API]
  - description, 276
- ULCreateNotificationQueueW method [UltraLite Embedded SQL API]
  - description, 277
- ULDatabaseManager class [UltraLite C++ API]
  - CreateDatabase method, 141
  - description, 139
  - DropDatabase method, 142
  - EnableAesDBEncryption method, 143
  - EnableAesFipsDBEncryption method, 143

---

- EnableEccE2ee method, 144
- EnableEccSyncEncryption method, 144
- EnableHttpsSynchronization method, 144
- EnableHttpSynchronization method, 144
- EnableRsaE2ee method, 144
- EnableRsaFipsE2ee method, 144
- EnableRsaFipsSyncEncryption method, 145
- EnableRsaSyncEncryption method, 145
- EnableTcpipSynchronization method, 145
- EnableTlsSynchronization method, 145
- EnableZlibSyncCompression method, 145
- Fini method, 146
- Init method, 146
- OpenConnection method, 146
- SetErrorCallback method, 148
- ValidateDatabase method, 149
- ULDatabaseSchema class [UltraLite C++ API]
  - Close method, 151
  - description, 151
  - GetConnection method, 151
  - GetNextPublication method, 152
  - GetNextPublicationW2 method, 152
  - GetNextTable method, 153
  - GetPublicationCount method, 153
  - GetTableCount method, 154
  - GetTableSchema method, 154
- ULDatabaseSchema object
  - UltraLite C++, 23
- ULDeclareEvent method [UltraLite Embedded SQL API]
  - description, 278
- ULDeclareEventW method [UltraLite Embedded SQL API]
  - description, 278
- ULDeleteAllRows method [UltraLite Embedded SQL API]
  - description, 279
- ULDestroyNotificationQueue method [UltraLite Embedded SQL API]
  - description, 279
- ULDestroyNotificationQueueW method [UltraLite Embedded SQL API]
  - description, 280
- ulecc.lib
  - UltraLite C++ development, 26
- ULECCLibraryVersion method [UltraLite Embedded SQL API]
  - description, 280
- ULEnableAesDBEncryption method [UltraLite Embedded SQL API]
  - description, 281
- ULEnableAesFipsDBEncryption method [UltraLite Embedded SQL API]
  - description, 281
- ULEnableEccE2ee method [UltraLite Embedded SQL API]
  - description, 282
- ULEnableEccSyncEncryption method [UltraLite Embedded SQL API]
  - description, 282
- ULEnableHttpSynchronization method [UltraLite Embedded SQL API]
  - description, 283
- ULEnableRsaE2ee method [UltraLite Embedded SQL API]
  - description, 283
- ULEnableRsaFipsE2ee method [UltraLite Embedded SQL API]
  - description, 283
- ULEnableRsaFipsSyncEncryption method [UltraLite Embedded SQL API]
  - description, 283
- ULEnableRsaSyncEncryption method [UltraLite Embedded SQL API]
  - description, 284
- ULEnableTcpipSynchronization method [UltraLite Embedded SQL API]
  - description, 284
- ULEnableZlibSyncCompression method [UltraLite Embedded SQL API]
  - description, 285
- ULError class [UltraLite C++ API]
  - Clear method, 156
  - description, 155
  - GetErrorInfo method, 156
  - GetParameter method, 157
  - GetParameterCount method, 158
  - GetSQLCode method, 158
  - GetSQLCount method, 158
  - GetString method, 159
  - GetURL method, 160
  - IsOK method, 161
  - ULError constructor, 155
- ULError constructor
  - ULError class [UltraLite C/C++ API], 155

- ULErrorInfoInitFromSqlca method [UltraLite Embedded SQL API]
  - description, 285
- ULErrorInfoParameterAt method [UltraLite Embedded SQL API]
  - description, 286
- ULErrorInfoParameterCount method [UltraLite Embedded SQL API]
  - description, 286
- ULErrorInfoString method [UltraLite Embedded SQL API]
  - description, 286
- ULErrorInfoURL method [UltraLite Embedded SQL API]
  - description, 287
- ulfips.lib
  - UltraLite C++ development, 26
- ULGetDatabaseID method [UltraLite Embedded SQL API]
  - description, 287
- ULGetDatabaseProperty method [UltraLite Embedded SQL API]
  - description, 288
- ULGetDatabasePropertyW method [UltraLite Embedded SQL API]
  - description, 288
- ULGetErrorParameter method [UltraLite Embedded SQL API]
  - description, 289
- ULGetErrorParameterCount method [UltraLite Embedded SQL API]
  - description, 289
- ULGetErrorParameterW method [UltraLite Embedded SQL API]
  - description, 290
- ULGetIdentity method [UltraLite Embedded SQL API]
  - description, 291
- ULGetLastDownloadTime method [UltraLite Embedded SQL API]
  - description, 291
- ULGetLastDownloadTimeW method [UltraLite Embedded SQL API]
  - description, 292
- ULGetNotification method [UltraLite Embedded SQL API]
  - description, 292
- ULGetNotificationParameter method [UltraLite Embedded SQL API]
  - description, 293
- ULGetNotificationParameterW method [UltraLite Embedded SQL API]
  - description, 294
- ULGetNotificationW method [UltraLite Embedded SQL API]
  - description, 295
- ULGetSyncResult method [UltraLite Embedded SQL API]
  - description, 295
- ULGlobalAutoincUsage method [UltraLite Embedded SQL API]
  - description, 296
- ULGrantConnectTo method [UltraLite Embedded SQL API]
  - description, 296
- ULGrantConnectToW method [UltraLite Embedded SQL API]
  - description, 297
- ulimp.lib
  - UltraLite C++ development, 26
- ULIndexSchema class [UltraLite C++ API]
  - Close method, 162
  - description, 161
  - GetColumnCount method, 162
  - GetColumnName method, 163
  - GetColumnNameW2 method, 163
  - GetConnection method, 163
  - GetIndexColumnID method, 163
  - GetIndexFlags method, 164
  - GetName method, 165
  - GetNameW2 method, 165
  - GetReferencedIndexName method, 165
  - GetReferencedIndexNameW2 method, 166
  - GetReferencedTableName method, 166
  - GetReferencedTableNameW2 method, 166
  - GetTableName method, 167
  - GetTableNameW2 method, 167
  - IsColumnDescending method, 167
- ULIndexSchema object
  - UltraLite C++ development, 23
- ULInitSyncInfo function [UL ESQL]
  - about, 50
- ULInitSyncInfo method [UltraLite Embedded SQL API]
  - description, 298
- ULInitSyncInfoW method [UltraLite Embedded SQL API]

---

- description, 298
- ULIsSynchronizeMessage function [UL ESQL]
  - ActiveSync usage, 63
- ULIsSynchronizeMessage method [UltraLite Embedded SQL API]
  - description, 298
- ULLibraryVersion method [UltraLite Embedded SQL API]
  - description, 299
- ULPreparedStatement class
  - UltraLite C++, 11
- ULPreparedStatement class [UltraLite C++ API]
  - AppendParameterByteChunk method, 169
  - AppendParameterStringChunk method, 169
  - Close method, 171
  - description, 167
  - ExecuteQuery method, 171
  - ExecuteStatement method, 171
  - GetConnection method, 171
  - GetParameterCount method, 171
  - GetParameterType method, 172
  - GetPlan method, 172
  - GetResultSetSchema method, 174
  - GetRowsAffectedCount method, 174
  - HasResultSet method, 174
  - SetParameterBinary method, 174
  - SetParameterDateTime method, 175
  - SetParameterDouble method, 175
  - SetParameterFloat method, 176
  - SetParameterGuid method, 176
  - SetParameterInt method, 176
  - SetParameterIntWithType method, 177
  - SetParameterNull method, 178
  - SetParameterString method, 178
- ulprotos.h header file
  - UltraLite Embedded SQL API, 259
- ULRegisterForEvent method [UltraLite Embedded SQL API]
  - description, 300
- ULRegisterForEventW method [UltraLite Embedded SQL API]
  - description, 300
- ULResetLastDownloadTime method [UltraLite Embedded SQL API]
  - description, 302
- ULResetLastDownloadTimeW method [UltraLite Embedded SQL API]
  - description, 302
- ULResultSet class [UltraLite C++ API]
  - AfterLast method, 181
  - AppendByteChunk method, 182
  - AppendStringChunk method, 183
  - BeforeFirst method, 186
  - Close method, 186
  - Delete method, 187
  - DeleteNamed method, 187
  - description, 179
  - First method, 188
  - GetBinary method, 188
  - GetBinaryLength method, 190
  - GetByteChunk method, 191
  - GetConnection method, 193
  - GetDateTime method, 193
  - GetDouble method, 195
  - GetFloat method, 196
  - GetGuid method, 197
  - GetInt method, 199
  - GetIntWithType method, 200
  - GetResultSetSchema method, 203
  - GetRowCount method, 203
  - GetState method, 203
  - GetString method, 203
  - GetStringChunk method, 206
  - GetStringLength method, 209
  - IsNull method, 211
  - Last method, 212
  - Next method, 212
  - Previous method, 213
  - Relative method, 213
  - SetBinary method, 213
  - SetDateTime method, 214
  - SetDefault method, 216
  - SetDouble method, 217
  - SetFloat method, 219
  - SetGuid method, 220
  - SetInt method, 221
  - SetIntWithType method, 223
  - SetNull method, 225
  - SetString method, 227
  - Update method, 229
  - UpdateBegin method, 229
- ULResultSetSchema class [UltraLite C++ API]
  - description, 230
  - GetColumnCount method, 230
  - GetColumnID method, 231
  - GetColumnName method, 232

- GetColumnNameW2 method, 232
- GetColumnPrecision method, 233
- GetColumnScale method, 233
- GetColumnSize method, 233
- GetColumnSQLType method, 234
- GetColumnType method, 234
- GetConnection method, 235
- IsAliased method, 235
- ULResultSetSchema object
  - UltraLite C++, 23
- ULRevokeConnectFrom method [UltraLite Embedded SQL API]
  - description, 303
- ULRevokeConnectFromW method [UltraLite Embedded SQL API]
  - description, 303
- ULRollbackPartialDownload method [UltraLite Embedded SQL API]
  - description, 304
- ulrsa.lib
  - UltraLite C++ development, 26
- ULRSALibraryVersion method [UltraLite Embedded SQL API]
  - description, 299
- ulrt.lib
  - UltraLite C++ development, 26
- ulrt12.dll
  - UltraLite C++ development, 26
- ulrtc.lib
  - UltraLite C++ development, 26
- ULSendNotification method [UltraLite Embedded SQL API]
  - description, 304
- ULSendNotificationW method [UltraLite Embedded SQL API]
  - description, 305
- ULSetDatabaseID method [UltraLite Embedded SQL API]
  - description, 306
- ULSetDatabaseOptionString method [UltraLite Embedded SQL API]
  - description, 306
- ULSetDatabaseOptionStringW method [UltraLite Embedded SQL API]
  - description, 306
- ULSetDatabaseOptionULong method [UltraLite Embedded SQL API]
  - description, 307
- ULSetErrorCallback method [UltraLite Embedded SQL API]
  - description, 307
- ULSetErrorCallbackW method [UltraLite Embedded SQL API]
  - description, 308
- ULSetSynchronizationCallback method [UltraLite Embedded SQL API]
  - description, 310
- ULSetSynchronizationCallbackEx method [UltraLite Embedded SQL API]
  - description, 310
- ULSetSyncInfo method [UltraLite Embedded SQL API]
  - description, 308
- ULSetSyncInfoW method [UltraLite Embedded SQL API]
  - description, 309
- ULSignalSyncIsComplete method [UltraLite Embedded SQL API]
  - description, 311
- ULStartSynchronizationDelete method [UltraLite Embedded SQL API]
  - description, 311
- ULStaticFini method [UltraLite Embedded SQL API]
  - description, 311
- ULStaticInit method [UltraLite Embedded SQL API]
  - description, 311
- ULStopSynchronizationDelete method [UltraLite Embedded SQL API]
  - description, 312
- ULSynchronize method [UltraLite Embedded SQL API]
  - description, 312
- ULSynchronizeFromProfile method [UltraLite Embedded SQL API]
  - description, 312
- ULSynchronizeFromProfileW method [UltraLite Embedded SQL API]
  - description, 313
- ULSynchronizeW method [UltraLite Embedded SQL API]
  - description, 314
- ULTable class
  - UltraLite C++ introduction, 17
- ULTable class [UltraLite C++ API]
  - DeleteAllRows method, 238
  - description, 235

---

- Find method, 239
- FindBegin method, 239
- FindFirst method, 239
- FindLast method, 240
- FindNext method, 240
- FindPrevious method, 240
- GetTableSchema method, 241
- Insert method, 241
- InsertBegin method, 241
- Lookup method, 242
- LookupBackward method, 242
- LookupBegin method, 242
- LookupForward method, 243
- TruncateTable method, 243
- ULTable object
  - UltraLite C++ data retrieval example, 15
- ULTableSchema class [UltraLite C++ API]
  - Close method, 245
  - description, 243
  - GetColumnDefault method, 245
  - GetColumnDefaultType method, 246
  - GetColumnDefaultW2 method, 246
  - GetGlobalAutoincPartitionSize method, 246
  - GetIndexCount method, 247
  - GetIndexSchema method, 247
  - GetName method, 248
  - GetNameW2 method, 248
  - GetNextIndex method, 249
  - GetOptimalIndex method, 249
  - GetOptimalIndexW2 method, 249
  - GetPrimaryKey method, 250
  - GetPublicationPredicate method, 250
  - GetTableSyncType method, 251
  - InPublication method, 252
  - IsColumnInIndex method, 252
  - IsColumnNullable method, 254
- ULTableSchema object
  - UltraLite C++ development, 23
- UltraLite
  - compiling library for iPhone, 339
  - creating an iPhone application project, 340
  - iPhone frameworks, 342
- UltraLite applications
  - synchronization of C/C++ applications, 26
- UltraLite C and Embedded SQL datatypes
  - UL\_AS\_SYNCHRONIZE variable, 90
  - ul\_binary structure, 80
  - ul\_column\_sql\_type enumeration, 70
  - ul\_column\_storage\_type enumeration, 72
  - ul\_database\_option\_id enumeration, 73
  - ul\_database\_property\_id enumeration, 74
  - ul\_error\_action enumeration, 76
  - ul\_error\_info structure, 80
  - UL\_RS\_STATE enumeration, 70
  - ul\_stream\_error structure, 81
  - UL\_SYNC\_ALL variable, 91
  - UL\_SYNC\_ALL\_PUBS variable, 91
  - ul\_sync\_info structure, 81
  - ul\_sync\_info\_w2 structure, 83
  - ul\_sync\_result structure, 86
  - ul\_sync\_state enumeration, 76
  - ul\_sync\_stats structure, 87
  - ul\_sync\_status structure, 87
  - UL\_SYNC\_STATUS\_FLAG\_IS\_BLOCKING variable, 91
  - UL\_VALID\_IS\_ERROR variable, 91
  - UL\_VALID\_IS\_INFO variable, 92
  - ul\_validate\_data structure, 88
  - ul\_validate\_status\_id enumeration, 78
  - ULVF\_DATABASE variable, 89
  - ULVF\_EXPRESS variable, 89
  - ULVF\_FULL\_VALIDATE variable, 89
  - ULVF\_IDX\_HASH variable, 90
  - ULVF\_IDX\_REDUNDANT variable, 90
  - ULVF\_INDEX variable, 90
  - ULVF\_TABLE variable, 90
- UltraLite C++
  - development, 7
  - iPhone build settings, 7
  - iPhone include files, 7
  - Mac OS X include files, 7
  - quick start, 7
- UltraLite C++ API
  - ul\_column\_default\_type enumeration, 254
  - ul\_column\_name\_type enumeration, 255
  - ul\_index\_flag enumeration, 255
  - ul\_table\_sync\_type enumeration, 256
  - ULConnection class, 93
  - ULDatabaseManager class, 139
  - ULDatabaseSchema class, 151
  - ULError class, 155
  - ULIndexSchema class, 161
  - ULPreparedStatement class, 167
  - ULResultSet class, 179
  - ULResultSetSchema class, 230
  - ULTable class, 235

- ULTableSchema class, 243
- UltraLite C/C++
  - about, 1
  - accessing schema information, 23
  - architecture, 2
  - authentication, 24
  - data manipulation with SQL, 10
  - encryption, 25
  - INCLUDE environment variable, 329
  - iPhone tutorial, 339
  - obfuscating UltraLite databases, 49
  - supported platforms, 2
  - synchronizing data, 26
  - tutorials, 329, 339
- UltraLite C/C++ API reference
  - ulcpp.h header file, 93
- UltraLite C/C++ Common API
  - alphabetical listing, 69
- UltraLite databases
  - connecting in UltraLite C++, 8
  - encrypting in embedded SQL, 48
  - UltraLite C++ information access , 23
  - UltraLite for Windows Mobile, 60
- UltraLite embedded SQL
  - authorization, 31
  - building CustDB application, 59
  - cursors, 44
  - developing applications, 29
  - fetching data, 43
  - functions, 259
  - host variables, 33
  - synchronization, 49
  - using, 259
- UltraLite Embedded SQL API
  - db\_fini method, 259
  - db\_init method, 259
  - ml\_file\_transfer\_info structure, 320
  - ml\_file\_transfer\_status structure, 325
  - MLFileDownload method, 266
  - MLFileUpload method, 267
  - MLFiniFileTransferInfo method, 268
  - mlft\_stream\_error structure, 326
  - MLFTEnableEccE2ee method, 260
  - MLFTEnableEccEncryption method, 261
  - MLFTEnableRsaE2ee method, 262
  - MLFTEnableRsaEncryption method, 262
  - MLFTEnableRsaFipsE2ee method, 263
  - MLFTEnableRsaFipsEncryption method, 264
  - MLFTEnableZlibCompression method, 265
  - MLInitFileTransferInfo method, 269
  - ULCancelGetNotification method, 270
  - ULChangeEncryptionKey method, 271
  - ULCheckpoint method, 272
  - ULCountUploadRows method, 272
  - ULCreateDatabase method, 274
  - ULCreateNotificationQueue method, 276
  - ULDeclareEvent method, 278
  - ULDeleteAllRows method, 279
  - ULDestroyNotificationQueue method, 279
  - ULECCLibraryVersion method, 280
  - ULEnableAesDBEncryption method, 281
  - ULEnableAesFipsDBEncryption method, 281
  - ULEnableEccE2ee method, 282
  - ULEnableEccSyncEncryption method, 282
  - ULEnableHttpSynchronization method, 283
  - ULEnableRsaE2ee method, 283
  - ULEnableRsaFipsE2ee method, 283
  - ULEnableRsaFipsSyncEncryption method, 283
  - ULEnableRsaSyncEncryption method, 284
  - ULEnableTcpipSynchronization method, 284
  - ULEnableZlibSyncCompression method, 285
  - ULErrorInfoInitFromSqlca method, 285
  - ULErrorInfoParameterAt method, 286
  - ULErrorInfoParameterCount method, 286
  - ULErrorInfoString method, 286
  - ULErrorInfoURL method, 287
  - ULGetDatabaseID method, 287
  - ULGetDatabaseProperty method, 288
  - ULGetErrorParameter method, 289
  - ULGetErrorParameterCount method, 289
  - ULGetIdentity method, 291
  - ULGetLastDownloadTime method, 291
  - ULGetNotification method, 292
  - ULGetNotificationParameter method, 293
  - ULGetSyncResult method, 295
  - ULGlobalAutoincUsage method, 296
  - ULGrantConnectTo method, 296
  - ULInitSyncInfo method, 298
  - ULIsSynchronizeMessage method, 298
  - ULLibraryVersion method, 299
  - ulprotos.h header file, 259
  - ULRegisterForEvent method, 300
  - ULResetLastDownloadTime method, 302
  - ULRevokeConnectFrom method, 303
  - ULRollbackPartialDownload method, 304
  - ULRSALibraryVersion method, 299



---

- ULSendNotification method, 304
- ULSetDatabaseID method, 306
- ULSetDatabaseOptionString method, 306
- ULSetDatabaseOptionULong method, 307
- ULSetErrorCallback method, 307
- ULSetSynchronizationCallback method, 310
- ULSetSynchronizationCallbackEx method, 310
- ULSetSyncInfo method, 308
- ULSignalSyncIsComplete method, 311
- ULStartSynchronizationDelete method, 311
- ULStaticFini method, 311
- ULStaticInit method, 311
- ULStopSynchronizationDelete method, 312
- ULSynchronize method, 312
- ULSynchronizeFromProfile method, 312
- ULTriggerEvent method, 314
- ULTruncateTable method, 315
- ULValidateDatabase method, 316
- ULValidateDatabaseTableName method, 317
- UltraLite Embedded SQL API [UNICODE]
  - ml\_file\_transfer\_info\_w structure, 322
  - MLFileDownloadW method, 266
  - MLFileUploadW method, 267
  - MLFiniFileTransferInfoW method, 268
  - mlft\_stream\_error\_w structure, 326
  - MLFTEnableEccE2eeW method, 260
  - MLFTEnableEccEncryptionW method, 261
  - MLFTEnableRsaE2eeW method, 262
  - MLFTEnableRsaEncryptionW method, 263
  - MLFTEnableRsaFipsE2eeW method, 264
  - MLFTEnableRsaFipsEncryptionW method, 264
  - MLFTEnableZlibCompressionW method, 265
  - MLInitFileTransferInfoW method, 269
  - ULCancelGetNotificationW method, 270
  - ULChangeEncryptionKeyW method, 271
  - ULCountUploadRowsW method, 273
  - ULCreateDatabaseW method, 275
  - ULCreateNotificationQueueW method, 277
  - ULDeclareEventW method, 278
  - ULDestroyNotificationQueueW method, 280
  - ULGetDatabasePropertyW method, 288
  - ULGetErrorParameterW method, 290
  - ULGetLastDownloadTimeW method, 292
  - ULGetNotificationParameterW method, 294
  - ULGetNotificationW method, 295
  - ULGrantConnectToW method, 297
  - ULInitSyncInfoW method, 298
  - ULRegisterForEventW method, 300
  - ULResetLastDownloadTimeW method, 302
  - ULRevokeConnectFromW method, 303
  - ULSendNotificationW method, 305
  - ULSetDatabaseOptionStringW method, 306
  - ULSetErrorCallbackW method, 308
  - ULSetSyncInfoW method, 309
  - ULSynchronizeFromProfileW method, 313
  - ULSynchronizeW method, 314
  - ULTriggerEventW method, 315
  - ULValidateDatabaseTableNameW method, 318
  - ULValidateDatabaseW method, 319
- UltraLite runtime
  - deploying Windows Mobile libraries, 61
  - UltraLite C++ libraries, 26
- UltraLite runtime libraries
  - iPhone, 28
  - Linux, 27
  - Mac OS X, 28
  - Windows, 27
  - Windows Mobile, 26
- ULTriggerEvent method [UltraLite Embedded SQL API]
  - description, 314
- ULTriggerEventW method [UltraLite Embedded SQL API]
  - description, 315
- ULTruncateTable method [UltraLite Embedded SQL API]
  - description, 315
- ULValidateDatabase method [UltraLite Embedded SQL API]
  - description, 316
- ULValidateDatabaseTableName method [UltraLite Embedded SQL API]
  - description, 317
- ULValidateDatabaseTableNameW method [UltraLite Embedded SQL API]
  - description, 318
- ULValidateDatabaseW method [UltraLite Embedded SQL API]
  - description, 319
- ULVF\_AS\_SYNCHRONIZE variable [UltraLite C and Embedded SQL datatypes]
  - description, 90
- ULVF\_DATABASE variable [UltraLite C and Embedded SQL datatypes]
  - description, 89

ULVF\_EXPRESS variable [UltraLite C and Embedded SQL datatypes]  
description, 89

ULVF\_FULL\_VALIDATE variable [UltraLite C and Embedded SQL datatypes]  
description, 89

ULVF\_IDX\_HASH variable [UltraLite C and Embedded SQL datatypes]  
description, 90

ULVF\_IDX\_REDUNDANT variable [UltraLite C and Embedded SQL datatypes]  
description, 90

ULVF\_INDEX variable [UltraLite C and Embedded SQL datatypes]  
description, 90

ULVF\_TABLE variable [UltraLite C and Embedded SQL datatypes]  
description, 90

uncommitted transactions  
UltraLite embedded SQL, 51

UNDER\_CE compiler directive  
about, 70

Unix  
documentation conventions, xi  
operating systems, xi

Update method  
ULResultSet class [UltraLite C/C++ API], 229

update mode  
UltraLite C++, 18

UpdateBegin method  
ULResultSet class [UltraLite C/C++ API], 229

updating  
UltraLite C++ API table rows, 21

user authentication  
UltraLite C++ development, 24  
UltraLite embedded SQL applications, 47

## V

ValidateDatabase method  
ULConnection class [UltraLite C/C++ API], 137  
ULDatabaseManager class [UltraLite C/C++ API], 149

values  
UltraLite C++ API accessing, 19

Visual C++  
UltraLite for Windows Mobile development, 59

## W

WindowProc function  
ActiveSync usage, 64

Windows  
documentation conventions, xi  
operating systems, xi  
UltraLite runtime libraries for, 27

Windows Mobile  
documentation conventions, xi  
operating systems, xi  
UltraLite application development overview, 59  
UltraLite application synchronization, 63  
UltraLite class names, 61  
UltraLite platform requirements, 59  
UltraLite runtime libraries for, 26  
UltraLite synchronization menu control, 66  
UltraLite synchronization with ActiveSync, 61  
Windows CE, xi

winsock.lib  
UltraLite Windows Mobile applications, 59