UltraLite®
C and C++ Programming

Published: March 2007

Copyright and trademarks

Copyright © 2007 iAnywhere Solutions, Inc. Portions copyright © 2007 Sybase, Inc. All rights reserved.

iAnywhere Solutions, Inc. is a subsidiary of Sybase, Inc.

iAnywhere grants you permission to use this document for your own informational, educational, and other non-commercial purposes; provided that (1) you include this and all other copyright and proprietary notices in the document in all copies; (2) you do not attempt to "pass-off" the document as your own; and (3) you do not modify the document. You may not publish or distribute the document or any portion thereof without the express prior written consent of iAnywhere.

This document is not a commitment on the part of iAnywhere to do or refrain from any activity, and iAnywhere may change the content of this document at its sole discretion without notice. Except as otherwise provided in a written agreement between you and iAnywhere, this document is provided “as is”, and iAnywhere assumes no liability for its use or any inaccuracies it may contain.

iAnywhere®, Sybase®, and the marks listed at http://www.ianywhere.com/trademarks are trademarks of Sybase, Inc. or its subsidiaries. ® indicates registration in the United States of America.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

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

Contents

About This Manual ix

SQL Anywhere documentation x
Documentation conventions xiii
Finding out more and providing feedback xvii

I. Introduction 1

Introduction to UltraLite for C/C++ Developers 3
UltraLite and the C/C++ programming languages 4
System requirements and supported platforms 6
UltraLite C++ Component architecture 7

II. Application Development 9

Common Features of UltraLite C/C++ Interfaces 11
Understanding the SQL Communications Area 12
Creating databases 13
Developing Applications Using the UltraLite C++ API 15
Using the UltraLite namespace 16
Connecting to a database 17
Accessing data using SQL 19
Accessing data with the Table API 23
Managing transactions 29
Accessing schema information 30
Handling errors 31
Authenticating users 32
Encrypting data 33
Synchronizing data 34
Compiling and linking your application 35
Developing Applications Using Embedded SQL 37
Introduction to embedded SQL development 38
Example of embedded SQL 39
Initializing the SQL Communications Area 41

Connecting to a database 43
Using host variables 45
Fetching data 55
Authenticating users 59
Encrypting data 61
Adding synchronization to your application 63
Building embedded SQL applications 70
Developing UltraLite Applications for the Palm OS 73
Introduction to Palm OS development 74
Developing UltraLite applications with Metrowerks CodeWarrior 75
Maintaining state in UltraLite Palm applications 79
Registering the Palm creator ID 81
Adding HotSync synchronization to Palm applications 82
Adding TCP/IP, HTTP, or HTTPS synchronization to Palm applications 84
Deploying Palm applications 85
Developing UltraLite Applications for Symbian OS 87
Introduction to Symbian OS development 88
Developing applications using CodeWarrior for Symbian 90
Developing UltraLite Applications for Windows CE 93
Introduction to Windows CE development 94
Building the CustDB sample application 96
Storing persistent data 98
Deploying Windows CE applications 99
Synchronization on Windows CE 102

III. Tutorials 105

Tutorial: Build an Application Using the C++ API 107
Introduction to UltraLite C++ development 108
Lesson 1: Create database and connect to database 109
Lesson 2: Insert data into the database 113
Lesson 3: Select and list rows from the table 114
Lesson 4: Add synchronization to your application 116
Code listing for tutorial 118
Tutorial: Build an Application Using Embedded SQL 121
Introduction to embedded SQL development tutorial 122

 Copyright © 2007, iAnywhere Solutions, Inc.

Lesson 1: Create the UltraLite database 123
Lesson 2: Configure eMbedded Visual C++ 124
Lesson 3: Write an embedded SQL source file 126
Lesson 4: Build the embedded SQL UltraLite tutorial application 132
Lesson 5: Add synchronization to your application 133
Tutorial: Build an Application Using ODBC 135
Introduction to UltraLite ODBC 136
Lesson 1: Getting started 137
Lesson 2: Create an UltraLite database 139
Lesson 3: Connect to the database 140
Lesson 4: Insert data into the database 142
Lesson 5: Query the database 143

IV. API Reference 145

UltraLite C/C++ Common API Reference 147
Introduction to UltraLite C/C++ common API 148
Callback function for ULRegisterErrorCallback 149
MLFileTransfer function 151
ULCreateDatabase function 154
ULEnableEccSyncEncryption function 156
ULEnableFileDB function (deprecated) 157
ULEnableFIPSStrongEncryption function 158
ULEnableHttpSynchronization function 159
ULEnableHttpsSynchronization function 160
ULEnablePalmRecordDB function (deprecated) 161
ULEnableRsaFipsSyncEncryption function 162
ULEnableRsaSyncEncryption function 163
ULEnableStrongEncryption function 164
ULEnableTcpipSynchronization function 165
ULEnableTlsSynchronization function 166
ULEnableUserAuthentication function (deprecated) 167
ULEnableZlibSyncCompression function 168
ULInitDatabaseManager 169
ULInitDatabaseManagerNoSQL 170
ULRegisterErrorCallback function 171

Copyright © 2007, iAnywhere Solutions, Inc.

Macros and compiler directives for UltraLite C/C++ applications 173
UltraLite C++ Component API 177
ul_synch_info_a struct 179
ul_synch_info_w2 struct 186
ul_synch_result struct 193
ul_synch_stats struct 196
ul_synch_status struct 198
ULSqlca class 200
ULSqlcaBase class 202
ULSqlcaWrap class 207
UltraLite_Connection class 209
UltraLite_Connection_iface class 210
UltraLite_Cursor_iface class 223
UltraLite_DatabaseManager class 230
UltraLite_DatabaseManager_iface class 231
UltraLite_DatabaseSchema class 234
UltraLite_DatabaseSchema_iface class 235
UltraLite_IndexSchema class 239
UltraLite_IndexSchema_iface class 240
UltraLite_PreparedStatement class 245
UltraLite_PreparedStatement_iface class 246
UltraLite_ResultSet class 250
UltraLite_ResultSet_iface class 251
UltraLite_ResultSetSchema class 252
UltraLite_RowSchema_iface class 253
UltraLite_SQLObject_iface class 258
UltraLite_StreamReader class 260
UltraLite_StreamReader_iface class 261
UltraLite_StreamWriter class 264
UltraLite_Table class 265
UltraLite_Table_iface class 266
UltraLite_TableSchema class 272
UltraLite_TableSchema_iface class 273
ULValue class 283
Embedded SQL API Reference 301

 Copyright © 2007, iAnywhere Solutions, Inc.

Introduction to embedded SQL API 303
db_fini function 304
db_init function 305
db_start_database function 306
db_stop_database function 307
ULChangeEncryptionKey function 308
ULCheckpoint function 309
ULClearEncryptionKey function 310
ULCountUploadRows function 311
ULDropDatabase function 312
ULGetDatabaseID function 313
ULGetDatabaseProperty function 314
ULGetLastDownloadTime function 315
ULGetSynchResult function 316
ULGlobalAutoincUsage function 318
ULGrantConnectTo function 319
ULHTTPSStream function (deprecated) 320
ULHTTPStream function (deprecated) 321
ULInitSynchInfo function 322
ULIsSynchronizeMessage function 323
ULResetLastDownloadTime function 324
ULRetrieveEncryptionKey function 325
ULRevokeConnectFrom function 326
ULRollbackPartialDownload function 327
ULSaveEncryptionKey function 328
ULSetDatabaseID function 329
ULSetDatabaseOptionString function 330
ULSetDatabaseOptionULong 331
ULSetSynchInfo function 332
ULSocketStream function (deprecated) 333
ULSynchronize function 334
UltraLite ODBC API Reference 335
Introduction to UltraLite ODBC API 336
SQLAllocHandle function 337
SQLBindCol function 338

SQLBindParameter function 339
SQLConnect function 340
SQLDescribeCol function 341
SQLDisconnect function 342
SQLEndTran function 343
SQLExecDirect function 344
SQLExecute function 345
SQLFetch function 346
SQLFetchScroll function 347
SQLFreeHandle function 348
SQLGetCursorName function 349
SQLGetData function 350
SQLGetDiagRec function 351
SQLGetInfo function 352
SQLNumResultCols function 353
SQLPrepare function 354
SQLRowCount function 355
SQLSetConnectionName function 356
SQLSetCursorName function 357
SQLSetSuspend function 358
SQLSynchronize function 359
Index 361

 Copyright © 2007, iAnywhere Solutions, Inc.

About This Manual

Subject

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

Audience

This manual is intended for C and C++ application developers, who want to take advantage of the performance, resource efficiency, robustness, and security of an UltraLite relational database for data storage and synchronization.

SQL Anywhere documentation

This book is part of the SQL Anywhere documentation set. This section describes the books in the documentation set and how you can use them.

The SQL Anywhere documentation

The complete SQL Anywhere documentation is available in two forms: an online form that combines all books, and as separate PDF files for each book. Both forms of the documentation contain identical information and consist of the following books:

- **SQL Anywhere 10 - Introduction** This book introduces SQL Anywhere 10—a product that provides data management and data exchange technologies, enabling the rapid development of database-powered applications for server, desktop, mobile, and remote office environments.
- **SQL Anywhere 10 - Changes and Upgrading** This book describes new features in SQL Anywhere 10 and in previous versions of the software, as well as upgrade instructions.
- **SQL Anywhere Server - Database Administration** This book covers material related to running, managing, and configuring SQL Anywhere databases. It describes database connections, the database server, database files, backup procedures, security, high availability, and replication with Replication Server, as well as administration utilities and options.
- **SQL Anywhere Server - SQL Usage** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- **SQL Anywhere Server - SQL Reference** This book provides a complete reference for the SQL language used by SQL Anywhere. It also describes the SQL Anywhere system views and procedures.
- **SQL Anywhere Server - Programming** This book describes how to build and deploy database applications using the C, C++, and Java programming languages, as well as Visual Studio .NET. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by these tools.
- **SQL Anywhere 10 - Error Messages** This book provides a complete listing of SQL Anywhere error messages together with diagnostic information.
- **MobiLink - Getting Started** This manual introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.
- **MobiLink - Server Administration** This manual describes how to set up and administer MobiLink server-side utilities and functionality.
- **MobiLink - Client Administration** This manual describes how to set up, configure, and synchronize MobiLink clients. MobiLink clients can be SQL Anywhere or UltraLite databases.
- **MobiLink - Server-Initiated Synchronization** This manual describes MobiLink server-initiated synchronization, a feature of MobiLink that allows you to initiate synchronization or other remote actions from the consolidated database.

Copyright © 2007, iAnywhere Solutions, Inc.

- **QAnywhere** This manual describes QAnywhere, which is a messaging platform for mobile and wireless clients as well as traditional desktop and laptop clients.
- **SQL Remote** This book describes the SQL Remote data replication system for mobile computing, which enables sharing of data between a SQL Anywhere consolidated database and many SQL Anywhere remote databases using an indirect link such as email or file transfer.
- **SQL Anywhere 10 - Context-Sensitive Help** This manual contains the context-sensitive help for the Connect dialog, the Query Editor, the MobiLink Monitor, MobiLink Model mode, the SQL Anywhere Console utility, the Index Consultant, and Interactive SQL.
- **UltraLite - Database Management and Reference** This manual introduces the UltraLite database system for small devices.
- **UltraLite - AppForge Programming** This manual describes UltraLite for AppForge. With UltraLite for AppForge you can develop and deploy database applications to handheld, mobile, or embedded devices, running Palm OS, Symbian OS, or Windows CE.
- **UltraLite - .NET Programming** This manual describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.
- **UltraLite - M-Business Anywhere Programming** This manual describes UltraLite for M-Business Anywhere. With UltraLite for M-Business Anywhere you can develop and deploy web-based database applications to handheld, mobile, or embedded devices, running Palm OS, Windows CE, or Windows XP.
- **UltraLite - C and C++ Programming** This manual describes UltraLite C and C++ programming interfaces. With UltraLite, you can develop and deploy database applications to handheld, mobile, or embedded devices.

Documentation formats

SQL Anywhere provides documentation in the following formats:

- **Online documentation** The online documentation contains the complete SQL Anywhere documentation, including the books and the context-sensitive help for SQL Anywhere tools. The online documentation is updated with each maintenance release of the product, and is the most complete and up-to-date source of documentation.

  To access the online documentation on Windows operating systems, choose Start ► Programs ► SQL Anywhere 10 ► Online Books. You can navigate the online documentation using the HTML Help table of contents, index, and search facility in the left pane, as well as using the links and menus in the right pane.

  To access the online documentation on Unix operating systems, see the HTML documentation under your SQL Anywhere installation or on your installation CD.
- **PDF files** The complete set of SQL Anywhere books is provided as a set of Adobe Portable Document Format (pdf) files, viewable with Adobe Reader.

Copyright © 2007, iAnywhere Solutions, Inc.

On Windows, the PDF books are accessible from the online documentation via the PDF link at the top of each page, or from the Windows Start menu (Start ► Programs ► SQL Anywhere 10 ► Online Books - PDF Format).

On Unix, the PDF books are available on your installation CD.

Copyright © 2007, iAnywhere Solutions, Inc.

Documentation conventions

This section lists the typographic and graphical conventions used in this documentation.

Syntax conventions

The following conventions are used in the SQL syntax descriptions:

- **Keywords** All SQL keywords appear in uppercase, like the words ALTER TABLE in the following example:

  ALTER TABLE [*owner*.]*table-name*

- **Placeholders** Items that must be replaced with appropriate identifiers or expressions are shown like the words *owner* and *table-name* in the following example:

  ALTER TABLE [*owner*.]*table-name*

- **Repeating items** Lists of repeating items are shown with an element of the list followed by an ellipsis (three dots), like *column-constraint* in the following example:

  ADD *column-definition* [*column-constraint*, …]

  One or more list elements are allowed. In this example, if more than one is specified, they must be separated by commas.

- **Optional portions** Optional portions of a statement are enclosed by square brackets.

  RELEASE SAVEPOINT [*savepoint-name*]

  These square brackets indicate that the *savepoint-name* is optional. The square brackets should not be typed.

- **Options** When none or only one of a list of items can be chosen, vertical bars separate the items and the list is enclosed in square brackets.

  [**ASC** | **DESC**]

  For example, you can choose one of ASC, DESC, or neither. The square brackets should not be typed.

- **Alternatives** When precisely one of the options must be chosen, the alternatives are enclosed in curly braces and a bar is used to separate the options.

  [**QUOTES** { **ON** | **OFF** }]

  If the QUOTES option is used, one of ON or OFF must be provided. The brackets and braces should not be typed.

Operating system conventions

- **Windows** The Microsoft Windows family of operating systems for desktop and laptop computers. The Windows family includes Windows Vista and Windows XP.

- **Windows CE** Platforms built from the Microsoft Windows CE modular operating system, including the Windows Mobile and Windows Embedded CE platforms.

  Windows Mobile is built on Windows CE. It provides a Windows user interface and additional functionality, such as small versions of applications like Word and Excel. Windows Mobile is most commonly seen on mobile devices.

  Limitations or variations in SQL Anywhere are commonly based on the underlying operating system (Windows CE), and seldom on the particular variant used (Windows Mobile).

- **Unix** Unless specified, Unix refers to both Linux and Unix platforms.

File name conventions

The documentation generally adopts Windows conventions when describing operating system dependent tasks and features such as paths and file names. In most cases, there is a simple transformation to the syntax used on other operating systems.

- **Directories and path names** The documentation typically lists directory paths using Windows conventions, including colons for drives and backslashes as a directory separator. For example,

  ```
  MobiLink\redirector
  ```

  On Unix, Linux, and Mac OS X, you should use forward slashes instead. For example,

  ```
  MobiLink/redirector
  ```

  If SQL Anywhere is used in a multi-platform environment you must be aware of path name differences between platforms.

- **Executable files** The documentation shows executable file names using Windows conventions, with the suffix *.exe*. On Unix, Linux, and Mac OS X, executable file names have no suffix. On NetWare, executable file names use the suffix *.nlm*.

  For example, on Windows, the network database server is *dbsrv10.exe*. On Unix, Linux, and Mac OS X, it is *dbsrv10*. On NetWare, it is *dbsrv10.nlm*.

- **install-dir** The installation process allows you to choose where to install SQL Anywhere, and the documentation refers to this location using the convention *install-dir*.

  After installation is complete, the environment variable SQLANY10 specifies the location of the installation directory containing the SQL Anywhere components (*install-dir*). SQLANYSH10 specifies the location of the directory containing components shared by SQL Anywhere with other Sybase applications.

  For more information on the default location of *install-dir*, by operating system, see "SQLANY10 environment variable" [*SQL Anywhere Server - Database Administration*].

- **samples-dir** The installation process allows you to choose where to install the samples that are included with SQL Anywhere, and the documentation refers to this location using the convention *samples-dir*.

Copyright © 2007, iAnywhere Solutions, Inc.

After installation is complete, the environment variable SQLANYSAMP10 specifies the location of the directory containing the samples (*samples-dir*). From the Windows Start menu, choosing Programs ► SQL Anywhere 10 ► Sample Applications and Projects opens a Windows Explorer window in this directory.

For more information on the default location of *samples-dir*, by operating system, see "Samples directory" [*SQL Anywhere Server - Database Administration*].

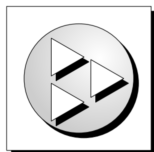
- **Environment variables** The documentation refers to setting environment variables. On Windows, environment variables are referred to using the syntax *%envvar%*. On Unix, Linux, and Mac OS X, environment variables are referred to using the syntax *$envvar* or *${envvar}*.

  Unix, Linux, and Mac OS X environment variables are stored in shell and login startup files, such as *.cshrc* or *.tcshrc*.

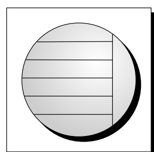
Graphic icons

The following icons are used in this documentation.

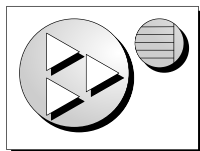
- A client application.



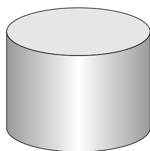
- A database server, such as SQL Anywhere.



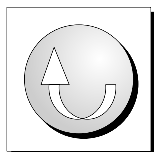
- An UltraLite application.



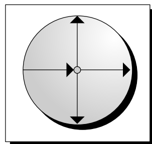
- A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



- Replication or synchronization middleware. These assist in sharing data among databases. Examples are the MobiLink server and the SQL Remote Message Agent.



- A Sybase Replication Server



- A programming interface.

Copyright © 2007, iAnywhere Solutions, Inc.

Finding out more and providing feedback

Finding out more

Additional information and resources, including a code exchange, are available at the iAnywhere Developer Network at http://www.ianywhere.com/developer/.

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 detailed information about your problem, including the build number of your version of SQL Anywhere. You can find this information by entering **dbeng10 -v** at a command prompt.

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

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

Feedback

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

You can email comments and suggestions to the SQL Anywhere documentation team at iasdoc@ianywhere.com. Although we do not reply to emails sent to that address, we read all suggestions with interest.

In addition, you can provide feedback on the documentation and the software through the newsgroups listed above.

Copyright © 2007, iAnywhere Solutions, Inc.

Part I. Introduction

This part introduces UltraLite for C/C++ programmers. The C/C++ language interface can be used to write UltraLite applications for Palm OS, Windows CE, Symbian OS, and Windows Desktop platforms.

CHAPTER 1

Introduction to UltraLite for C/C++ Developers

Contents

UltraLite and the C/C++ programming languages 4
System requirements and supported platforms 6
UltraLite C++ Component architecture 7

Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite and the C/C++ programming languages

The C and C++ interface provides the following benefits for UltraLite developers targeting small devices:

- A small, high-performance database store.
- The power, efficiency, and flexibility of the C or C++ language.
- The ability to deploy an application on Windows CE, Symbian OS, Palm OS, and Windows desktop platforms.

For more information about the features of UltraLite databases, see "Creating and Configuring UltraLite Databases" [*UltraLite - Database Management and Reference*].

UltraLite developers that use C++ have two options available:

- The UltraLite C++ API.
- The ODBC programming interface (component interface).

UltraLite developers that use C must use Embedded SQL or the ODBC programming interface.

Developing embedded SQL applications

When developing embedded SQL applications, you mix SQL statements with standard C or C++ source code. In order 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. Create your UltraLite database.
2. Write your source code in an embedded SQL source file, which typically has extension *.sqc*.

   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 SELECT price, prod_name
               INTO :cost, :pname
               FROM ULProduct
               WHERE prod_id= :pid;
   if((SQLCODE==SQLE_NOTFOUND)||(SQLCODE<0)) {
           return(-1);
   }
   ```
3. Preprocess the *.sqc* files.

   SQL Anywhere includes a SQL preprocessor (sqlpp), which reads the *.sqc* 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.

 Copyright © 2007, iAnywhere Solutions, Inc.

You must link the files with the UltraLite runtime library.

For more information about embedded SQL development, see "Building embedded SQL applications" on page 70.

System requirements and supported platforms

Development platforms

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

- A Microsoft Windows desktop as a development platform.
- A supported C/C++ compiler.

Target platforms

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

- Windows CE 3.0 or later
- Palm OS 4.0 or later
- Symbian OS 7.0 or 8.0 or later.

For more information about supported target platforms, see UltraLite Deployment Option for SQL Anywhere.

 Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite C++ Component architecture

The UltraLite C++ component interface is defined in the *uliface.h* header file. The following list describes some of the commonly used objects:

- **DatabaseManager** Create one DatabaseManager object for each application.
- **Connection** Represents a connection to an UltraLite database. You can create one or more Connection objects.
- **Table** Provides access to the data in the database.
- **PreparedStatement, ResultSet, and ResultSetSchema** Create Dynamic SQL statements, make queries and execute INSERT, UPDATE, and DELETE statements, and attain programmatic control over database result sets.
- **SyncParms** Synchronize your UltraLite database with a MobiLink server.

For more information about accessing the API reference, see "UltraLite C++ API Reference" on page 177.

Part II. Application Development

This part provides development notes for UltraLite C/C++ programmers.

CHAPTER 2

Common Features of UltraLite C/C++ Interfaces

Contents

Understanding the SQL Communications Area 12
Creating databases 13

Understanding the SQL Communications Area

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

All UltraLite C/C++ interfaces share the same basic data structure for marshaling data between the UltraLite runtime and your application. This data structure is the SQL Communications Area or SQLCA. Each SQLCA has a current connection, and separate threads cannot share a common SQLCA.

Your application code must carry out the following tasks before connecting to a database:

- Initialize a SQLCA. This prepares your application for communication with the UltraLite runtime.
- Register your error callback function.
- Start a database. This operation can be carried out as part of opening the connection.

The following functions are equivalent ways of carrying out these tasks.

Task	Interface	Function
Initialize SQLCA	Embedded SQL	db_init
	C++	ULSqlca::Initialize
Initialize SQLCA and start database	Embedded SQL	db_init db_start_database
	C++	The database is started as part of the connection function in UltraLite_DatabaseManager

 Copyright © 2007, iAnywhere Solutions, Inc.

Creating databases

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

- ♦ The Create Database wizard in Sybase Central.
- ♦ A command line utility like ulcreate or ulinit.
- ♦ Calling the ULCreateDataBase function.

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

The ulcreate utility creates an empty database that does not have any tables defined. Applications that create a database by calling ULCreateDatabase need to also execute SQL CREATE statements to create tables and index definitions.

Name the database explicitly

Different interfaces may use different default file names for the database. If you are mixing the interfaces, it is best to always explicitly name the database when creating or connecting. You can do this using the DBN= connection parameter. See "UltraLite DBN connection parameter" [*UltraLite - Database Management and Reference*].

CHAPTER 3

Developing Applications Using the UltraLite C++ API

Contents

Using the UltraLite namespace 16
Connecting to a database 17
Accessing data using SQL 19
Accessing data with the Table API 23
Managing transactions 29
Accessing schema information 30
Handling errors 31
Authenticating users 32
Encrypting data 33
Synchronizing data 34
Compiling and linking your application 35

Using the UltraLite namespace

The UltraLite C++ interface provides a set of classes with names that are prefixed by UltraLite_ (for example, UltraLite_Connection and UltraLite_DatabaseManager). Most of the functions for each of these classes implement a function from an underlying interface with the string _iface appended to it. For example, the UltraLite_Connection class implements functions from UltraLite_Connection_iface.

When you explicitly use the UltraLite namespace, you can use a shorter name to refer to each class. Instead of declaring a connection as an UltraLite_Connection object, you can declare it as a Connection object if you are using the UltraLite namespace:

```
using namespace UltraLite;
ULSqlca sqlca;
sqlca.Initialize();
DatabaseManager * dbMgr = ULInitDatabaseManager(sqlca);
Connection *  conn = UL_NULL;
```

As a result of this architecture, code samples in this chapter use types such as DatabaseManager, Connection, and TableSchema, but links for more information may direct you to UltraLite_DatabaseManager_iface, UltraLite_Connection_iface, and UltraLite_TableSchema_iface, respectively.

 Copyright © 2007, iAnywhere Solutions, Inc.

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.

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

Properties of the Connection object

- **Commit behavior** In the UltraLite C++ API, there is no AutoCommit mode. Each transaction must be followed by a Conn->Commit() statement.

  See "Managing transactions" on page 29.

- **User authentication** You can change the user ID and password for the application (from the default values of DBA and sql, respectively) by using methods to grant and revoke connection permissions. Each database can have a maximum of four user IDs.

  See "Authenticating users" on page 32.

- **Synchronization** You can synchronize an UltraLite database with a consolidated database by using methods of the Connection object.

  See "Synchronizing data" on page 34.

- **Tables** UltraLite database tables are accessed using methods of the Connection object.

  See "Accessing data with the Table API" on page 23.

- **Prepared statements** Methods are provided to handle the execution of SQL statements.

  See "Accessing data using SQL" on page 19 and "UltraLite_PreparedStatement class" on page 245.

Connecting to an UltraLite database

♦ To connect to an UltraLite database

1. Use the UltraLite namespace.

   Using the UltraLite namespace allows you to use simple names for classes in the C++ interface.

   ```
   using namespace UltraLite;
   ```

2. Create and initialize a DatabaseManager object and an UltraLite SQL communications area (ULSqlca). The ULSqlca is a structure that handles communication between the application and the database.

   The DatabaseManager object is at the root of the object hierarchy. You create only one DatabaseManager object per application. It is often best to declare the DatabaseManager object as global to the application.

   ```
   ULSqlca sqlca;
   sqlca.Initialize();
   DatabaseManager * dbMgr = ULInitDatabaseManager(sqlca);
   ```

Copyright © 2007, iAnywhere Solutions, Inc.

If the application does not require SQL support and directly links the UltraLite runtime, the application can call ULInitDatabaseManagerNoSQL to initialize the ULSqlca. This variant reduces the size of the application.

See "UltraLite_DatabaseManager_iface class" on page 231.

3. Open a connection to an existing database, or create a new database if the specified database file does not exist. See "OpenConnection function" on page 232.

   UltraLite applications can be deployed with an initial empty database or the application can create the UltraLite database if it does not already exist. Deploying an initial database is the simplest solution; otherwise, the application must call the ULCreateDatabase function to create the database and must create all the tables the application requires. See "ULCreateDatabase function" on page 154.

```
COnnection * conn = dbMgr->OpenConnection( sqlca, UL_TEXT
("dbf=mydb.udb") );
if( sqlca.GetSQLCode() ==
  SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
    printf( "Open failed with sql code: %d.\n" , sqlca.GetSQLCode() );
  }
}
```

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.

 Copyright © 2007, iAnywhere Solutions, Inc.

Accessing data using SQL

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

For more information about using the Table API, see "Accessing data with the Table API" on page 23.

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 Statement Reference" [*UltraLite - Database Management and Reference*].

Data manipulation: Insert, Delete, and Update

With UltraLite, you can perform SQL data manipulation by using the ExecuteStatement method (a member of the PreparedStatement class).

See the "UltraLite_PreparedStatement class" on page 245.

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 PreparedStatement.

   ```
   PreparedStatement * prepStmt;
   ```

   See "PrepareStatement function" on page 217.

2. Assign a SQL statement to the PreparedStatement object.

   ```
   prepStmt = conn->PrepareStatement( UL_TEXT("INSERT INTO MyTable(MyColumn)
   values (?)") );
   ```

3. Assign input parameter values for the statement.

   The following code shows a string parameter.

   ```
   prepStmt->SetParameter( 1, UL_TEXT("newValue") );
   ```

4. Execute the prepared statement.

   The return value indicates the number of rows affected by the statement.

```
ul_s_long rowsInserted;
rowsInserted = prepStmt->ExecuteStatement();
```

5. Commit the change.

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

♦ To delete a row

1. Declare a PreparedStatement.

```
PreparedStatement * prepStmt;
```

2. Assign a SQL statement to the PreparedStatement object.

```
ULValue sqltext(
    );
prepStmt = conn->PrepareStatement( UL_TEXT("DELETE FROM MyTable WHERE
MyColumn = ?") );
```

3. Assign input parameter values for the statement.

```
prepStmt->SetParameter( 1, UL_TEXT("deleteValue") );
```

4. Execute the statement.

```
ul_s_long rowsDeleted;
rowsDeleted = prepStmt->ExecuteStatement();
```

5. Commit the change.

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

♦ To update a row

1. Declare a PreparedStatement.

```
PreparedStatement * prepStmt;
```

2. Assign a statement to the PreparedStatement object.

```
prepStmt = conn->PrepareStatement(
    UL_TEXT("UPDATE MyTable SET MyColumn1 = ? WHERE MyColumn1 = ?") );
```

3. Assign input parameter values for the statement.

```
prepStmt->SetParameter( 1, UL_TEXT("newValue") );
prepStmt->SetParameter( 2, UL_TEXT("oldValue") );
```

4. Execute the statement.

```
ul_s_long rowsUpdated;
rowsUpdated = prepStmt->ExecuteStatement();
```

5. Commit the change.

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

 Copyright © 2007, iAnywhere Solutions, Inc.

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 "UltraLite_PreparedStatement_iface class" on page 246.

♦ To execute a SELECT statement

1. Create a prepared statement object.

```
PreparedStatement * prepStmt =
    conn->PrepareStatement( UL_TEXT("SELECT MyColumn FROM MyTable") );
```

2. Execute the statement.

   In the following code, the result of the SELECT query contains a string, which is output to the console.

```
#define MAX_NAME_LEN     100
ULValue val;
ResultSet * rs = prepStmt->ExecuteQuery();
while( rs->Next() ){
   char mycol[ MAX_NAME_LEN ];
   val = rs->Get( 1 );
   val.GetString( mycol, MAX_NAME_LEN );
   printf( "mycol= %s\n", mycol );
}
```

Navigating SQL result sets

You can navigate through a result set using methods associated with the ResultSet object.

The result set object 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 position of the cursor 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.

See "UltraLite_ResultSet_iface class" on page 251.

Result set schema description

The ResultSet->GetSchema 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 ResultSet.GetSchema method to display schema information in a console window.

```
ResultSetSchema * rss = rs->GetSchema();
ULValue val;
char name[ MAX_NAME_LEN ];

for( int i = 1;
     i <= rss->GetColumnCount();
     i++ ){
   val = rss->GetColumnName( i );
   val.GetString( name, MAX_NAME_LEN );
   printf( "id= %d, name= %s \n", i, name );
}
```

See “GetSchema function” on page 214.

 Copyright © 2007, iAnywhere Solutions, Inc.

Accessing data with the Table API

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

For more information about accessing data by executing SQL statements, see "Accessing data using SQL" on page 19.

This section explains how to perform the following tasks using the Table API:

- 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.

Navigating the rows of a table

The UltraLite C++ API provides you with a number of methods to navigate a table to perform a wide range of navigation tasks.

The table 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 position of the cursor 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.

See "UltraLite_Table_iface class" on page 266.

Example

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

```
Table * tbl = conn->openTable( "MyTable" );
ul_column_num colID =
   tbl->GetSchema()->GetColumnID( "MyColumn" );
```

```
while ( tbl->Next() ){
   char buffer[ MAX_NAME_LEN ];
   ULValue colValue = tbl->Get(colID);
   colValue.GetString(buffer, MAX_NAME_LEN);
   printf( "%s\n", buffer );
}
```

You expose the rows of the table to the application when you open the table 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 code fragment moves to the first row of the MyTable table as ordered by the ix_col index.

```
ULValue table_name( UL_TEXT("MyTable") )
ULValue index_name( UL_TEXT("ix_col") )
Table * tbl =
   conn->OpenTableWithIndex( table_name, index_name );
```

See "UltraLite_Table_iface class" on page 266.

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 Table 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 Table 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.

 Copyright © 2007, iAnywhere Solutions, Inc.

Retrieving column values

The Table object provides a set of methods for retrieving column values. These methods take the column ID as the argument.

The following code fragment retrieves the value of the lname column, which is a character string.

```
ULValue val;
char lname[ MAX_NAME_LEN ];
val = tbl->Get( UL_TEXT("lname") );
val.GetString( lname, MAX_NAME_LEN );
```

The following code retrieves the value of the cust_id column, which is an integer.

```
int id = tbl->Get( UL_TEXT("cust_id") );
```

Modifying column values

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

For example, the following code sets the value of the lname column to Kaminski.

```
ULValue lname_col( UL_TEXT("fname") );
ULValue v_lname( UL_TEXT("Kaminski") );
tbl->Set( lname_col, v_lname );
```

By setting column values, you do not directly alter the data in the database. You can assign values to the properties, 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();
id = tbl.Get( 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 Table 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 Table 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 Table object was opened.

♦ To search for a row

1. Enter find or lookup mode.

   Call a method on the table object to set the mode. For example, the following code enters find mode.

   ```
   tbl.FindBegin();
   ```

2. Set the search values.

   Set the search values in the current row. Setting these values only affects the buffer holding the current row, not the database. For example, the following code fragment sets the value in the buffer to Kaminski.

   ```
   ULValue lname_col = t->GetSchema()->GetColumnID( UL_TEXT("lname") );
   ULValue v_lname( UL_TEXT("Kaminski") );
   tbl.Set( lname_col, v_lname );
   ```

3. Search for the row.

   Call the appropriate method to carry out the search. For example, the following code looks for the first row that exactly matches the specified value in the current index.

   For multi-column indexes, a value for the first column is always used, but you can omit the other columns.

   ```
   tCustomer.FindFirst();
   ```

4. Search for the next instance of the row.

   Call the appropriate method to carry out the search. For a find operation, FindNext locates the next instance of the parameters in the index. For a lookup, MoveNext locates the next instance.

See “UltraLite_Table_iface class” on page 266.

Updating rows

The following procedure updates a row.

♦ To update a row

1. Move to the row you want to update.

 Copyright © 2007, iAnywhere Solutions, Inc.

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.Set( UL_TEXT("id"), 3 );
   ```

4. Execute the Update.

   ```
   tbl.Update();
   ```

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 you commit them using `conn->Commit()`. See “Managing transactions” on page 29.

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. The order of row insertion into the table has no significance since data is always inserted in the database according to the index.

Example

The following code fragment inserts a new row.

```
tbl.InsertBegin();
tbl.Set( UL_TEXT("id"), 3 );
tbl.Set( UL_TEXT("lname"), "Carlo" );
tbl.Insert();
tbl.Commit();
```

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.

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 Table.Delete method.

```
tbl.Delete();
```

 Copyright © 2007, iAnywhere Solutions, Inc.

Managing transactions

The UltraLite C++ API does not support AutoCommit mode. Transactions must be explicitly committed or rolled back.

♦ To commit a transaction

- Execute a Conn->Commit statement.

  See “Commit function” on page 211.

♦ To roll back a transaction

- Execute a Conn->Rollback statement.

  See “Rollback function” on page 218.

For more information about transaction management in UltraLite, see “UltraLite transaction processing and isolation levels” [*UltraLite - Database Management and Reference*].

Accessing schema information

The objects in the API represent tables, columns, indexes, and synchronization publications. Each object has a GetSchema method that provides access to information about the structure of that object.

You cannot modify the schema through the API. You can only retrieve information about the schema.

You can access the following schema objects and information:

- **DatabaseSchema** Exposes the number and names of the tables in the database, as well as global properties such as the format of dates and times.

  To obtain a DatabaseSchema object, use Conn->GetSchema.

  See "GetSchema function" on page 214.

- **TableSchema** Exposes the number and names of the columns and indexes for this table.

  To obtain a TableSchema object, use tbl->GetSchema.

  See "GetSchema function" on page 214.

- **IndexSchema** Returns information about the column in the index. As an index has no data directly associated with it there is no separate Index class, just the IndexSchema class.

  To obtain a IndexSchema object, call the table_schema->GetIndexSchema or table_schema->GetPrimaryKey methods.

  See "UltraLite_Table_iface class" on page 266.

 Copyright © 2007, iAnywhere Solutions, Inc.

Handling errors

You should check for errors after each database operation by using methods of the ULSqlca object. For example, LastCodeOK checks if the operation was successful, while GetSQLCode returns the numerical value of the SQLCode. For more information about the meaning of these values, see "Error messages sorted by Sybase error code" [*SQL Anywhere 10 - Error Messages*].

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 107.

For a sample callback function, see "Callback function for ULRegisterErrorCallback" on page 149 and "ULRegisterErrorCallback function" on page 171.

For a list of error codes thrown by the UltraLite C++ API, see "Error messages sorted by Sybase error code" [*SQL Anywhere 10 - Error Messages*].

Copyright © 2007, iAnywhere Solutions, Inc.

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.

♦ 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 function" on page 215.

♦ 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 function" on page 217.

Copyright © 2007, iAnywhere Solutions, Inc.

Encrypting data

You can chose 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 creation-time database properties 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.

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

See "UltraLite DBKEY connection parameter" [*UltraLite - Database Management and Reference*].

Obfuscation

To obfuscate the database, specify `obfuscate=1` as a database creation parameter.

See "UltraLite security considerations" [*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. In all cases, you use methods and properties of the connection object to control synchronization.

For more information about synchronization, see "UltraLite Clients" [*MobiLink - Client Administration*].

For more information about the ul_sync_info structure used for synchronization, see "ul_synch_info_a struct" on page 179 or "ul_synch_info_w2 struct" on page 186 depending whether you are using ASCII or wide characters.

For more information about synchronization parameters, see "Synchronization parameters for UltraLite" [*MobiLink - Client Administration*].

Copyright © 2007, iAnywhere Solutions, Inc.

Compiling and linking your application

A set of runtime libraries is available for some platforms when using the UltraLite C++ API. These include, for Windows CE and Windows, a database engine that permits multi-process access to the same database.

The runtime libraries are provided in the *install-dir\ultralite\palm*, *install-dir\ultralite\ce*, and *install-dir \ultralite\win32* directories.

Runtime libraries for Palm OS

The following libraries are supplied for applications on the Palm OS.

- **ulrt.lib** A static library. This library is located in *install-dir\ultralite\palm\68k\lib\cw*.
- **ulbase.lib** A library containing extra functions that can not be provided in a separate dynamic link library (DLL). C/C++ applications should link against this library to ensure access to UltraLite features.

Runtime libraries for Windows CE

The Windows CE libraries are in the *install-dir\ultralite\ce\platform* directories, where *platform* is one of *arm*, *386*, *arm.50*, or *armt*.

Dynamic libraries are provided for Windows CE:

- **ulbase.lib** A library containing extra functions that can not be provided in a separate dynamic link library (DLL). C/C++ applications should link against this library to ensure access to UltraLite features.
- **ulrt10.dll** To use this library, link your application against the import library, *install-dir\ultralite\ce \platform\lib\ulimp.lib*.

  When linking against this library, be sure to specify the following compilation options:

  ```
  /DUNICODE /DUL_USE_DLL
  ```

- **ulrt.lib** This library is located in *install-dir\ultralite\ce\platform\lib*.

  When linking against this library, be sure to specify the following compilation option:

  ```
  /DUNICODE
  ```

- **ulrtc.lib** A Unicode character set static library for use with the UltraLite engine for multi-process access to an UltraLite database. This is located in *install-dir\ultralite\ce\platform\lib*.

  When linking against this library, be sure to specify the following compilation option:

  ```
  /DUNICODE
  ```

Runtime libraries for Windows 32-bit desktops

The *install-dir\ultralite\win32\386* directory contains libraries for supported Windows operating systems other than Windows CE. This includes the following:

Copyright © 2007, iAnywhere Solutions, Inc.

- **ulbase.lib** A library containing functions that can not be provided in a separate dynamic link library (DLL). C/C++ applications should link against this library to ensure access to UltraLite features.

- **ulrt10.dll** An ANSI character set dynamic link library. To use this library, link your application against the import library, *install-dir\ultralite\win32\386\ulimp.lib*.

  When linking against this library, be sure to specify the following compilation option:

  ```
  /DUL_USE_DLL
  ```

 Copyright © 2007, iAnywhere Solutions, Inc.

CHAPTER 4

Developing Applications Using Embedded SQL

Contents

Introduction to embedded SQL development 38
Example of embedded SQL 39
Initializing the SQL Communications Area 41
Connecting to a database 43
Using host variables 45
Fetching data 55
Authenticating users 59
Encrypting data 61
Adding synchronization to your application 63
Building embedded SQL applications 70

Introduction to embedded SQL development

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

For an overview of the UltraLite C/C++ development process, see "Introduction to UltraLite for C/C++ Developers" on page 3.

For embedded SQL reference information, see "Embedded SQL API Reference" on page 301.

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

Copyright © 2007, iAnywhere Solutions, Inc.

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 *.sqc* 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 function" on page 305.

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.

Copyright © 2007, iAnywhere Solutions, Inc.

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\h\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\h\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 "Database Error Messages" [*SQL Anywhere 10 - 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 (*%1*) 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.

 Copyright © 2007, iAnywhere Solutions, Inc.

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 String Parameters Reference" [*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 function" on page 305.

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 thus has no performance impact. 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*].

 Copyright © 2007, iAnywhere Solutions, Inc.

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;
```

Copyright © 2007, iAnywhere Solutions, Inc.

```
    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 n-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 */
```


Copyright © 2007, iAnywhere Solutions, Inc.

- NULL terminated system-dependent character string.

  A TCHAR is equivalent to a WCHAR for systems that use UNICODE (for example, Windows CE) 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 {
     unsigned short int 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 {
     unsigned short int 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 the programmer 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*].

Copyright © 2007, iAnywhere Solutions, Inc.

- **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.

```
typedef struct BINARY {
  unsigned short int len;
  char array[1];
} BINARY;
```

- **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\h\sqlca.h* file. The VARCHAR, BINARY, and TYPE_DECIMAL types contain a one-character array and are thus not useful for declaring host variables, but 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.

 Copyright © 2007, iAnywhere Solutions, Inc.

- 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 {
```

 Copyright © 2007, iAnywhere Solutions, Inc.

```
   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;
```


Copyright © 2007, iAnywhere Solutions, Inc.

```
    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

An **indicator variable** is a C variable that holds 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 **short int**. 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 );
```

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	Fetched a non-NULL value.
–1	NULL value	Fetched 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 (lower case).

NULL is not the same as any value of the column's defined type. Thus, in order 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];
short int 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 41.

 Copyright © 2007, iAnywhere Solutions, Inc.

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 41.

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];
   short int    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];
   short int 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" );
      }
```

 Copyright © 2007, iAnywhere Solutions, Inc.

```
        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.

Copyright © 2007, iAnywhere Solutions, Inc.

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 SQLE_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 SQLE_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 SQLE_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 is difficult for most programmers to detect whether or not 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.

 Copyright © 2007, iAnywhere Solutions, Inc.

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.

On Palm OS, if you want to authenticate users whenever they return to an application from some other application, you must include the prompt for user and password information in your **PilotMain** routine.

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 function" on page 319
- "ULRevokeConnectFrom function" on page 326

Copyright © 2007, iAnywhere Solutions, Inc.

Encrypting data

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

See “Encrypting data” on page 61.

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 function” on page 308.

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 creation-time database properties for UltraLite" [*UltraLite - Database Management and Reference*].

 Copyright © 2007, iAnywhere Solutions, Inc.

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 63.

2. Assign the parameter values for your application.

   See "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*].

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

   See "Invoking synchronization" on page 64.

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

Synchronization parameters

The ul_sync_infor stucture is documented in the C/C++ component chapter; however, members of the structures are common to embedded SQL development as well. See "ul_synch_info_a struct" on page 179 or "ul_synch_info_w2 struct" on page 186 depending whether you are using ASCII or wide characters.

For a general description of synchronization parameters, see "Synchronization parameters for UltraLite" [*MobiLink - Client Administration*].

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\h\ulglobal.h*.

♦ **To initialize the synchronization parameters (embedded SQL)**

- Call the **ULInitSynchInfo** function. For example:

```
auto ul_synch_info synch_info;
ULInitSynchInfo( &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 machine `test.internal`, on port `2439`:

```
auto ul_synch_info synch_info;
ULInitSynchInfo( &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 = ULSocketStream();
synch_info.stream_parms =
   UL_TEXT("host=test.internal;port=2439");
ULSynchronize( &sqlca, &synch_info );
```

The following code for an application on the Palm Computing Platform is called when the user exits the application. It allows HotSync synchronization to take place, with a MobiLink user name of `50`, an empty password, a script version of `custdb`. The HotSync conduit communicates over TCP/IP with a MobiLink server running on the same machine as the conduit (`localhost`), on the default port (2439):

```
auto ul_synch_info synch_info;
ULInitSynchInfo( &synch_info );
synch_info.name = UL_TEXT("Betty Best");
synch_info.version = UL_TEXT("default");
synch_info.stream = ULConduitStream();
synch_info.stream_parms =
   UL_TEXT("stream=tcpip;host=localhost");
ULSetSynchInfo( &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. You need to add error handling code to your application in case the synchronization cannot be carried out.

Copyright © 2007, iAnywhere Solutions, Inc.

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

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

♦ **To invoke synchronization (HotSync)**

- Call ULInitSynchInfo to initialize the synchronization parameters, and call ULSetSynchInfo to manage synchronization before exiting the application.

  See "ULSetSynchInfo function" on page 332.

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" [*MobiLink - Client Administration*].

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_id         = %d\t(ss_stream_id)\n"
                "    stream_context    = %d\t(ss_stream_context)\n"
                "    stream_error_code = %ld\t(ss_error_code)\n"
                "    error_string      = \"%s\"\n"
                "    system_error_code = %ld\n",
        (int)info.stream_error.stream_id,
        (int)info.stream_error.stream_context,
        (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. More information about the specific error is supplied to your application in the members of the stream_error synchronization parameter.

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:

```
ULInitSynchInfo( &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_synch_status** structure as parameter.

The **ul_synch_status** structure has the following members:

```
ul_synch_status   state;
ul_u_short        tableCount;
ul_u_short        tableIndex;
   struct {
      ul_u_long   bytes;
      ul_u_short  inserts;
      ul_u_short  updates;
      ul_u_short  deletes;
```


Copyright © 2007, iAnywhere Solutions, Inc.

```
    }       sent;
    struct {
      ul_u_long   bytes;
      ul_u_short  inserts;
      ul_u_short  updates;
      ul_u_short  deletes;
    }       received;
p_ul_synch_info   info;
ul_bool           stop;
```

- **state** One of the following states:
  - **UL_SYNCH_STATE_STARTING** No synchronization actions have yet been taken.
  - **UL_SYNCH_STATE_CONNECTING** The synchronization stream has been built, but not yet opened.
  - **UL_SYNCH_STATE_SENDING_HEADER** The synchronization stream has been opened, and the header is about to be sent.
  - **UL_SYNCH_STATE_SENDING_TABLE** A table is being sent.
  - **UL_SYNCH_STATE_SENDING_DATA** Schema information or data is being sent.
  - **UL_SYNCH_STATE_FINISHING_UPLOAD** The upload stage is completed and a commit is being carried out.
  - **UL_SYNCH_STATE_RECEIVING_UPLOAD_ACK** An acknowledgement that the upload is complete is being received.
  - **UL_SYNCH_STATE_RECEIVING_TABLE** A table is being received.
  - **UL_SYNCH_STATE_SENDING_DATA** Schema information or data is being received.
  - **UL_SYNCH_STATE_COMMITTING_DOWNLOAD** The download stage is completed and a commit is being carried out.
  - **UL_SYNCH_STATE_SENDING_DOWNLOAD_ACK** An acknowledgement that download is complete is being sent.
  - **UL_SYNCH_STATE_DISCONNECTING** The synchronization stream is about to be closed.
  - **UL_SYNCH_STATE_DONE** Synchronization has completed successfully.
  - **UL_SYNCH_STATE_ERROR** Synchronization has completed, but with an error.

  For more information about the synchronization process, see "The synchronization process" [*MobiLink - Getting Started*].

- **tableCount** Returns the number of tables being synchronized. For each table there is a sending and receiving phase, so this number may be more than the number of tables being synchronized.
- **tableIndex** The current table which is being uploaded or downloaded, starting at 0. This number may skip values when not all tables are being synchronized.
- **info** A pointer to the ul_synch_info structure.

- **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.
- **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.
- **getUserData** Returns the user data object.
- **getStatement** Returns the statement that called the synchronization. The statement is an internal UltraLite statement, and this method is unlikely to be of practical use, but is included for completion.
- **getErrorCode** When the synchronization state is set to ERROR, this method returns a diagnostic error code.
- **isOKToContinue** This is set to false when cancelSynchronization is called. Otherwise, it is true.

Example

The following code illustrates a very simple observer function:

```
extern void __stdcall ObserverFunc(
    p_ul_synch_status status )
{
    printf( "UL_SYNCH_STATE is %d: ",
            status->state );
    switch( status->state ) {
        case UL_SYNCH_STATE_STARTING:
        printf( "Starting\n");
        break;
    case UL_SYNCH_STATE_CONNECTING:
        printf( "Connecting\n"  );
        break;
    case UL_SYNCH_STATE_SENDING_HEADER:
        printf( "Sending Header\n" );
        break;
    case UL_SYNCH_STATE_SENDING_TABLE:
        printf( "Sending Table %d of %d\n",
                status->tableIndex + 1,
                status->tableCount );
        break;
...
```

This observer produces the following output when synchronizing two tables:

 Copyright © 2007, iAnywhere Solutions, Inc.

```
UL_SYNCH_STATE is 0: Starting
UL_SYNCH_STATE is 1: Connecting
UL_SYNCH_STATE is 2: Sending Header
UL_SYNCH_STATE is 3: Sending Table 1 of 2
UL_SYNCH_STATE is 3: Sending Table 2 of 2
UL_SYNCH_STATE is 4: Receiving Upload Ack
UL_SYNCH_STATE is 5: Receiving Table 1 of 2
UL_SYNCH_STATE is 5: Receiving Table 2 of 2
UL_SYNCH_STATE is 6: Sending Download Ack
UL_SYNCH_STATE is 7: Disconnecting
UL_SYNCH_STATE is 8: Done
```

CustDB example

An example of an observer function is included in the CustDB sample application. The implementation in CustDB provides a dialog 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 10 - 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.

Copyright © 2007, iAnywhere Solutions, Inc.

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 (object file, in most cases) 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.

The UltraLite plug-in for Metrowerks CodeWarrior automatically provides Palm Computing Platform developers with the techniques described here. For more information about this plug-in, see "Developing UltraLite applications with Metrowerks CodeWarrior" on page 75.

For a tutorial describing a very simple project, see "Tutorial: Build an Application Using Embedded SQL" on page 121.

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 *.sqc* files to your development project.

   The **development project** is defined in your development tool.

2. Add a custom build rule for each *.sqc* 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):

   ```
   "%SQLANY10%\win32\sqlpp.exe" -q -u $(InputPath) $(InputName).cpp
   ```

   where *SQLANY10* 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 *.sqc* 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 *$(SQLANY10)\h* and any other include folders you require to your include path, as a comma-separated list.

 Copyright © 2007, iAnywhere Solutions, Inc.

CHAPTER 5

Developing UltraLite Applications for the Palm OS

Contents

Introduction to Palm OS development 74
Developing UltraLite applications with Metrowerks CodeWarrior 75
Maintaining state in UltraLite Palm applications 79
Registering the Palm creator ID 81
Adding HotSync synchronization to Palm applications 82
Adding TCP/IP, HTTP, or HTTPS synchronization to Palm applications 84
Deploying Palm applications 85

Introduction to Palm OS development

Development environments

You can use one of the following development environments to build UltraLite Palm applications:

- Metrowerks CodeWarrior, using embedded SQL or UltraLite C++.

  See "Developing UltraLite applications with Metrowerks CodeWarrior" on page 75.

  CodeWarrior includes a version of the Palm SDK. Depending on the particular devices you are targeting, you may want to upgrade your Palm SDK to a more recent version than that included with the development tool.

  For a list of supported platforms, see "Supported platforms" [*SQL Anywhere 10 - Introduction*].

- AppForge MobileVB, using UltraLite MobileVB.

  See UltraLite - AppForge Programming [*UltraLite - AppForge Programming*].

Target platforms

For a list of supported target operating systems, see "Supported platforms" [*SQL Anywhere 10 - Introduction*].

Copyright © 2007, iAnywhere Solutions, Inc.

Developing UltraLite applications with Metrowerks CodeWarrior

A Metrowerks CodeWarrior plug-in simplifies the creation of UltraLite applications. The plug-in is supplied in the *install-dir\ultralite\palm\68k\cwplugin* directory. Please read the file *readme.txt* in that directory.

Installing the UltraLite plug-in for CodeWarrior

The files for the UltraLite plug-in for CodeWarrior are placed on your disk during UltraLite installation, but you cannot use the plug-in without an additional installation step.

♦ **Installing the UltraLite plug-in for CodeWarrior**

1. From a command prompt, change to the *install-dir\ultralite\palm\68k\cwplugin* directory.
2. Run *install.bat* to copy the appropriate files into your CodeWarrior installation directory. The *install.bat* file requires two arguments:

   - CodeWarrior directory
   - CodeWarrior version.

   For example, the following command on one line, installs the plug-in for CodeWarrior 9 in the default CodeWarrior installation directory.

   ```
   install "c:\Program Files\Metrowerks\CodeWarrior for Palm OS Platform
   9.0" r9
   ```

   You need double quotes around the directory if the path has any embedded spaces.

Uninstalling the CodeWarrior plug-in

You can use *uninstall.bat* to uninstall the UltraLite plug-in from CodeWarrior. The *uninstall.bat* file requires the same arguments as described above for *install.bat*.

Creating UltraLite projects in CodeWarrior

♦ **To create an UltraLite project in CodeWarrior**

1. Start CodeWarrior.
2. Create a new project:

   a. From the CodeWarrior menu, choose File ► New. A tabbed dialog appears.

   b. On the Projects tab, choose Palm OS Application Stationery.

   c. Also on the Projects tab, choose a name and location for the project. Click OK.

3. Choose an UltraLite stationery.

   The UltraLite plug-in adds the following choices to the stationery list:

   - Palm OS UltraLite C++ App
   - Palm OS UltraLite ESQL App

   Choose the development model you want to use and click OK to create the project.

   The stationery is standard C stationery for embedded SQL, and standard C++ stationery for C++.

4. If you are using embedded SQL, configure the settings in the UltraLite Preprocessor Panel for your project. If you are using C++ these settings are ignored.

   a. On your project window (*.mcp*), click the Settings icon on the toolbar. The Project Settings window opens.

   b. In the tree on the left pane, choose Target ► UltraLite Preprocessor. Enter the settings for your project.

Preprocessing

When you build an embedded SQL project, the UltraLite plug-in calls *sqlpp* to preprocess *.sqc* files into *.c/.cpp* files and also converts ESQL statements to UltraLite function calls.

When building UltraLite Embedded SQL or C++ applications in the CodeWarrior environment, the plug-in does not add the access paths to *install-dir\h* and *install-dir\ultralite\palm\68k\lib\cw* and to the UltraLite libraries *ulrt.lib* and *ulbase.lib*. The plug-in only adds the generated files from running the SQL preprocessor to the CodeWarrior project for UltraLite Embedded SQL applications.

Converting existing CodeWarrior project to an UltraLite application

If you install the UltraLite plug-in into CodeWarrior, you will be asked to convert older projects when they are first opened. In this conversion, CodeWarrior sets the default SQL preprocessor settings and saves them in the project file. This causes no disruption to projects that do not use the SQL preprocessor. If you want to further convert a project to invoke the SQL preprocessor automatically, you need to do the following:

1. Add a file mapping entry for *.sqc* files to the File Mappings panel of the Target settings.

   The file type is **TEXT** and the Compiler is **UltraLite Preprocessor**. All flags for these files must be unchecked.

2. For embedded SQL applications, remove all generated files from the Files view. These files are automatically generated and re-added when the *.sqc* files are built.

3. Remove any file mappings for *.ulg* files.

4. Verify the references to required library files.

Copyright © 2007, iAnywhere Solutions, Inc.

Using the UltraLite plug-in for CodeWarrior

For embedded SQL, the UltraLite plug-in for CodeWarrior integrates the UltraLite preprocessing steps into the CodeWarrior compilation model. It ensures that the SQL preprocessor runs when required.

Using prefix files

A **prefix file** is a header file that all source files in a Metrowerks CodeWarrior project must include. You should use *install-dir\h\ulpalmos.h* as your prefix file.

If you have your own prefix file, it must include *ulpalmos.h*. The *ulpalmos.h* file defines macros required by UltraLite Palm applications and also sets CodeWarrior compiler options required by UltraLite.

Encrypted synchronization
If you are using either the TLS or HTTPS protocols to implement encrypted synchronization, you must add *ulrsa.lib*, *ulecc.lib*, or *ulfips.lib* and *gselst.lib* to your CodeWarrior UltraLite projects.

Building the CustDB sample application in CodeWarrior

CustDB is a simple sales-status application.

For a diagram of the sample database schema, see "About the CustDB sample database" [*SQL Anywhere 10 - Introduction*].

Files for the application are located in the *samples-dir\UltraLite\CustDB* directory. Generic files are located in the *CustDB* directory. Files specific to CodeWarrior for the Palm OS are in the following directories:

- *samples-dir\UltraLite\CustDB\cwcommon*
- *samples-dir\UltraLite\CustDB\cw*

The instructions in this section describe how to build the CustDB application using CodeWarrior 9.

♦ To build the CustDB sample application using CodeWarrior

1. Start the CodeWarrior IDE.
2. Open the CustDB project file:
   - Choose File ► Open.
   - Open the project file *samples-dir\UltraLite\CustDB\cw\custdb.mcp*
3. To build the target application (*custdb.prc*), choose Project ► Make.

Building Expanded Mode applications

CodeWarrior supports a code generation mode called **expanded mode**, which improves memory use for global data. If you are using CodeWarrior version 9 you can use expanded mode with an A5-based jump table. To do so, you must use the expanded mode version of the UltraLite runtime library and the UltraLite base library. The expanded mode versions of these libraries are located as follows:

- *install-dir\ultralite\palm\68k\lib\cw9_a4a5jt\ulrt.lib*
- *install-dir\ultralite\palm\68k\lib\cw9_a4a5jt\ulbase.lib*

Expanded mode may be helpful for large applications that would otherwise exceed the 64 KB global data limit. A limitation of expanded mode is that encrypted synchronization can be used only via HotSync, as the synchronization security libraries for UltraLite do not use expanded mode.

 Copyright © 2007, iAnywhere Solutions, Inc.

Maintaining state in UltraLite Palm applications

You can save the state of tables and cursors when an application is closed by suspending the connection instead of closing it.

The current state is only stored for tables that are open when the connection object remains open.

Whenever your UltraLite application is closed or the user switches to another application, UltraLite saves the state of any open cursors and tables.

1. When the user returns to the application, call the appropriate open methods:

   - For embedded SQL, call the following functions:

     - db_init
     - EXEC SQL CONNECT

   - For C++, call the following functions:

     - ULSqlca.Initialize
     - ULInitDatabaseManager
     - OpenConnection

2. Confirm the connection was restored properly by checking that the SQLCODE is SQLE_CONNECTION_RESTORED.

3. For cursor objects, including instances of generated result set classes, you can do one of the following:

   - Ensure the object is closed when the user switches away from the application, and call Open when you next need the object. If you choose this option, the current position is not restored.

   - Do not close the object when the user switches away, and call Reopen when you next need to access the object. The current position is then maintained, but the application takes more memory in the Palm when the user is using other applications.

4. For table objects, including instances of generated table classes, you cannot save a position. You must close table objects before a user moves away from the application, and Open them when the user needs them again. Do not use Reopen on table objects.

Closing a connection rolls back any uncommitted transactions. By not closing connection objects, any outstanding transactions are saved (not committed), so that when the application restarts, those transactions appear and can be committed or rolled back. Uncommitted changes are not synchronized.

Restoring state in UltraLite Palm applications

When an application restarts on the Palm OS, UltraLite restores the state of any cursors or tables that were not explicitly closed when the application was most recently shut down.

Saving, retrieving, and clearing encryption keys on Palm OS

On Palm OS, applications are automatically shut down by the system whenever a user switches to a different application. Therefore, if you encrypt an UltraLite database on Palm OS, the user is prompted to re-enter the key *each* time they switch back to the application.

♦ To avoid re-entering the encryption key

1. Save the encryption key in dynamic memory as a Palm **feature**.

   Features are indexed by creator and a feature number. Applications can then pass in their creator ID or NULL, along with the feature number or NULL, to save and retrieve the encryption key.

2. Program the application to retrieve the key upon re-launch.

   Note
   Because the encryption key is cleared on any reset of the device, the retrieval of the key will fail at this time. The user is prompted to re-enter the key in this case.

   The following sample code (embedded SQL) illustrates how to save and retrieve the encryption key:

```
startupRoutine() {
   ul_char buffer[MAX_PWD];

   if( !ULRetrieveEncryptionKey(
          buffer, MAX_PWD, NULL, NULL  ) ){
      // prompt user for key
      userPrompt( buffer, MAX_PWD );


      if( !ULSaveEncryptionKey( buffer, NULL, NULL ) ) {
         // inform user save failed
      }
   }
}
```

3. Use a menu item to clear the encryption key and thereby secure the device.

   The following code sample shows the method to accommodate this goal:

```
case MenuItemClear
    ULClearEncryptionKey( NULL, NULL );
    break;
```

See also

- For UltraLite for AppForge: "Encryption and obfuscation" [*UltraLite - AppForge Programming*]
- For UltraLite.NET: "ULConnectionParms class" [*UltraLite - .NET Programming*]
- For UltraLite C++: "UltraLite_Connection_iface class" on page 210
- "UltraLite DBKEY connection parameter" [*UltraLite - Database Management and Reference*]
- For UltraLite for M-Business Anywhere: "Database encryption and obfuscation" [*UltraLite - M-Business Anywhere Programming*]
- "UltraLite obfuscate property" [*UltraLite - Database Management and Reference*]

 Copyright © 2007, iAnywhere Solutions, Inc.

Registering the Palm creator ID

UltraLite applications for the Palm OS, like all Palm OS applications, require a **creator ID**. You assign this creator ID to your application at creation time and, if you are using HotSync synchronization, you register the creator ID with HotSync manager for use by the MobiLink synchronization.

For information about assigning creator IDs to applications, see your development tool documentation. For more information about registering creator IDs with HotSync manager, see "HotSync on Palm OS" [*MobiLink - Client Administration*].

The creator ID is a string from one to four characters long. The first character should be an upper case letter, as Palm OS reserves the use of an initial lower-case letter for PALM OS system use.

Adding HotSync synchronization to Palm applications

HotSync synchronization takes place when an UltraLite application is closed. It is initiated by the HotSync.

If you use HotSync, then you synchronize by calling ULSetSynchInfo before closing the application. Do not use **ULSynchronize** or **ULConnection.Synchronize** for HotSync synchronization.

To enable HotSync synchronization from your application you must add code for the following steps:

1. Prepare a **ul_synch_info** structure.
2. Call the ULSetSynchInfo function, supplying the **ul_synch_info** structure as an argument.

   This function is called when the user switches away from the UltraLite application. You must ensure that all outstanding operations are committed before calling **db_fini**. The **ul_synch_info.stream** parameter is ignored, and so does not need to be set.

   For example:

   ```
   //C++ API
   ul_synch_info info;
   ULInitSynchInfo( &info );
   info.stream_parms =
     UL_TEXT( "stream=tcpip;host=localhost" );
   info.user_name = UL_TEXT( "50" );
   info.version = UL_TEXT( "custdb" );

   ULSetSynchInfo( &sqlca, &info );

   if( !db.Close( ) ) {
     return( false );
   }
   ```
3. Call db_fini.

See "Maintaining state in UltraLite Palm applications" on page 79 and "Synchronization parameters for UltraLite" [*MobiLink - Client Administration*].

An UltraLite HotSync conduit is required for HotSync synchronization of UltraLite applications. If there are uncommitted transactions when you close your Palm application, and if you synchronize, the conduit reports that synchronization fails because of uncommitted changes in the database.

Specifying stream parameters

The synchronization stream parameters in the ul_synch_info structure control communication with the MobiLink server. For HotSync synchronization, the UltraLite application does not communicate directly with a MobiLink server; it is the HotSync conduit instead.

You can supply synchronization stream parameters to govern the behavior of the MobiLink conduit in one of the following ways:

- Supply the required information in the stream_parms member of ul_synch_info passed to ULSetSynchInfo.

 Copyright © 2007, iAnywhere Solutions, Inc.

For a list of available values, see “Network protocol options for UltraLite synchronization streams” [*MobiLink - Client Administration*].

- Supply a null value for the stream_parms member. The MobiLink conduit then searches in the *ClientParms* registry entry on the machine where it is running for information on how to connect to the MobiLink server.

  The stream and stream parameters in the registry entry are specified in the same format as in the ul_synch_info structure stream_parms field.

  See “Deploying the UltraLite HotSync conduit to the end-user's desktop” [*MobiLink - Client Administration*].

See also

- “HotSync on Palm OS” [*MobiLink - Client Administration*]

Adding TCP/IP, HTTP, or HTTPS synchronization to Palm applications

This section describes how to add TCP/IP, HTTP, or HTTPS synchronization to your Palm application.

For a general description of how to add synchronization to UltraLite applications, see "Adding synchronization to your UltraLite application" [*MobiLink - Client Administration*].

Transport layer security on the Palm OS

You can use transport-layer security with Palm applications built with Metrowerks CodeWarrior.

For more information about transport-layer security, see "Encrypting MobiLink client/server communications" [*SQL Anywhere Server - Database Administration*].

Palm devices can synchronize using TCP/IP, HTTP, or HTTPS communication by setting the **stream** member of the **ul_synch_info** structure to the appropriate stream, and calling **ULSynchronize** or **ULConnection.Synchronize** to carry out the synchronization.

When using TCP/IP, HTTP, or HTTPS synchronization, db_init or db_fini save and restore the state of the application on exiting and activating the application, but do not participate in synchronization.

Before closing the application, set the synchronization information using ULSetSynchInfo, providing **ul_synch_info** structure as an argument.

When using TCP/IP, HTTP, or HTTPS synchronization from a Palm device, you must specify an explicit host name or IP address in the **stream_parms** member of the **ul_synch_info** structure. Specifying NULL defaults to `localhost`, which represents the device, not the host.

For more information about the **ul_synch_info** structure, see "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*].

Copyright © 2007, iAnywhere Solutions, Inc.

Deploying Palm applications

This section describes the following aspects of deploying Palm applications:

- Deploying the application.

  See "Deploying the application" on page 85

- Deploying the UltraLite synchronization conduit for HotSync.

  See "HotSync on Palm OS" [*MobiLink - Client Administration*].

- Deploying an initial copy of the UltraLite database.

  See "Deploying UltraLite databases" on page 85.

Install your UltraLite application on your Palm device as you would any other Palm OS application.

Deploying the application

♦ To install an application on a Palm device

1. Open the Install Tool, included with your Palm Desktop Organizer Software.
2. Choose Add and specify the location of your compiled application (*.prc* file).
3. Close the Install Tool.
4. Use the HotSync utility to copy the application to your Palm device.

Deploying the MobiLink synchronization conduit

For applications using HotSync synchronization, each end user must have the MobiLink synchronization conduit installed on their desktop.

For more information about installing the MobiLink synchronization conduit, see "HotSync on Palm OS" [*MobiLink - Client Administration*].

Deploying UltraLite databases

If you deploy your application without a database, the application must contain relatively complex code to create the database. The recommended approach is to create an initial database on a Windows desktop and copy the database file to the Palm device. Sybase Central (or the utility ulcreate) can be used to create an initial database. The user must then obtain an initial copy of data on the first synchronization. You can use the uldbutil utility to back up the UltraLite database to the PC. To deploy many UltraLite databases with an initial database including data, you can perform an initial synchronization and then back up the UltraLite database. The database can be deployed on other devices so they do not need to perform an initial synchronization.

See "UltraLite Data Management utility for Palm OS (ULDBUtil)" [*UltraLite - Database Management and Reference*].

If you are using HotSync synchronization, each of your end users must also install the synchronization conduit onto their desktop machine.

For more information about installing the synchronization conduit, see “Deploying the UltraLite HotSync conduit to the end-user's desktop” [*MobiLink - Client Administration*].

If you deploy a database using HotSync, HotSync sets a **backup bit** on the database. When this backup bit is set, the entire database is backed up to the desktop machine on each synchronization. This behavior is generally not appropriate for UltraLite databases. When an UltraLite application is launched, the Palm data store is checked to see if its backup bit is set to true. If it is set, it is cleared. If it is not set, there is no change.

If you want the backup bit to remain set to true, you can set the store parameter **palm_allow_backup** in the database connection string.

 Copyright © 2007, iAnywhere Solutions, Inc.

CHAPTER 6

Developing UltraLite Applications for Symbian OS

Contents

Introduction to Symbian OS development 88
Developing applications using CodeWarrior for Symbian 90

Introduction to Symbian OS development

Development environments

You can use one of the following development environments to build UltraLite applications on Symbian OS:

- CodeWarrior for Symbian. Current version is 3.1 but earlier versions are also supported.

  CodeWarrior includes a version of the Symbian SDK (this may be located on a separate CD in the installation package). Depending on the particular devices you are targeting, you may want to upgrade your SDK to a more recent version than that included in the development tool.

  For a list of supported platforms, see "Supported platforms" [*SQL Anywhere 10 - Introduction*].

- The Carbide C++ Express Development environment from Nokia. This is a development tool and Integrated Development Environment for 32-bit Windows desktops. See http://www.forum.nokia.com/carbide/

Target platforms

For a list of supported target operating systems, see "Supported platforms" [*SQL Anywhere 10 - Introduction*].

Choosing how to link the runtime library

Two sets of library files are provided for UltraLite support for Symbian OS. These libraries are located in *install-dir\ultralite\Symbian\winscw* and *install-dir\ultralite\Symbian\thumb*.

- *ulrt10.dll* and *\lib\ulimp.lib* are for dynamically linking with the application
- *\lib\ulrt.lib* and *\lib\ulbase.lib* are for statically linking with the application.

Both the static and dynamic UltraLite libraries use the Symbian OS "Thread Local Storage" technique to store static data in a DLL. This means that both libraries can only be used in a DLL environment. If the database application is a GUI application under the Symbian OS, the APP target type is just a special type of DLL. This means developers can link their application with either the static UltraLite runtime libraries (*ulrt.lib* and *ulbase.lib*), or the dynamic UltraLite runtime library (*ulimp.lib*).

If the developer is writing a console application, which has target type of EXE under the Symbian OS, the application can only be linked with the dynamic version of the UltraLite runtime library (*ulimp.lib*). In this case, remember to copy *ulrt10.dll* file to the emulator directory and the actual device.

Memory utilization in Symbian OS applications

The UltraLite runtime can use a significant amount of memory for stack and heap storage. The UltraLite database application on Symbian OS should specify a larger stack size and heap size than the default setting. Both specifications can be changed in the Symbian project (mmp) file. The syntax of the specifications for

Copyright © 2007, iAnywhere Solutions, Inc.

the Symbian project file is described in the Symbian SDK documentation: Tools And Utilities » Build tools reference » mmp file syntax.

Use the epocstacksize statement to specify a stack size for your executable. The default size is 8 KB.

Use the epocheapsize statement to specify the minimum and maximum sizes of the initial heap for a process. The default sizes are 4 KB minimum and 1 MB maximum.

Copyright © 2007, iAnywhere Solutions, Inc.

Developing applications using CodeWarrior for Symbian

This section outlines the steps to successfully compile and deploy a sample UltraLite application for a Nokia Series 60 device using CodeWarrior for Symbian version 3.1. The source code and project files for this sample program are located in *samples-dir\UltraLite\SymbianTutorial*

♦ To compile the tutorial application

1. Prepare the tutorial files.

   Copy the tutorial directory (including subfolders) from the SQL Anywhere installation to your C: drive if you are using CodeWarrior for Symbian version 3.1. CodeWarrior 3.1 only allows importing a project file (.mpp extension) from the C: drive. Although it is not necessary to do this if you are using CodeWarrior v3.0 or an earlier version, the rest of this procedure assumes this step has been performed.

   ```
   xcopy /s %SQLANYSAMP10%\UltraLite\SymbianTutorial\ C:\Symbian\ULTutorial
   ```

   Copy the UTF8-encoded UltraLite sample database file to the "group" directory of the tutorial. This enables the CodeWarrior packaging script to add the database file to the installer sis file for device deployment. Also copy the sample database file to the virtual file directory of the Nokia Series 60 SDK emulator. This is for running the UltraLite tutorial in the emulator. The environment variable "%EPOC_ROOT%" is normally already set to the location of the EPOC32 directory in the SDK.

   ```
   copy %SQLANYSAMP10%\UltraLite\CustDB\custdb.utf8.udb C:\Symbian
   \ULTutorial\group\custdb.udb
   copy %SQLANYSAMP10%\UltraLite\CustDB\custdb.utf8.udb %EPOC_ROOT%\winscw\c
   ```

   Copy the UltraLite runtime dll and library file to the Nokia Series 60 SDK build target directory.

   ```
   copy %SQLANY10%\ultralite\Symbian\winscw\ulrt10.dll %EPOC_ROOT%\release
   \winscw\udeb
   copy %SQLANY10%\ultralite\Symbian\winscw\lib\ulimp.lib %EPOC_ROOT%
   \release\winscw\udeb

   copy %SQLANY10%\ultralite\Symbian\thumb\ulrt10.dll %EPOC_ROOT%\release
   \thumb\urel
   copy %SQLANY10%\ultralite\Symbian\thumb\lib\ulimp.lib %EPOC_ROOT%\release
   \thumb\urel
   ```

2. Import the mmp project file into CodeWarrior for Symbian IDE.

   a. Launch the CodeWarrier IDE.

   b. Choose File ▶ Import Project from mmp files.

   c. Select the vendor: Vendor Nokia/SDK Series 60 2.0+.

   d. Choose the platform, either WINSCW or THUMB.

3. Add the UltraLite include path to the project settings.

 Copyright © 2007, iAnywhere Solutions, Inc.

a. Choose Edit ► WINSCW UDEB Settings ► Target Settings Panels ► Target ► Access Paths ► User Paths.

b. Click Add to add %SQLANY10%\h to User Paths (use the absolute path without environment variable).

c. Change the build target to THUMB UREL and do the same.

d. Click Add to add %SQLANY10%\h to User Paths (use the absolute path without environment variable).

4. Build the application.

   a. Change target to WINSCW, and then choose Project ► Make.

      You can now run the application in the Emulator environment by choosing Project ► Run.

   b. Change target to THUMB UREL, and then choose Project ► Make.

      This compiles the project for the device environment.

5. Create the package for device.

   a. Edit the *group\ULTutorial.pkg* file: replace references to the Epoc32 folder with your %EPOC_ROOT% setting (environment variables are not recognized in the *.pkg* file, so you must specify the absolute path) and replace the path to *custdb.udb* with the location of the sample database on your computer.

   b. Under the THUMB UREL target, right-click the source to add the file *group\ULTutorial.pkg*.

   c. Build the project for target THUMB UREL again and make certain that the file *group\Application.sis* has been created. This is the file that is deployed to the device.

6. Deploy to the device.

   To deploy the tutorial application to a Nokia Series 60 phone, you must connect to the device (using a USB cradle or a wireless connection—either Bluetooth or IRDA) and use the Nokia PC Suite software to transfer the Application.sis file to the phone.

Notes about the tutorial program

The tutorial program directory contains a number of files. Most of the files are concerned with the user interface (GUI) on the phone. The main UltraLite application code is in the file named *ULTutorialDB.cpp* located in the *\src* subdirectory.

There are some considerations regarding the location of the database file on the phone and how to reference that in the UltraLite connection code. The C++ code to connect to the database uses the setting DBF=C:\\custdb.udb to specify the file name (and path) for the database file. This indicates the database file is deployed in the main phone memory (not on a storage card).

If the application fails to start on the phone there is a strong possibility that the location of the database file is not correct.

CHAPTER 7

Developing UltraLite Applications for Windows CE

Contents

Introduction to Windows CE development 94
Building the CustDB sample application 96
Storing persistent data 98
Deploying Windows CE applications 99
Synchronization on Windows CE 102

Introduction to Windows CE development

For a list of supported host platforms and development tools for Windows CE development, and for a list of supported target Windows CE platforms, see "Supported platforms" [*SQL Anywhere 10 - Introduction*].

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

Preparing for Windows CE development

Microsoft eMbedded Visual C++ can be used to develop applications for the Windows CE environment. This development environment is available from Microsoft as part of eMbedded Visual Tools.

You can download eMbedded Visual C++ from the Microsoft Developer Network at http://msdn.microsoft.com/.

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

A first application

A sample eMbedded Visual C++ project is provided in the *samples-dir\UltraLite\CEStarter* directory. The workspace file is *samples-dir\UltraLite\CEStarter\ul_wceapplication.vcw*.

When preparing to use eMbedded Visual C++ for UltraLite applications, you should make the following changes to the project settings. The CEStarter application has these changes made.

- Compiler settings:
  - Add *$(SQLANY10)\h* to the include path.
  - Define appropriate compiler directives. For example, the UNDER_CE macro should be defined for eMbedded Visual C++ projects.
- Linker settings:
  - Add "*$(SQLANY10)\ultralite\ce\processor\lib\ulrt.lib*"

    where *processor* is the target processor for your application.
  - Add *winsock.lib*.
- The *.sqc* file (embedded SQL only):
  - Add *ul_database.sqc* and *ul_database.cpp* to the project
  - Add the following custom build step for the *.sqc* file:

    ```
    "$(SQLANY10)\win32\sqlpp" -q $(InputPath) ul_database.cpp
    ```
  - Set the output file to *ul_database.cpp*.
  - Disable the use of precompiled headers for *ul_database.cpp*.

 Copyright © 2007, iAnywhere Solutions, Inc.

Choosing how to link the runtime library

Windows CE 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.

Copyright © 2007, iAnywhere Solutions, Inc.

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. Generic files are located in the *CustDB* subdirectory. Files specific to Windows CE are located in the *EVC* subdirectory of *CustDB*.

The CustDB application is provided as an eMbedded Visual C++ 3.0 project.

For a diagram of the sample database schema, see "About the CustDB sample database" [*SQL Anywhere 10 - Introduction*].

♦ To build the CustDB sample application

1. Start eMbedded Visual C++.
2. Open the project file that corresponds to your version of eMbedded Visual C++:
   - *samples-dir\UltraLite\CustDB\EVC\EVCCustDB.vcp* for eVC 3.0.
   - *samples-dir\UltraLite\CustDB\EVC40\EVCCustDB.vcp* for eVC 4.0.
3. Choose Build ► Set Active Platform to set the target platform.
   - Set a platform of your choice.
4. Choose Build ► Set Active Configuration to select the configuration.
   - Set an active configuration of your choice.
5. If you are building CustDB for the Pocket PC x86em emulator platform only:
   - Choose Project ► Settings.

     The Project Settings dialog appears.
   - On the Link tab, in the Object/library Modules field, change the UltraLite runtime library entry to the *emulator30* directory rather than the *emulator* directory.
6. Build the application:
   - Press F7 or choose Build ► Build EVCCustDB.exe to build CustDB.

     When eMbedded Visual C++ has finished building the application, it automatically attempts to upload it to the remote device.
7. Start the Mobilink server:
   - To start the MobiLink server, choose Programs ► SQL Anywhere 10 ► MobiLink ► Mobilink Server Sample.
8. Run the CustDB application:

Copyright © 2007, iAnywhere Solutions, Inc.

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.

Press Ctrl+F5 or choose Build ▶ Execute CustDB.exe.

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 70.

Storing persistent data

The UltraLite database is stored in the Windows CE file system. The default file is *\UltraLiteDB \ul_<project>.udb*, with *project* being truncated to eight characters. 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"
```

 Copyright © 2007, iAnywhere Solutions, Inc.

Deploying Windows CE applications

When compiling UltraLite applications for Windows CE, 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 CE emulator, place the DLL in the appropriate subdirectory of your Windows CE tools directory. The following directory is the default setting for the Pocket PC emulator:

```
C:\Program Files\Windows CE Tools\wce300\MS Pocket PC\
emulation\palm300\windows
```

Deploying applications that use ActiveSync

Applications that use ActiveSync synchronization must be registered with ActiveSync and copied to the device. The MobiLink provider for ActiveSync must also be installed.

See "Registering applications with the ActiveSync Manager" [*MobiLink - Client Administration*].

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 and eMbedded Visual C++.

♦ To assign a window class name for MFC applications using eMbedded Visual C++

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 **CMyAppDlg**.

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.

   If eMbedded Visual C++ is open, save your changes and close your project and workspace.

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.

Copyright © 2007, iAnywhere Solutions, Inc.

- Save the *.rc* file.

7. Reopen eMbedded Visual C++ and load your project.

8. Add code to catch the synchronization message.

   See "Adding ActiveSync synchronization (MFC)" on page 103.

Synchronization on Windows CE

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

- **ActiveSync** See "Adding ActiveSync synchronization to your application" on page 102.
- **TCP/IP** See "TCP/IP, HTTP, or HTTPS synchronization from Windows CE" on page 104.
- **HTTP** See "TCP/IP, HTTP, or HTTPS synchronization from Windows CE" on page 104.

The *user_name* and *stream_parms* parameters must be surrounded by the **UL_TEXT()** macro for Windows CE when initializing, since the compilation environment is Unicode wide characters.

For more information about synchronization parameters, see "Synchronization parameters for UltraLite" [*MobiLink - Client Administration*].

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 CE 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 the Synchronization command 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 99.

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 ActiveSync in the UltraLite table in SQL Anywhere 10.0.1 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.

 Copyright © 2007, iAnywhere Solutions, Inc.

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 function that actually calls ULSynchronize.

See “ULIsSynchronizeMessage function” on page 323.

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 99.

♦ 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,
   ```

Copyright © 2007, iAnywhere Solutions, Inc.

```
   LPARAM lParam
);
```

The return value of this function should be 0.

For more information about handling the synchronization message, see "ULIsSynchronizeMessage function" on page 323.

♦ To add ActiveSync synchronization in the Application class

1. Open up the Class Wizard for the application class.
2. In the Messages list, highlight PreTranslateMessage and then click the Add Function button.
3. Click the Edit Code button. 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 function" on page 323.

TCP/IP, HTTP, or HTTPS synchronization from Windows CE

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.

 Copyright © 2007, iAnywhere Solutions, Inc.

Part III. Tutorials

This part provides tutorials that walk you through the development of a simple UltraLite application.

CHAPTER 8

Tutorial: Build an Application Using the C++ API

Contents

Introduction to UltraLite C++ development 108
Lesson 1: Create database and connect to database 109
Lesson 2: Insert data into the database 113
Lesson 3: Select and list rows from the table 114
Lesson 4: Add synchronization to your application 116
Code listing for tutorial 118

Introduction to UltraLite C++ development

This tutorial guides you through the process of building an UltraLite C++ application. The application is built for Windows operating systems, and runs at a command prompt.

This tutorial is based on development using Microsoft Visual C++ .NET, 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 "Creating an UltraLite database from Sybase Central" [*UltraLite - Database Management and Reference*].

The goal for the tutorial is to gain competence with the process of developing an UltraLite C++ application.

Copyright © 2007, iAnywhere Solutions, Inc.

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. Create a directory to hold the files you 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*.

2. Using UltraLite in Sybase Central, create a database named *ULCustomer.udb* in your new directory with the following characteristics.

   For more information about using UltraLite in Sybase Central, see "Creating an UltraLite database from Sybase Central" [*UltraLite - Database Management and Reference*].

3. 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	

♦ Connecting to the UltraLite database

1. In Microsoft Visual C++ .NET, 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 and use the UltraLite namespace.

   Copy the code below into *customer.cpp*:

```
#include <stdafx.h>
#include "uliface.h"
using namespace UltraLite;
#define MAX_NAME_LEN 100
ULSqlca Tutca;
```

   This code fragment defines an UltraLite SQL communications area (ULSqlca) named Tutca.

   For more information about using the UltraLite namespace to make class declarations simpler, see "Using the UltraLite namespace" on page 16.

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 =
  UL_TEXT( "UID=DBA;PWD=sql" )
  UL_TEXT( ";DBF=C:\\tutorial\\cpp\\ULCustomer.udb" );
```

For more information about connection parameters, see "UltraLite_Connection_iface class" on page 210.

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 function 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 function or can choose to use an error callback function.

   This is a sample callback function.

```
ul_error_action UL_GENNED_FN_MOD MyErrorCallBack(
    SQLCA *     Tutca,
    ul_void *    user_data,
    ul_char *    message_param )
{
    ul_error_action rc;

       (void) user_data;

    switch( Tutca->sqlcode ){
           // The following error is used for flow control - don't report
it here
              case SQLE_NOTFOUND:
                     rc = UL_ERROR_ACTION_CONTINUE;
              break;

              default:
                     if (Tutca->sqlcode >= 0) { // warning or success
                            rc = UL_ERROR_ACTION_DEFAULT;
                     } else { // negative is real error
                            _tprintf( _TEXT( "Error %ld: %s\n" ), Tutca-
>sqlcode, message_param );
                            rc = UL_ERROR_ACTION_CANCEL;
                     }
              break;
        }
    return rc;
    }
```

 Copyright © 2007, iAnywhere Solutions, Inc.

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 31.

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.

```
Connection * open_conn( DatabaseManager * dm ) {
  Connection * conn = dm->OpenConnection( Tutca, ConnectionParms );

  if( conn == UL_NULL ) {
      _tprintf( _TEXT("Unable to open existing database.\n") );
   }
  return conn;
}
```

8. Implement the main method to carry out the following tasks:

   - Instantiates a DatabaseManager object. All UltraLite objects are created from the DatabaseManager object.
   - Registers the error handling function.
   - Opens a connection to the database.
   - Closes the connection and shuts down the database manager.

```
int main() {
  ul_char buffer[ MAX_NAME_LEN ];

  Connection * conn;

  Tutca.Initialize();

  ULRegisterErrorCallback(
    Tutca.GetCA(), MyErrorCallBack,
    UL_NULL, buffer, sizeof (buffer));

  DatabaseManager * dm = ULInitDatabaseManager( Tutca );

  conn = open_conn( dm );

  if( conn == UL_NULL ){
      dm->Shutdown( Tutca );
      Tutca.Finalize();
      return 1;
  }
  // main processing code to be inserted here
  dm->Shutdown( Tutca );
  Tutca.Finalize();
  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"$(SQLANY10)\h"
```

- In the makefile, add directories to your libraries path.

```
LibraryFolders=/LIBPATH:"$(SQLANY10)\ultralite\win32\386\lib"
```

- 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 the makefile.

```
nmake
```

  This creates an executable named *customer.exe*.

10. Run the application.

    At the command prompt, enter **customer**.

Copyright © 2007, iAnywhere Solutions, Inc.

Lesson 2: Insert data into the database

The following procedure demonstrates how to add data to a database.

♦ Adding rows to your database

1. Add the procedure below to *customer.cpp*, immediately before the main method:

```
bool do_insert( Connection * conn ) {
  Table * table = conn->OpenTable( UL_TEXT("ULCustomer") );
  if( table == UL_NULL ) {
      _tprintf( _TEXT("Table not found: ULCustomer\n") );
      return  false;
  }
  if( table->GetRowCount() == 0 ) {
      _tprintf( _TEXT("Inserting one row.\n") );
      table->InsertBegin();
      table->Set( UL_TEXT("cust_name"), UL_TEXT("New Customer") );
      table->Insert();
      conn->Commit();
  } else {
      _tprintf( _TEXT("The table has %lu rows\n"),
      table->GetRowCount() );
  }
  table->Release();
  return true;
}
```

   This procedure carries out 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.

     The commit method is only required when autocommit is turned off. By default, autocommit is enabled, but it may be disabled for better performance, or for multi-operation transactions.
   - 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 after the call to open_conn.

```
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.

♦ Listing 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.

```
bool do_select( Connection * conn ) {
  Table * table = conn->OpenTable( _TEXT("ULCustomer") );
  if( table == UL_NULL ) {
    return false;
  }
  TableSchema * schema = table->GetSchema();
  if( schema == UL_NULL ) {
    table->Release();
    return false;
  }
  ul_column_num id_cid =
     schema->GetColumnID( UL_TEXT("cust_id") );
  ul_column_num cname_cid =
     schema->GetColumnID( UL_TEXT("cust_name") );

  schema->Release();

  _tprintf( "\n\nTable 'ULCustomer' row contents:\n");

  while( table->Next() ) {
    ul_char cname[ MAX_NAME_LEN ];

    table->Get( cname_cid ).GetString(
                cname, MAX_NAME_LEN );

    _tprintf( "id=%d, name=%s \n",
              (int)table->Get(id_cid), cname);
  }
  table->Release();
  return true;
}
```

2. Add the following line to the main method, immediately after the call to the insert method:

 Copyright © 2007, iAnywhere Solutions, Inc.

```
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 10.0 Sample database. The UltraLite 10.0 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.

♦ Add synchronization to your application

1. Add the method below to *customer.cpp*. This method carries out the following tasks:

   - Sets the synchronization stream to TCP/IP by invoking ULEnableTcpipSynchronization. Synchronization can also be carried out over HTTP, HotSync, or HTTPS. See "UltraLite Clients" [*MobiLink - Client Administration*].

   - 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.

```
bool do_sync( Connection * conn ) {
  ul_synch_info info;
  ul_stream_error * se = &info.stream_error;

  ULEnableTcpipSynchronization( Tutca.GetCA() );
  conn->InitSynchInfo( &info );
  info.stream = ULSocketStream();
  info.version = UL_TEXT( "custdb 10.0" );
  info.user_name = UL_TEXT( "50" );
  info.download_only = true;
  if( !conn->Synchronize( &info ) ) {
      _tprintf( _TEXT("Synchronization error \n" ));
      _tprintf( _TEXT("  stream_error_code is '%lu'\n"), se-
>stream_error_code );
      _tprintf( _TEXT("  system_error_code is '%ld'\n"), se-
>system_error_code );
      _tprintf( _TEXT("  error_string is '") );
      _tprintf( _TEXT("%s"), se->error_string );
      _tprintf( _TEXT("'\n") );
      return false;
  }
  return true;
}
```

 Copyright © 2007, iAnywhere Solutions, Inc.

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:

```
mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -v+ -zu+
```

   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 window displays status messages indicating the synchronization progress. If synchronization is successful, the final message displays `Synchronization complete`.

Code listing for tutorial

Following is the complete code for the tutorial program described in the preceding sections.

```
#include <stdafx.h>
#include "uliface.h"
using namespace UltraLite;
#define MAX_NAME_LEN 100
ULSqlca Tutca;

static ul_char const * ConnectionParms =
  UL_TEXT( "UID=DBA;PWD=sql;" )
  UL_TEXT( "DBF=C:\\temp\\ULCustomer.udb" );

ul_error_action UL_GENNED_FN_MOD MyErrorCallBack(
    SQLCA *      Tutca,
    ul_void *     user_data,
    ul_char *     message_param )
{
    ul_error_action rc;

        (void) user_data;

    switch( Tutca->sqlcode ){
           // The following error is used for flow control - don't report it
here
              case SQLE_NOTFOUND:
                     rc = UL_ERROR_ACTION_CONTINUE;
              break;

              default:
                     if (Tutca->sqlcode >= 0) { // warning or success
                            rc = UL_ERROR_ACTION_DEFAULT;
                     } else { // negative is real error
                        _tprintf( _TEXT( "Error %ld: %s\n" ), Tutca->sqlcode,
message_param );
                            rc = UL_ERROR_ACTION_CANCEL;
                     }
              break;
        }
    return rc;
    }
Connection * open_conn( DatabaseManager * dm ) {
  Connection * conn = dm->OpenConnection( Tutca, ConnectionParms );
  if( conn == UL_NULL ) {
      _tprintf( _TEXT("Unable to open existing database.\n") );
   }
  return conn;
}

// Open table, insert 1 row if table is currently empty

bool do_insert( Connection * conn ) {
  Table * table = conn->OpenTable( UL_TEXT("ULCustomer") );
  if( table == UL_NULL ) {
      _tprintf( _TEXT("Table not found: ULCustomer\n") );
      return  false;
   }
  if( table->GetRowCount() == 0 ) {
      _tprintf( _TEXT("Inserting one row.\n") );
      table->InsertBegin();
      table->Set( UL_TEXT("cust_name"), UL_TEXT("New Customer") );
      table->Insert();
```


Copyright © 2007, iAnywhere Solutions, Inc.

```
        conn->Commit();

    } else {
        _tprintf( _TEXT("The table has %lu rows\n"),
        table->GetRowCount() );
    }
    table->Release();
    return true;
}// Open table, display data from all rows

bool do_select( Connection * conn ) {
    Table * table = conn->OpenTable( _TEXT("ULCustomer") );
    if( table == UL_NULL ) {
    return false;
    }
    TableSchema * schema = table->GetSchema();
    if( schema == UL_NULL ) {
      table->Release();
      return false;
    }
    ul_column_num id_cid = schema->GetColumnID( UL_TEXT("cust_id") );
    ul_column_num cname_cid = schema->GetColumnID( UL_TEXT("cust_name") );

    schema->Release();

    _tprintf( "\n\nTable 'ULCustomer' row contents:\n");

    while( table->Next() ) {
      ul_char cname[ MAX_NAME_LEN ];

      table->Get( cname_cid ).GetString( cname, MAX_NAME_LEN );

      _tprintf( "id=%d, name=%s \n", (int)table->Get( id_cid ), cname );
    }
    table->Release();
    return true;
}

// sync database with MobiLink connection to reference database

bool do_sync( Connection * conn ) {
    ul_synch_info info;
    ul_stream_error * se = &info.stream_error;

    ULEnableTcpipSynchronization( Tutca.GetCA() );
    conn->InitSynchInfo( &info );
    info.stream = ULSocketStream();
    info.version = UL_TEXT( "custdb 10.0" );
    info.user_name = UL_TEXT( "50" );
    info.download_only = true;
    if( !conn->Synchronize( &info ) ) {
        _tprintf( _TEXT("Synchronization error \n" ));
        _tprintf( _TEXT("  stream_error_code is '%lu'\n"), se-
>stream_error_code );
        _tprintf( _TEXT("  system_error_code is '%ld'\n"), se-
>system_error_code );
        _tprintf( _TEXT("  error_string is '") );
        _tprintf( _TEXT("%s"), se->error_string );
        _tprintf( _TEXT("'\n") );
        return false;
    }
    return true;
}
```

Copyright © 2007, iAnywhere Solutions, Inc.

```
int main() {
  ul_char buffer[ MAX_NAME_LEN ];

  Connection * conn;

  Tutca.Initialize();

  ULRegisterErrorCallback(
    Tutca.GetCA(), MyErrorCallBack,
    UL_NULL, buffer, sizeof (buffer));

  DatabaseManager * dm = ULInitDatabaseManager( Tutca );

  if( dm == UL_NULL ){
    //  You  may  have  mismatched  UNICODE  vs.  ANSI  runtimes.
    Tutca.Finalize();
    return 1;
  }

  conn = open_conn( dm );

  if( conn == UL_NULL ){
      dm->Shutdown( Tutca );
      Tutca.Finalize();
      return 1;
  }

  do_insert (conn);
  do_sync (conn);
  do_select (conn);


  dm->Shutdown( Tutca );
  Tutca.Finalize();
  return 0;
}
```


Copyright © 2007, iAnywhere Solutions, Inc.

CHAPTER 9

Tutorial: Build an Application Using Embedded SQL

Contents

Introduction to embedded SQL development tutorial 122
Lesson 1: Create the UltraLite database 123
Lesson 2: Configure eMbedded Visual C++ 124
Lesson 3: Write an embedded SQL source file 126
Lesson 4: Build the embedded SQL UltraLite tutorial application 132
Lesson 5: Add synchronization to your application 133

Introduction to embedded SQL development tutorial

In this tutorial, you develop an embedded SQL UltraLite application using Microsoft eMbedded Visual C++. This UltraLite application can be executed on a Windows CE emulator or device.

For an overview of the development process, see "Developing embedded SQL applications" on page 4.

For more information about developing embedded SQL UltraLite Applications, see "Developing Applications Using Embedded SQL" on page 37.

This tutorial assumes that you have UltraLite and Microsoft eMbedded Visual Tools installed on your computer. If you use a different C/C++ development tool, you will have to translate the eMbedded Visual C++ instructions into their equivalent for your development tool.

♦ To prepare for the tutorial

- Create a directory on your desktop computer to hold the files you will create.

  The remainder of this tutorial assumes that the directory *C:\tutorial* is the location of the project files.

 Copyright © 2007, iAnywhere Solutions, Inc.

Lesson 1: Create the UltraLite database

You must create an UltraLite database to be used by the tutorial application. One way to do this is to use Sybase Central to create the UltraLite database on the desktop computer. The file containing the UltraLite database is later transferred to the Windows CE device. For this tutorial, create the database on the desktop computer with the file name *C:\tutorial\esqldb.udb*.

Sybase Central creates the database with the user ID DBA and password sql (UltraLite database passwords are case sensitive; user names are not).

The database has one table named ULProduct that contains the following fields with characteristics as shown:

Pkey	Name	Data type	Nulls	Unique
yes	prod_id	integer	no	yes
	price	integer	no	no
	prod_name	varchar(30)	no	no

Copy the database file to the target device using Windows Explorer or the Remote File Viewer tool provided in eMbedded Visual C++. The database connection information in the tutorial program specifies **dbf=\esqldb.udb** inidcating the database file on the device is located in the root folder of the CE device. The database file may be located anywhere in the file system for the device—provided the connection information is adjusted accordingly. Since the Windows CE file explorer does not show file extensions, the database file is named *esqldb.udb* to clearly distinguish the database file from the executable file *esql.exe*

Copyright © 2007, iAnywhere Solutions, Inc.

Lesson 2: Configure eMbedded Visual C++

The following procedure configures eMbedded Visual C++ for UltraLite development. This procedure has been tested on version 4.0 of that product; the process is similar for other versions.

♦ To configure eMbedded Visual C++ for UltraLite development

1. Start Microsoft eMbedded Visual C++.

   From the Start menu, choose All Programs ► Microsoft eMbedded Visual C++ 4.0 ► eMbedded Visual C++ 4.0

2. Configure eMbedded Visual C++ to search the appropriate directories for embedded SQL header files and UltraLite library files.

   a. Choose Tools ► Options.

      The Options dialog appears.

   b. Click the Directories tab.

   c. For each combination of target platform and CPU:

      - Choose Include Files from the Show Directories for dropdown list. Use the Browse button to include the **\h** directory located in the SQL Anywhere installation directory, so that the embedded SQL header files are accessible.

        install-dir`\h`

      - Choose Library Files from the Show Directories for dropdown list. Include the UltraLite *\lib* directory, located in a platform-specific directory. For example, for the Pocket PC 2002 emulator, specify the following (do not use the environment variable literally; use the full path name):

        install-dir`\UltraLite\ce\386\lib`

   d. Click OK.

3. Configure the project-level settings for this UltraLite tutorial project:

   a. Right-click esql files and select Settings.

      The Project Settings dialog appears.

   b. Select the appropriate platform under the Settings For dropdown list.

   c. Click the Link tab and add the appropriate runtime library to the Object/Library Modules box:

      Specify ulbase.lib

      For static UltraLite linked library specify ulrt.lib

      For dynamic UltraLite link library specify ulimp.lib

Copyright © 2007, iAnywhere Solutions, Inc.

Using UltraLite library or dynamic link library

Depending on the platform, your application may use the static UltraLite library (*ulrt.lib*) or the UltraLite dynamic link library (DLL). If you elect to use the DLL, you must copy the appropriate .dll file to the device and you must set UL_USE_DLL as a C/C++ Preprocessor definition.

For example, for the WCE ARM platform, copy *install-dir\ultralite\ce\arm\ulrt10.dll* to the *Windows* folder of the ARM-based CE device. For the WCE x86 (emulator) platform, copy *install-dir\ultralite\ce\386\ulrt10.dll* to the *Windows* folder on the emulator device.

Using the static library means that the application is not dependent on the presence of other files and may make distribution and deployment of the application easier.

Lesson 3: Write an embedded SQL source file

The following procedure creates a tutorial program that establishes a connection with the UltraLite CustDB sample database and executes a query.

♦ To build the tutorial Embedded SQL UltraLite application

1. Start Microsoft eMbedded Visual C++.

   From the Start menu, choose All Programs ▶ Microsoft eMbedded Visual C++ 4.0 ▶ eMbedded Visual C++ 4.0

2. In eMbedded Visual C++ create a new workspace named Ultralite:

   a. Select File ▶ New.

   b. Click the Workspaces tab.

   c. Choose Blank Workspace. Specify a workspace name Ultralite and specify *C:\tutorial* as the location to save this workspace. Click OK.

      The Ultralite workspace is added to the Workspace window.

3. Create a new project named esql and add it to the Ultralite workspace.

   a. Select File ▶ New.

   b. Click the Projects tab.

   c. The application types available depend on the SDKs you have installed. Choose an application type appropriate for the device you are using. For example, if you are using a Pocket PC 2002 device, choose WCE Pocket PC 2002 Application. Specify a project name esql and select Add To Current Workspace. Select the applicable CPUs. Click OK.

   d. Choose Create An Empty Project and click Finish.

      The project is saved in the *c:\tutorial\esql* folder.

4. Create the *sample.sqc* source file.

   a. Choose File ▶ New.

   b. Click the Files tab.

   c. Select C++ Source File.

   d. Select Add to Project and choose esql from the dropdown list.

   e. Name the file *sample.sqc*. Click OK.

   f. Copy the following source code into the file. The next section describes this tutorial code in detail.

```
#include <tchar.h>
#include <windows.h>
```


Copyright © 2007, iAnywhere Solutions, Inc.

```
EXEC SQL INCLUDE SQLCA;

int WINAPI WinMain( HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPTSTR lpCmdLine,
    int nShowCmd)
{
   /* Declare fields */
   EXEC SQL BEGIN DECLARE SECTION;
      long pid=1;
      long cost;
      TCHAR pname[31];
   EXEC SQL END DECLARE SECTION;

   TCHAR         output[200];
   TCHAR         result[10];

   /* Before working with data*/
   db_init(&sqlca);

   /* Connect to database */

   EXEC SQL WHENEVER SQLERROR GOTO error;

   EXEC SQL CONNECT USING 'dbf=\Windows\Start Menu\esqldb.udb';

   /* Fill table with data first */
   EXEC SQL INSERT INTO ULProduct(
         prod_id, price, prod_name)
      VALUES (1, 400, '4x8 Drywall x100');

   EXEC SQL INSERT INTO ULProduct (
         prod_id, price, prod_name)
      VALUES (2, 3000, '8''2x4 Studs x1000');

   /* Commit changes (INSERTs) to database */
   EXEC SQL COMMIT;


   /* Fetch row 1 from database */
   EXEC SQL SELECT price, prod_name
         INTO :cost, :pname
         FROM ULProduct
         WHERE prod_id= :pid;
         /* pid was initialized to 1, so get first row inserted above
*/

   /* Print query results */

   wsprintf( output
          , TEXT("Product id: %d\n     price: %d\n      name: %s")
          , pid , cost , pname );
   MessageBox(NULL, output, result, MB_OK);

   /* Preparing to exit: disconnect */
   EXEC SQL DISCONNECT;
   db_fini(&sqlca);
   return(0);

   /* Error handling.  If the row does not exist,
      or if an error occurs, -1 is returned */
   error:
    if((SQLCODE==SQLE_NOTFOUND)||(SQLCODE<0)) {
```

```
                wsprintf (output, TEXT("error: %d"), SQLCODE );
             MessageBox(NULL, output, result, MB_OK);
                return(-1);
        }
     }
```

g. Save the file.

5. Configure the *sample.sqc* source file settings to invoke the SQL preprocessor to preprocess the source file:

   a. Right-click *sample.sqc* in the Workspace window and choose Settings.

      The Project Settings dialog appears.

   b. From the Settings For dropdown list, choose All Configurations.

   c. In the Custom Build tab, enter the following statement in the Commands field. Ensure that the statement is entered all on one line.

      ```
      "%SQLANY10%\win32\sqlpp.exe" -q -u $(InputPath) sample.cpp
      ```

      This statement runs the SQL preprocessor *sqlpp* on the *sample.sqc* file, and writes the processed output to a file named *sample.cpp*. The SQL preprocessor translates SQL statements in the source file into C/C++ function calls.

      For more information about the SQL preprocessor, see "SQL preprocessor" [*SQL Anywhere Server - Programming*].

   d. Type **sample.cpp** in the Outputs field.

   e. Click OK to submit the changes.

6. Preprocess the *sample.sqc* file.

   a. Select *sample.sqc* in the Workspace window. Choose Build ► Compile sample.sqc. A *sample.cpp* file is created and saved in the *tutorial\esql* directory.

7. Add *sample.cpp* (the output of the SQL preprocessor) to the project:

   a. Right-click the Source Files folder in the Workspace window and choose Add Files to Folder.

   b. Browse to *c:\tutorial\esql\sample.cpp* and click OK.

      The *sample.cpp* file now appears inside the Source Files folder.

Explanation of the tutorial program

Although the tutorial program is simple, it contains elements that must be present in every embedded SQL source file used for database access.

The following list describes the key elements in the tutorial program. Use these steps as a guide when creating your own embedded SQL UltraLite application.

 Copyright © 2007, iAnywhere Solutions, Inc.

1. Include the appropriate header files.

   The tutorial program includes the header files *tchar.h* to support use of the TCHAR type specification and *windows.h*to support the use of the MessageBox function to display messages on the CE device.

2. Define the SQL communications area, sqlca.

   ```
   EXEC SQL INCLUDE SQLCA;
   ```

   This definition must be the first embedded SQL statement in each source file; place it at the end of your include list.

   Prefix SQL statements
   All SQL statements must be prefixed with the keywords EXEC SQL and must end with a semicolon.

3. Define host variables by creating a declaration section.

   Host variables are used to send values to the database server or receive values from the database server. In this tutorial, a query retrieves a product id, cost and product name from the database. Define the host variables as follows:

   ```
   EXEC SQL BEGIN DECLARE SECTION;
       long pid=1;
       long cost;
       TCHAR pname[31];
   EXEC SQL END DECLARE SECTION;
   ```

   See "Using host variables" on page 45.

4. Declare local program variables (in this tutorial application the local variables are used for building and displaying screen messages:

   ```
      TCHAR        output[200];
      TCHAR        result[10];
   ```

5. Call the embedded SQL library function db_init to initialize the UltraLite runtime library.

   Call this function as follows:

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

6. Define a label where control is to be transferred in case of a runtime error while processing a SQL statement:

   ```
   EXEC SQL WHENEVER SQLERROR GOTO error;
   ```

7. Connect to the database using the CONNECT statement.

   To connect to the UltraLite sample database, the application must supply the name and location of the database file. The database file name is *esqldb.udb* located in the root directory of the device.

   ```
   EXEC SQL CONNECT USING 'dbf=\esqldb.udb'
   ```

8. Insert data into database tables.

   ```
      /* Fill table with data first */
   ```

```
EXEC SQL INSERT INTO ULProduct(
      prod_id, price, prod_name)
   VALUES (1, 400, '4x8 Drywall x100');

EXEC SQL INSERT INTO ULProduct (
      prod_id, price, prod_name)
   VALUES (2, 3000, '8''2x4 Studs x1000');
         /* Commit changes (INSERTs) to database */
EXEC SQL COMMIT;
```

When you synchronize the remote database with the consolidated database, the tables are filled with values so that you may execute Select, Update, or Delete commands.

Rather than using synchronization, this code directly inserts data into the tables. Directly inserting data is a useful technique during the early stages of UltraLite development.

If you use synchronization and your application fails to execute a query, it might be due to a problem in the synchronization process or due to a mistake in your program. Locating the source of such a failure may be difficult. If you directly fill tables with data in your source code rather than relying on synchronization, if your application fails, you will know automatically that the failure is due to a mistake in your program.

After you have tested that there are no mistakes in your program, remove the insert statements and replace them with a call to the ULSynchronize function to synchronize the remote database with the consolidated database.

See “Lesson 5: Add synchronization to your application” on page 133.

9. Execute your SQL query.

```
/* Fetch row 1 from database */
EXEC SQL SELECT price, prod_name
         INTO :cost, :pname
         FROM ULProduct
         WHERE prod_id= :pid;
         /* pid was initialized to 1, get first row inserted */
```

The tutorial program executes a query that returns one row of results. The results are stored in the previously defined host variables cost and pname. In the embedded SQL statement the variable names are prefixed by a colon character to indicate to the sql preprocessor that the names are program variables.

10. Display the results of the query:

```
  /* Print query results */

  wsprintf( output, TEXT("Product id: %d\n     price: %d\n       name: %
s"), pid, cost, pname );
  MessageBox(NULL, output, result, MB_OK);
```

11. Provide the error handling routine.

The Embedded SQL statement

```
EXEC SQL WHENEVER SQLERROR GOTO error;
```


Copyright © 2007, iAnywhere Solutions, Inc.

that appears near the beginning of this tutorial program directs any error condition to be handled by transferring control to the program label `error`. The program defines this handler code to write a message containing the SQL error code from the field `SQLCODE` and exit the main function with a -1 result.

12. Disconnect from the database.

    Before disconnecting from the database the application must be careful to commit any changes (by issuing the EXEC SQL COMMIT statement, otherwise pending changes are discarded during disconnection.

    To disconnect, use the DISCONNECT statement as follows:

    ```
    EXEC SQL DISCONNECT;
    ```

13. To properly close the database interface and free resources that were used, call the function db_fini:

    ```
    db_fini(&sqlca);
    ```

Lesson 4: Build the embedded SQL UltraLite tutorial application

The following procedure uses the source file generated in the previous lesson, *sample.cpp*, to create the embedded SQL UltraLite tutorial application.

♦ To build the embedded SQL UltraLite tutorial application

1. Build the executable:

   ♦ Choose Build ► Build esql.exe.

   The esql executable is created. Depending on your settings, the executable may be created in a Debug directory within your tutorial directory.

2. Run the application:

   ♦ Choose Build ► Execute esql.exe.

   The executable is copied to the target device and begins execution. If there are no errors during execution, a message box appears displaying the contents of the first row of the product table. If any error occurs, an error message box is displayed containing the numerical SQL error code associated with the error. See "Error messages sorted by SQL Anywhere SQLCODE" [*SQL Anywhere 10 - Error Messages*].

 Copyright © 2007, iAnywhere Solutions, Inc.

Lesson 5: Add synchronization to your application

Once you have tested that your program functions properly, replace the code that manually inserts data into the ULProduct table with code to synchronize the remote database with the consolidated database. Synchronization fills the tables with data and the application can operate on that data.

Synchronization via TCP/IP

You can synchronize the remote database with the consolidated database using a TCP/IP socket connection (for example). Call ULEnableTcpipSynchronization (or a similar ULEnableXXX function) before issuing the call to ULSynchronize. For more information about types of synchronization, see "Synchronization parameters for UltraLite" [*MobiLink - Client Administration*].

To synchronize with the CustDB consolidated database, the application must supply a value for the employee ID field. This ID identifies an instance of an application to the MobiLink server. You may choose a value of 50, 51, 52, or 53. The MobiLink server uses this value to determine the download content, to record the synchronization state, and to recover from any interruptions during synchronization.

See "ULSynchronize function" on page 334.

Running the tutorial application with synchronization

After you have made changes to *sample.sqc*, rebuild *esql.exe*.

♦ To synchronize your application

1. Delete the INSERT embedded SQL commands and add the following code:

```
ULEnableTcpipSynchronization( &sqlca );
auto ul_synch_info synch_info;
ULInitSynchInfo( &synch_info );
synch_info.user_name = UL_TEXT("50");
synch_info.version = UL_TEXT("custdb 10.0");
ULSynchronize( &sqlca, &synch_info );
```

2. Preprocess *sample.sqc*.

   Choose Build ▶ Compile sample.sqc to recompile the altered file. When prompted, choose to reload *sample.cpp*.

3. Build the executable.

   Choose Build ▶ Build esql.exe to build the executable.

4. Ensure that the SQL Anywhere database server is still running.

5. Start the MobiLink server.

   At a command prompt, execute the following command on a single line:

```
mlsrv10 -c "DSN=SQL Anywhere 10 CustDB" -o ulsync.mls -v+ -x tcpip
```

6. Run the application:

   ♦ Choose Build ► Execute esql.exe to run the application.

     The remote database synchronizes with the consolidated database, filling the tables in the remote database with data. The query in the application is processed, and a row of query results appears on the screen.

Copyright © 2007, iAnywhere Solutions, Inc.

CHAPTER 10

Tutorial: Build an Application Using ODBC

Contents

Introduction to UltraLite ODBC 136
Lesson 1: Getting started 137
Lesson 2: Create an UltraLite database 139
Lesson 3: Connect to the database 140
Lesson 4: Insert data into the database 142
Lesson 5: Query the database 143

Copyright © 2007, iAnywhere Solutions, Inc.

Introduction to UltraLite ODBC

ODBC is a standard database programming interface. UltraLite supports a subset of the ODBC interface, together with extensions to permit synchronization. For a listing of the functions UltraLite supports, see “UltraLite ODBC API Reference” on page 335.

This section walks you through the creation of a simple UltraLite ODBC application. It does not provide an extensive guide to ODBC programming, as the main reference for ODBC is the Microsoft ODBC SDK documentation.

UltraLite ODBC does not share some features with other C/C++ interfaces. For a listing of functions that cannot be used from UltraLite ODBC, see “UltraLite C/C++ Common API Reference” on page 147.

Copyright © 2007, iAnywhere Solutions, Inc.

Lesson 1: Getting started

In this tutorial, you create a set of files, including source files and executable files. You should make a directory to hold these files. In the remainder of the tutorial, it is assumed the directory is *c:\tutorial\ulodbc*. If you choose a different name, use that name throughout.

The ODBC interface does not depend on any particular C/C++ compiler or development environment. The tutorial uses a makefile with Microsoft's nmake syntax. If you are using a different development environment, make the appropriate substitutions.

♦ Create and test your build environment

1. Add the following code to a file called *makefile* in your tutorial directory:

```
IncludeFolders= /I"$(SQLANY10)\h"

CompileOptions=/c /nologo /W3 /Od /Zi /DWIN32 /DUL_USE_DLL

LibraryFolders= \
/LIBPATH:"$(SQLANY10)\ultralite\win32\386\lib"

Libraries= ulimp.lib

LinkOptions=/NOLOGO /DEBUG

sample.exe: sample.obj
   link $(LinkOptions) sample.obj $(LibraryFolders) $(Libraries)

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

   These options compile a source file called *sample.cpp* into an executable *sample.exe*, using the UltraLite import library for Windows (*ulimp.lib*). They rely on the environment variable SQLANY10, which is defined as your SQL Anywhere installation directory. See “SQL Anywhere Environment Variables” [*SQL Anywhere Server - Database Administration*].

2. Add the following code to a file called *sample.cpp* in your tutorial directory:

```
#include "ulodbc.h"
#include <stdio.h>
#include <tchar.h>
int main() {
    return 0;
}
```

   This application simply returns 0 to the calling environment.

3. Compile and link *sample.cpp*.

   If you are using the Microsoft compiler, type **nmake** at a command prompt to compile and link your application. Otherwise, use the appropriate command for your development environment.

Compiling and linking the application confirms that your build environment is set up properly. You are now ready for the rest of the tutorial.

 Copyright © 2007, iAnywhere Solutions, Inc.

Lesson 2: Create an UltraLite database

This tutorial uses a simple one-table database.

This section assumes you can use the UltraLite plug-in for Sybase Central to create a database.

For more information about creating a database, see "Creating an UltraLite database from Sybase Central" [*UltraLite - Database Management and Reference*].

♦ Create UltraLite database

- Create a database using the Create Database wizard for UltraLite in Sybase Central.

  Create your database as follows:

  - ♦ **Database filename** *c:\tutorial\ulodbc\customer.udb*
  - ♦ **Table name** customer
  - ♦ **Columns in customer table**

Column Name	Data Type (Size)	Column allows NULL values?	Default value
id	integer	No	autoincrement
fname	varchar(15)	No	None
lname	varchar(20)	No	None
city	varchar(20)	Yes	None
phone	varchar(12)	Yes	555-1234

  - ♦ **Primary key** ascending id

Copyright © 2007, iAnywhere Solutions, Inc.

Lesson 3: Connect to the database

UltraLite uses standard ODBC programming methods to connect to a database. Each application requires an environment handle to manage the communication with UltraLite and a connection handle for a specific connection.

♦ Write code to allocate an environment handle

1. Add the opendb and closedb functions to *sample.cpp*:

```
static SQLHANDLE opendb( void ){
    SQLRETURN retn;
    SQLHANDLE henv;
    retn = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv );
    if( retn == SQL_SUCCESS ){
        _tprintf( "success in opendb: %d.\n", retn );
        return henv;
    } else {
        _tprintf( "error in opendb: %d.\n", retn );
        return henv;
    }
}

static void closedb( SQLHANDLE henv ){
    SQLRETURN retn;
    retn = SQLFreeHandle( SQL_HANDLE_ENV, henv );
}
```

   These functions do not connect to the database, they simply allocate the environment handle `henv` that manages UltraLite features.

2. Call opendb and closedb from the main function.

   Alter your main function in *sample.cpp* so that it reads as follows:

```
int main() {
    SQLHANDLE henv;
    henv = opendb();
    closedb( henv );
    return 0;
}
```

3. Compile, link, and run your application to confirm that the application builds properly.

For more information about the functions called in this procedure, see “SQLAllocHandle function” on page 337, and “SQLFreeHandle function” on page 348.

The next step is to connect to the UltraLite database

♦ Write code to connect to your database

1. Add connect and disconnect functions to *sample.cpp*:

```
static SQLHANDLE connect ( SQLHANDLE henv ){
```

 Copyright © 2007, iAnywhere Solutions, Inc.

```
        SQLRETURN retn;
        SQLHANDLE hcon;
        retn = SQLAllocHandle( SQL_HANDLE_DBC, henv, &hcon );
        retn = SQLConnect( hcon
          , (SQLTCHAR*)UL_TEXT(
              "dbf=customer.udb" )
          , SQL_NTS
          , (SQLTCHAR*)UL_TEXT( "DBA" )
          , SQL_NTS
          , (SQLTCHAR*)UL_TEXT( "sql" )
          , SQL_NTS );
        if( retn == SQL_SUCCESS ){
            _tprintf( "success in connect: %d.\n", retn );
            return hcon;
        } else {
            _tprintf( "error in connect: %d.\n", retn );
            return hcon;
        }
    }

    static void disconnect( SQLHANDLE hcon, SQLHANDLE henv ){
        SQLRETURN retn;
        retn = SQLDisconnect( hcon );
        retn = SQLFreeHandle( SQL_HANDLE_DBC, hcon );
    }
```

2. Call connect and disconnect from the main function.

   Alter your main function in *sample.cpp* so that it reads as follows:

```
    int main() {
        SQLHANDLE henv;
        SQLHANDLE hcon;
        henv = opendb();
        hcon = connect( henv );
        disconnect( hcon, henv );
        closedb( henv );
        return 0;
    }
```

3. Compile, link, and run your application to confirm that the application builds properly.

For more information about the functions called in this procedure, see "SQLConnect function" on page 340, and "SQLDisconnect function" on page 342.

You now have an application that connects to and disconnects from a database. The next step is to add some data to the database.

Lesson 4: Insert data into the database

ODBC provides a set of functions to carry out operations on the database. This lesson uses the simplest statement, SQLExecDirect.

♦ Write code to insert data into the database

1. Add an insert function to *sample.cpp*:

```
static ul_bool insert( SQLHANDLE hcon )
{
    SQLRETURN retn;
    SQLHANDLE hstmt;
    retn = SQLAllocHandle( SQL_HANDLE_STMT, hcon, &hstmt );
    static const ul_char * sql = UL_TEXT(
     "INSERT customer( id, fname, lname ) VALUES ( 42, 'jane',
'doe' )" );
    retn = SQLExecDirect( hstmt, (SQLTCHAR*)sql, SQL_NTS );
    if( retn == SQL_SUCCESS ) {
        _tprintf( "success in insert.\n" );
    } else {
        _tprintf( "error in insert: %d.\n", retn );
    retn = SQLFreeHandle( SQL_HANDLE_STMT, hstmt );
    hstmt = 0;
    }
    return retn == SQL_SUCCESS;
}
```

2. Call insert from the main function.

   Alter your main function in *sample.cpp* so that it reads as follows:

```
int main() {
    SQLHANDLE henv;
    SQLHANDLE hcon;
    henv = opendb();
    hcon = connect( henv );
    insert( hcon );
    disconnect( hcon, henv );
    closedb( henv );
    return 0;
}
```

3. Compile, link, and run your application to confirm that the application builds properly.

   You should see that the application reports success on inserting the data.

For more information about the function called in this procedure, see "SQLExecDirect function" on page 344.

 Copyright © 2007, iAnywhere Solutions, Inc.

Lesson 5: Query the database

In order to process query result sets, ODBC requires that statements be prepared before they are executed. In this lesson you prepare and execute a statement, and print out the results.

♦ Write code to query the database

1. Add prepare, execute, and fetch functions to *sample.cpp*:

```
static SQLHANDLE prepare( SQLHANDLE hcon ){
    SQLRETURN retn;
    SQLHANDLE hstmt;
    static const ul_char * sql =
        UL_TEXT( "SELECT id, fname, lname FROM customer" );
    retn = SQLAllocHandle( SQL_HANDLE_STMT, hcon, &hstmt );
    retn = SQLPrepare( hstmt, (SQLTCHAR*)sql, SQL_NTS );
    if( retn == SQL_SUCCESS ) {
        _tprintf( "success in prepare.\n" );
    } else {
        _tprintf( "error in prepare: %d.\n", retn );
        retn = SQLFreeHandle( SQL_HANDLE_STMT, hstmt );
        hstmt = 0;
    }
    return hstmt;
}
```

The prepare function does not execute the SQL statement.

```
static ul_bool execute( SQLHANDLE hstmt )
{
    SQLRETURN retn;
    retn = SQLExecute( hstmt );
    if( retn == SQL_SUCCESS ) {
        _tprintf( "success in execute.\n" );
    } else {
        _tprintf( "error in execute: %d.\n", retn );
    retn = SQLFreeHandle( SQL_HANDLE_STMT, hstmt );
    hstmt = 0;
    }
    return retn == SQL_SUCCESS;
}
```

The execute function executes the query, but does not make the result set directly available to the client application. Your application must explicitly fetch the rows it needs from the result set.

```
static ul_bool fetch( SQLHANDLE hstmt )
{
#define NAME_LEN 20
    SQLCHAR       fName[NAME_LEN], lName[NAME_LEN];
    SQLINTEGER    id;
    SQLINTEGER       cbID = 0, cbFName = SQL_NTS, cbLName = SQL_NTS;
    SQLRETURN     retn;

    SQLBindCol( hstmt, 1, SQL_C_ULONG, &id, 0, &cbID );
    SQLBindCol( hstmt, 2, SQL_C_CHAR,
                fName, sizeof(fName), &cbFName );
```

```
        SQLBindCol( hstmt, 3, SQL_C_CHAR,
                    lName, sizeof(lName), &cbLName );
        while ( ( retn = SQLFetch( hstmt ) ) != SQL_NO_DATA ){
            if (retn == SQL_SUCCESS || retn == SQL_SUCCESS_WITH_INFO){
                fName[ cbFName ] = '\0';
                lName[ cbLName ] = '\0';
                _tprintf( "%20s %d %20s\n", fName, id, lName );
            } else {
                _tprintf ( "error while fetching: %d.\n", retn );
                break;
            }
        }
        return retn == SQL_SUCCESS;
    }
```

The values are fetched into variables that have been bound to the column. String variables are not returned with a null terminator, and so the null terminator is added for printing purposes. The length of the actual string that was returned is available in the final parameter of SQLBindCol.

2. Call prepare, execute, and fetch from the main function.

   Alter your main function in *sample.cpp* so that it reads as follows:

```
    int main() {
        SQLHANDLE henv;
        SQLHANDLE hcon;
        SQLHANDLE hstmt;

        henv = opendb();
        hcon = connect( henv );
        insert( hcon );
        hstmt = prepare( hcon );
        execute( hstmt );
        fetch( hstmt );
        SQLFreeHandle( SQL_HANDLE_STMT, hstmt );
        disconnect( hcon );
        closedb( henv );
        return 0;
    }
```

3. Compile, link, and run your application to confirm that the application builds properly.

   You should see that the application displays the data from the row you inserted.

For more information about the functions called in this procedure, see:

- “SQLPrepare function” on page 354
- “SQLExecute function” on page 345
- “SQLBindCol function” on page 338
- “SQLFetch function” on page 346

This completes the tutorial.

 Copyright © 2007, iAnywhere Solutions, Inc.

Part IV. API Reference

This part provides API reference material for UltraLite C/C++ programmers.

CHAPTER 11

UltraLite C/C++ Common API Reference

Contents

Introduction to UltraLite C/C++ common API 148
Callback function for ULRegisterErrorCallback 149
MLFileTransfer function 151
ULCreateDatabase function 154
ULEnableEccSyncEncryption function 156
ULEnableFileDB function (deprecated) 157
ULEnableFIPSStrongEncryption function 158
ULEnableHttpSynchronization function 159
ULEnableHttpsSynchronization function 160
ULEnablePalmRecordDB function (deprecated) 161
ULEnableRsaFipsSyncEncryption function 162
ULEnableRsaSyncEncryption function 163
ULEnableStrongEncryption function 164
ULEnableTcpipSynchronization function 165
ULEnableTlsSynchronization function 166
ULEnableUserAuthentication function (deprecated) 167
ULEnableZlibSyncCompression function 168
ULInitDatabaseManager 169
ULInitDatabaseManagerNoSQL 170
ULRegisterErrorCallback function 171
Macros and compiler directives for UltraLite C/C++ applications 173

Introduction to UltraLite C/C++ common API

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

See "Common Features of UltraLite C/C++ Interfaces" on page 11.

 Copyright © 2007, iAnywhere Solutions, Inc.

Callback function for ULRegisterErrorCallback

Handles errors that the UltraLite runtime signals to your application.

For a description of error handling using this technique, see "ULRegisterErrorCallback function" on page 171.

Syntax

ul_error_action **UL_GENNED_FN_MOD** *error-callback-function* **(**
SQLCA * *sqlca*,
ul_void * *user_data*,
ul_char * *buffer*
);

Parameters

- **error-callback-function** The name of your function. You must supply the name to ULRegisterErrorCallback.
- **sqlca** A pointer to the SQL communications area (SQLCA).

  The SQLCA contains the SQL code in `sqlca->sqlcode`. Any error parameters have already been retrieved from the SQLCA and stored in *buffer*.

  This sqlca pointer does not necessarily point to the SQLCA in your application, and cannot be used to call back to UltraLite. It is used only to communicate the SQL code to the callback.

  In the C++ component, use the Sqlca.GetCA method.
- **user_data** The user data supplied to ULRegisterErrorCallback. UltraLite does not change this data in any way. Because the callback function may be signaled anywhere in your application, the user_data argument is an alternative to creating a global variable.
- **buffer** The buffer supplied when the callback function was registered. UltraLite fills the buffer with a string, which holds any substitution parameters for the error message. To keep UltraLite as small as possible, UltraLite does not supply error messages themselves. The substitution parameters depend on the specific error. For more information about error parameters for SQL errors, see "Database Error Messages" [*SQL Anywhere 10 - Error Messages*].

Return value

Returns one of the following actions:

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

See also

- "ULRegisterErrorCallback function" on page 171

- "Error messages sorted by Sybase error code" [*SQL Anywhere 10 - Error Messages*]

Copyright © 2007, iAnywhere Solutions, Inc.

MLFileTransfer function

Downloads a file from a MobiLink server with the MobiLink interface.

Syntax

ul_bool **MLFileTransfer (** ml_file_transfer_info * *info* **);**

Parameters

info A structure containing the file transfer information.

ML File Transfer parameters

The ML File Transfer parameters are members of a structure that is passed as a parameter to the MLFileTransfer function. The ml_file_transfer_info structure is defined in a header file named *mlfiletransfer.h*. The individual fields of the structure are specified as follows:

filename Required. The file name to be transferred from the server running MobiLink. MobiLink searches the *username* subdirectory first, before defaulting to the root directory. See "-ftr option" [*MobiLink - Server Administration*].

If the file cannot be found, an error is set in the error field. The file name must not include any drive or path information, or MobiLink cannot find it.

dest_path 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 CE, if dest_path is empty, the file is stored in the root (\) directory of the device.
- On the desktop and on Symbian OS, if the dest_path is empty, the file is stored in the user's current directory.
- On Palm OS, when downloading to the device external storage, prefix the dest_path with **vfs:**. The path should then be specified with the platform file naming convention. See "Palm OS" [*UltraLite - Database Management and Reference*].

  If dest_path field is empty, MLFileTransfer assumes it is downloading a Palm record database (*.pdb*).

dest_filename The local name for the downloaded file. If this parameter is empty, the value in file name is used.

stream Required. The protocol can be one of: TCPIP, TLS, HTTP, or HTTPS. See "Stream Type synchronization parameter" [*MobiLink - Client Administration*].

stream_parms The protocol options for a given stream. See "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*].

username Required. MobiLink user name.

password The password for the MobiLink user name.

version Required. The MobiLink script version.

observer A callback can be provided to observe file download progress through the 'observer' field. For more details, see description of Callback Function that follows.

user_data The application-specific information made available to the synchronization observer. See "User Data synchronization parameter" [*MobiLink - Client Administration*].

force_download Set to true to download the file even if the timestamp indicates it is already present. Set to false to downloaded only if the server version and the local version are different. In this case, the server version of the file overwrites the client version. Any previous file of the same name on the client is discarded before the file is downloaded. MLFileTransfer compares the server and client version of the file by computing a cryptographic hash value for each file; the hash values are identical only if the files are identical in content.

enable_resume If set to true, MLFileTransfer 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 force_download parameter overrides this parameter.

num_auth_parms The number of authentication parameters being passed to authentication parameters in MobiLink events. See "Number of Authentication Parameters parameter" [*MobiLink - Client Administration*].

auth_parms Supplies parameters to authentication parameters in MobiLink events. See "Authentication Parameters synchronization parameter" [*MobiLink - Client Administration*].

downloaded_file Is set to one of the following:

- 1 if the file was successfully downloaded.
- 0 if an error occurs. An error occurs if the file is already up-to-date when MLFileTransfer is invoked. In this case, the function returns true rather than false. For the Palm OS, when downloading a record database (*.pdb*) file, MLFileTransfer always downloads the file, whether the file is up-to-date or not.

auth_status Reports the status of MobiLink user authentication. The MobiLink server provides this information to the client. See "Authentication Status synchronization parameter" [*MobiLink - Client Administration*].

auth_value Reports results of a custom MobiLink user authentication script. The MobiLink server provides this information to the client. See "Authentication Value synchronization parameter" [*MobiLink - Client Administration*].

file_auth_code Contains the return code of the optional authenticate_file_transfer script on the server.

error Contains information about any error that occurs.

Returns

- **ul_true** The file was successfully downloaded.
- **ul_false** The file was not downloaded successfully. You can supply Error information in the error field of the ml_file_transfer_info structure. Incomplete file transfers are resumable.

 Copyright © 2007, iAnywhere Solutions, Inc.

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.

Callback Function

The callback to observe file transfer progress through the observer parameter has the following prototype:

```
typedef void(*ml_file_transfer_observer_fn)( ml_file_transfer_status *
status );
```

The ml_file_transfer_status object passed to the callback is defined as follows:

```
typedef struct ml_file_transfer_status {
    asa_uint64                  file_size;
    asa_uint64                  bytes_received;
    asa_uint64                  resumed_at_size;
    ml_file_transfer_info_a *   info;
    asa_uint16                  flags;
    asa_uint8                   stop;
} ml_file_transfer_status;
```

file_size The total size in bytes of the file being downloaded.

bytes_received Indicates how much of the file has been downloaded so far, including previous syncs if the download is resumed.

resumed_at_size Used with download resumption and indicates at what point the current download resumed.

info Points to the info object passed to MLFileTransfer. You can access the user_data parameter through this pointer.

flags Provides additional information. The value MLFT_STATUS_FLAG_IS_BLOCKING is set when MLFileTransfer is blocking on a network call and the download status has not changed since the last time the observer function was called.

stop May be set to true to cancel the current download. You can resume the download in a subsequent call to MLFileTransfer, but only if you have set the enable_resume parameter.

ULCreateDatabase function

Creates an UltraLite database.

Syntax

ul_bool **ULCreateDatabase (** SQLCA * *sqlca*,
ul_char * *connection-parms*,
void const * *collation*,
ul_char * *creation-parms*,
void * *reserved*
);

Parameters

sqlca A pointer to the initialized SQLCA.

connection-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. For a complete list, see "UltraLite Connection String Parameters Reference" [*UltraLite - Database Management and Reference*].

collation
The desired collation sequence for the database. You can get a collation sequence by calling the appropriate function. For example:

```
void const * collation = ULGetCollation_1250LATIN2();
```

The function name is constructed by prefixing the name of the desired collation with **ULGetCollation_**. For a list of all available collation functions see *install-dir\h\ulgetcoll.h*. You must include this file in programs that call any of the **ULGetCollation_** functions.

creation-parms
A semicolon separated string of connection parameters, which are set as keyword value pairs. For example:

```
page_size=2048;obfuscate=yes
```

For a complete list, see "Choosing creation-time database properties for UltraLite" [*UltraLite - Database Management and Reference*].

reserved This parameter is reserved for future use.

Returns

- **ul_true** Indicates that database was successfully created.
- **ul_false** A detailed error message is defined by the SQLCODE field in the SQLCA. Typically this 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-parms 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).

 Copyright © 2007, iAnywhere Solutions, Inc.

- The creation-parms 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.

Example

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
  };
```

ULEnableEccSyncEncryption function

Enables ECC encryption for SSL or TLS streams. 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.

Syntax

void **ULEnableEccSyncEncryption(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Remarks

You can use this function in C++ API applications as well as 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 10 - Introduction*].

See also

- "ULEnableZlibSyncCompression function" on page 168
- "ULEnableRsaFipsSyncEncryption function" on page 162

 Copyright © 2007, iAnywhere Solutions, Inc.

ULEnableFileDB function (deprecated)

Use a file-based data store on a device operating the Palm OS version 4.0 or later.

Deprecated function
This function is not required in UltraLite beginning with version 10. The file name determines the location of the data store on a device operating under Palm OS.

Syntax

void **ULEnableFileDB(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the SQLCA. This argument is supplied in C++ Component applications.

In the C++ Component use the Sqlca.GetCA method.

Remarks

To use the file-based data store on a Palm expansion card, an UltraLite application must call ULEnableFileDB to load the persistent storage file-I/O modules before connecting to the database.

Examples

The following embedded SQL code illustrates the use of ULEnableFileDB.

```
db_init( & sqlca );
ULEnableFileDB( &sqlca );
// connection code here
if( SQLCODE == SQLE_CONNECTION_RESTORED ){
    // connection was restored
    // cursor is already open
} else {
    // open cursor
}
```

See also

- “ULEnablePalmRecordDB function (deprecated)” on page 161

ULEnableFIPSStrongEncryption function

Enables FIPS-based strong encryption for the database. Calling this function causes the appropriate encryption routines to be included in the application and increases the size of the application code accordingly.

Syntax

void **ULEnableFIPSStrongEncryption(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Remarks

You can use this function in C++ API applications as well as 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 10 - Introduction*].

See also

♦ "ULEnableStrongEncryption function" on page 164

Copyright © 2007, iAnywhere Solutions, Inc.

ULEnableHttpSynchronization function

Enables HTTP synchronization.

Syntax

void **ULEnableHttpSynchronization(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Remarks

You can use this function in C++ API applications as well as 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.

ULEnableHttpsSynchronization function

Enables the SSL synchronization stream for HTTP.

Syntax

void **ULEnableHttpsSynchronization(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Remarks

You can use this function in C++ API applications as well as 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.

Example

```
ULEnableHttpsSynchronization( sqlca );
ULEnableRsaSyncEncryption( sqlca );
synch_info.stream = "https";
synch_info.stream_parms = "tls_type=rsa";
// rsa is default, so setting this parameter is optional
conn->Synchronize( sqlca );
```

 Copyright © 2007, iAnywhere Solutions, Inc.

ULEnablePalmRecordDB function (deprecated)

Use a standard record-based data store on a device operating the Palm Computing Platform.

> **Deprecated function**
> This function is not required in UltraLite beginning with version 10. The file name determines the location of the data store on a device operating under Palm OS.

Syntax

void **ULEnablePalmRecordDB(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the SQLCA. This argument is supplied even in C++ Component and C++ API applications.

In the C++ Component use the Sqlca.GetCA method.

Examples

The following embedded SQL code illustrates the use of ULEnablePalmRecordDB.

```
db_init( & sqlca );
ULEnablePalmRecordDB( &sqlca );
// connection code here
if( SQLCODE == SQLE_CONNECTION_RESTORED ){
    // connection was restored
    // cursor is already open
} else {
    // open cursor
}
```

See also

- "ULEnableFileDB function (deprecated)" on page 157

ULEnableRsaFipsSyncEncryption function

Enables RSA FIPS encryption for SSL or TLS streams. 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.

Syntax

void **ULEnableRsaFipsSyncEncryption(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Remarks

You can use this function in C++ API applications as well as 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

- "ULEnableRsaSyncEncryption function" on page 163
- "ULEnableEccSyncEncryption function" on page 156

Copyright © 2007, iAnywhere Solutions, Inc.

ULEnableRsaSyncEncryption function

Enables RSA encryption for SSL or TLS streams. 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.

Syntax

void **ULEnableRsaSyncEncryption(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Remarks

You can use this function in C++ API applications as well as 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

- "ULEnableEccSyncEncryption function" on page 156
- "ULEnableRsaFipsSyncEncryption function" on page 162

ULEnableStrongEncryption function

Enables strong encryption.

Syntax

void **ULEnableStrongEncryption(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Remarks

You can use this function in C++ API applications as well as embedded SQL applications. You must call this function before db_init or ULInitDatabaseManager.

> **Note**
> Calling this function causes the encryption routines to be included in the application and increases the size of the application code accordingly.

Copyright © 2007, iAnywhere Solutions, Inc.

ULEnableTcpipSynchronization function

Enables TCP/IP synchronization.

Syntax

void **ULEnableTcpipSynchronization(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Remarks

You can use this function in C++ API applications as well as 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.

Copyright © 2007, iAnywhere Solutions, Inc.

ULEnableTlsSynchronization function

Enables TLS synchronization.

Syntax

void **ULEnableTlsSynchronization(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Remarks

You can use this function in C++ API applications as well as 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.

 Copyright © 2007, iAnywhere Solutions, Inc.

ULEnableUserAuthentication function (deprecated)

Enable user authentication in the UltraLite application.

> **Deprecated function**
> In versions of UltraLite prior to 10, applications could invoke this function to enable user authentication. Beginning with version 10 of UltraLite, user authentication is enabled by default and cannot be disabled.

Syntax

void **ULEnableUserAuthentication(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the SQLCA.

In the C++ API use the Sqlca.GetCA method.

Remarks

If this function is not called, you do not need your user to supply a user ID or password to access an UltraLite database. Once this function is called your application must instead supply a valid user ID and password directly. UltraLite databases are created with a single authenticated user ID **DBA** which has initial password **sql** (lowercase).

This function can be used in C++ API applications as well as embedded SQL applications. You must call it before a connection is opened.

ULEnableZlibSyncCompression function

Enables ZLIB compression for a synchronization stream.

Syntax

void **ULEnableZlibSyncCompression(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Remarks

You can use this function in C++ API applications as well as 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.

 Copyright © 2007, iAnywhere Solutions, Inc.

ULInitDatabaseManager

Initializes the UltraLite database manager.

Syntax

pointer **ULInitDatabaseManager(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Returns

- A pointer to the database manager.
- NULL if the function fails.

Remarks

The function fails if you have not previously initialized the Database Manager and did not issue a Shutdown.

Copyright © 2007, iAnywhere Solutions, Inc.

ULInitDatabaseManagerNoSQL

Initialize the UltraLite database manager and exclude support for processing SQL statements (this can significantly reduce the application run time size).

Syntax

pointer **ULInitDatabaseManagerNoSQL(** SQLCA * *sqlca* **);**

Parameters

sqlca A pointer to the initialized SQLCA.

In the C++ API use the Sqlca.GetCA method.

Returns

- A pointer to the database manager.
- NULL if the function fails.

Remarks

The function fails if you have not previously initialized the Database Manager and did not issue a Shutdown.

The application must access data through the Table API and cannot use SQL statements. You cannot use the call if the database schema contains publication predicates; use ULInitDatabaseManager instead.

 Copyright © 2007, iAnywhere Solutions, Inc.

ULRegisterErrorCallback function

Registers a callback function that handles errors.

Syntax

void **ULRegisterErrorCallback (**
 SQLCA * *sqlca*,
 ul_error_callback_fn *callback*,
 ul_void * *user_data*,
 ul_char * *buffer*,
 size_t *len*
);

Parameters

- **sqlca** A pointer to the SQL Communications Area.

  In the C++ API use the Sqlca.GetCA method.

- **callback** The name of your callback function. For more information about the prototype of the function, see "Callback function for ULRegisterErrorCallback" on page 149.

  A callback value of UL_NULL disables any previously registered callback function.

- **user_data** An alternative to global variables to make any context information globally accessible. This is required because you can call the callback function from any location in your application. UltraLite does not modify the supplied data; it simply passes it to your callback function when it is invoked.

  You can declare any data type and cast it into the correct type in your callback function. For example, you could include a line of the following form in your callback function:

  ```
  MyContextType * context = (MyContextType *)user_data;
  ```

- **buffer** A character array holding the substitution parameters for the error message, including a null terminator. To keep UltraLite as small as possible, UltraLite does not supply error messages. The substitution parameters depend on the specific error. For a complete list, see "Database Error Messages" [*SQL Anywhere 10 - Error Messages*].

  The buffer must exist as long as UltraLite is active. Supply UL_NULL if you do not want to receive parameter information.

- **len** The length of the buffer (preceding parameter), in ul_char characters. A value of 100 is large enough to hold most error parameters. If the buffer is too small, the parameters are truncated.

Remarks

Once you call this function, the user-supplied callback function is called whenever UltraLite signals an error. You should therefore call ULRegisterErrorCallback immediately after initializing the SQLCA.

Error handling with this callback technique is particularly helpful during development, as it ensures that your application is notified of any and all errors that occur. However, the callback function does not control execution flow, so the application should check the SQLCODE field in the SQLCA after all calls to UltraLite functions.

Example

The following code registers a callback function for an UltraLite C++ Component application:

```
int main(){
    ul_char buffer[100];
    DatabaseManager * dm;
    Connection * conn;
    Sqlca.Initialize();
    ULRegisterErrorCallback(
        Sqlca.GetCA(),
        MyErrorCallBack,
        UL_NULL,
        buffer,
        sizeof (buffer) );
    dm = ULInitDatabaseManager( Sqlca );
    ...
```

The following is a sample callback function:

```
 ul_error_action UL_GENNED_FN_MOD MyErrorCallBack(
    SQLCA *     Sqlca,
    ul_void *    user_data,
    ul_char *    message_param )
{
    ul_error_action rc = 0;
    (void) user_data;

    switch( Sqlca->sqlcode ){
        // The following error is used for flow control - don't report it
here
        case SQLE_NOTFOUND:
        break;

        case SQLE_ULTRALITE_DATABASE_NOT_FOUND:
          _tprintf( _TEXT( "Error %ld: Database file %s not found\n" ), Sqlca-
>sqlcode, message_param );
        break;

        default:
            _tprintf( _TEXT( "Error %ld: %s\n" ), Sqlca->sqlcode,
message_param );
        break;
        }
    return rc;
}
```

See also

- “Error messages sorted by Sybase error code” [*SQL Anywhere 10 - Error Messages*]
- “Callback function for ULRegisterErrorCallback” on page 149

Copyright © 2007, iAnywhere Solutions, Inc.

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"$(VCDIR)\include" \
/I"$(SQLANY10)\h"

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

  VCDIR is your Visual C++ directory and *SQLANY10* is your SQL Anywhere installation directory.

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

UL_AS_SYNCHRONIZE macro

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

Remarks

Applies to Windows CE applications using ActiveSync only.

See also

- "Adding ActiveSync synchronization to your application" on page 102

UL_SYNC_ALL macro

Provides a publication mask that refers to all tables in the database, including those not in publications.

See also

- "PublicationMask synchronization parameter" [*MobiLink - Client Administration*]
- "ULGetLastDownloadTime function" on page 315
- "ULCountUploadRows function" on page 311
- "UL_SYNC_ALL_PUBS macro" on page 174

UL_SYNC_ALL_PUBS macro

Provides a publication mask that refers to all tables in the database that are in publications.

See also

- "PublicationMask synchronization parameter" [*MobiLink - Client Administration*]
- "ULGetLastDownloadTime function" on page 315
- "ULCountUploadRows function" on page 311
- "UL_SYNC_ALL macro" on page 173

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 CE and Windows applications.

UNDER_CE macro

By default, this macro is defined in all new eMbedded Visual C++ projects by the Microsoft eMbedded Visual C++ compiler.

Remarks

Applies to Windows CE applications.

Example

```
/D UNDER_CE=$(CEVersion)
```

See also

- "Developing UltraLite Applications for Windows CE" on page 93

UNDER_PALM_OS macro

This macro is defined in the *ulpalmos.h* header file included in UltraLite Palm OS applications by the UltraLite plug-in. See "Using the UltraLite plug-in for CodeWarrior" on page 77.

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Applies to a compiler directive for Palm OS only.

See also

- "Developing UltraLite Applications for the Palm OS" on page 73

Copyright © 2007, iAnywhere Solutions, Inc.

CHAPTER 12

UltraLite C++ API Reference

Contents

ul_synch_info_a struct 179
ul_synch_info_w2 struct 186
ul_synch_result struct 193
ul_synch_stats struct 196
ul_synch_status struct 198
ULSqlca class 200
ULSqlcaBase class 202
ULSqlcaWrap class 207
UltraLite_Connection class 209
UltraLite_Connection_iface class 210
UltraLite_Cursor_iface class 223
UltraLite_DatabaseManager class 230
UltraLite_DatabaseManager_iface class 231
UltraLite_DatabaseSchema class 234
UltraLite_DatabaseSchema_iface class 235
UltraLite_IndexSchema class 239
UltraLite_IndexSchema_iface class 240
UltraLite_PreparedStatement class 245
UltraLite_PreparedStatement_iface class 246
UltraLite_ResultSet class 250
UltraLite_ResultSet_iface class 251
UltraLite_ResultSetSchema class 252
UltraLite_RowSchema_iface class 253
UltraLite_SQLObject_iface class 258
UltraLite_StreamReader class 260
UltraLite_StreamReader_iface class 261
UltraLite_StreamWriter class 264
UltraLite_Table class 265
UltraLite_Table_iface class 266

UltraLite_TableSchema class 272
UltraLite_TableSchema_iface class 273
ULValue class 283

Copyright © 2007, iAnywhere Solutions, Inc.

ul_synch_info_a struct

Syntax

public **ul_synch_info_a**

Remarks

The structure used to describe synchronization data.

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).

Members

All members of ul_synch_info_a, including all inherited members.

- "auth_parms variable" on page 180
- "auth_status variable" on page 180
- "auth_value variable" on page 180
- "checkpoint_store variable" on page 180
- "disable_concurrency variable" on page 180
- "download_only variable" on page 181
- "ignored_rows variable" on page 181
- "init_verify variable" on page 181
- "keep_partial_download variable" on page 181
- "new_password variable" on page 181
- "num_auth_parms variable" on page 182
- "observer variable" on page 182
- "partial_download_retained variable" on page 182
- "password variable" on page 182
- "ping variable" on page 182
- "publication variable" on page 183
- "resume_partial_download variable" on page 183
- "send_column_names variable" on page 183
- "send_download_ack variable" on page 183
- "stream variable" on page 183
- "stream_error variable" on page 184
- "stream_parms variable" on page 184
- "table_order variable" on page 184
- "upload_ok variable" on page 184
- "upload_only variable" on page 184
- "user_data variable" on page 185
- "user_name variable" on page 185
- "version variable" on page 185

auth_parms variable

Synopsis

char ** **ul_synch_info_a::auth_parms**

Remarks

An array of authentication parameters in MobiLink events.

auth_status variable

Synopsis

ul_auth_status **ul_synch_info_a::auth_status**

Remarks

The status of MobiLink user authentication. The MobiLink server provides this information to the client.

auth_value variable

Synopsis

ul_s_long **ul_synch_info_a::auth_value**

Remarks

The results of a custom MobiLink user authentication script. The MobiLink server provides this information to the client to determine the authentication status.

checkpoint_store variable

Synopsis

ul_bool **ul_synch_info_a::checkpoint_store**

Remarks

Adds additional checkpoints of the database during synchronization to limit database growth during the synchronization process.

disable_concurrency variable

Synopsis

ul_bool **ul_synch_info_a::disable_concurrency**

Remarks

Disallow database access from other threads during synchronization.

Copyright © 2007, iAnywhere Solutions, Inc.

download_only variable

Synopsis

ul_bool **ul_synch_info_a::download_only**

Remarks

Do not upload any changes from the UltraLite database during the current synchronization.

ignored_rows variable

Synopsis

ul_bool **ul_synch_info_a::ignored_rows**

Remarks

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 variable

Synopsis

ul_synch_info_a * **ul_synch_info_a::init_verify**

Remarks

Initialize verification.

keep_partial_download variable

Synopsis

ul_bool **ul_synch_info_a::keep_partial_download**

Remarks

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 variable

Synopsis

char * **ul_synch_info_a::new_password**

Remarks

A string specifying a new MobiLink password associated with the user name. This parameter is optional.

num_auth_parms variable

Synopsis

ul_byte **ul_synch_info_a::num_auth_parms**

Remarks

The number of authentication parameters being passed to authentication parameters in MobiLink events.

observer variable

Synopsis

ul_synch_observer_fn **ul_synch_info_a::observer**

Remarks

A pointer to a callback function or event handler that monitors synchronization. This parameter is optional.

partial_download_retained variable

Synopsis

ul_bool **ul_synch_info_a::partial_download_retained**

Remarks

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 variable

Synopsis

char * **ul_synch_info_a::password**

Remarks

A string specifying the existing MobiLink password associated with the user name. This parameter is optional.

ping variable

Synopsis

ul_bool **ul_synch_info_a::ping**

Remarks

Confirm communications between the UltraLite client and the MobiLink server. When this parameter is set to true, no synchronization takes place.

Copyright © 2007, iAnywhere Solutions, Inc.

publication variable

Synopsis

ul_publication_mask **ul_synch_info_a::publication**

Remarks

An OR'd list of publications for which the last download time is retrieved. The set is supplied as a mask. This parameter is optional.

resume_partial_download variable

Synopsis

ul_bool **ul_synch_info_a::resume_partial_download**

Remarks

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 variable

Synopsis

ul_bool **ul_synch_info_a::send_column_names**

Remarks

Instructs the application that column names should be sent to the Mobilikn server in the upload.

send_download_ack variable

Synopsis

ul_bool **ul_synch_info_a::send_download_ack**

Remarks

Instructs the MobiLink server whether or not the client will provide a download acknowledgment.

stream variable

Synopsis

const char * **ul_synch_info_a::stream**

Remarks

The MobiLink network protocol to use for synchronization.

Copyright © 2007, iAnywhere Solutions, Inc.

stream_error variable

Synopsis

ul_stream_error **ul_synch_info_a::stream_error**

Remarks

The structure to hold communications error reporting information.

stream_parms variable

Synopsis

char * **ul_synch_info_a::stream_parms**

Remarks

The options to configure the network protocol you selected.

table_order variable

Synopsis

char * **ul_synch_info_a::table_order**

Remarks

The table order required for priority synchronization, if the UltraLite default table ordering is not suitable for your deployment. This parameter is optional.

upload_ok variable

Synopsis

ul_bool **ul_synch_info_a::upload_ok**

Remarks

The status of data uploaded to the MobiLink server. This field reports true if upload succeeded.

upload_only variable

Synopsis

ul_bool **ul_synch_info_a::upload_only**

Remarks

Do not download any changes from the consolidated database during the current synchronization. This can save communication time, especially over slow communication links.

 Copyright © 2007, iAnywhere Solutions, Inc.

user_data variable

Synopsis

ul_void * **ul_synch_info_a::user_data**

Remarks

Make application-specific information available to the synchronization observer. This parameter is optional.

user_name variable

Synopsis

char * **ul_synch_info_a::user_name**

Remarks

A string that the MobiLink server uses to identify a unique MobiLink user.

version variable

Synopsis

char * **ul_synch_info_a::version**

Remarks

The version string allows an UltraLite application to choose from a set of synchronization scripts.

Copyright © 2007, iAnywhere Solutions, Inc.

ul_synch_info_w2 struct

Syntax

public **ul_synch_info_w2**

Remarks

The wide character structure used to describe synchronization.

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). See ul_synch_info_a struct.

Members

All members of ul_synch_info_w2, including all inherited members.

- "auth_parms variable" on page 187
- "auth_status variable" on page 187
- "auth_value variable" on page 187
- "checkpoint_store variable" on page 187
- "disable_concurrency variable" on page 187
- "download_only variable" on page 187
- "ignored_rows variable" on page 188
- "init_verify variable" on page 188
- "keep_partial_download variable" on page 188
- "new_password variable" on page 188
- "num_auth_parms variable" on page 188
- "observer variable" on page 189
- "partial_download_retained variable" on page 189
- "password variable" on page 189
- "ping variable" on page 189
- "publication variable" on page 189
- "resume_partial_download variable" on page 190
- "send_column_names variable" on page 190
- "send_download_ack variable" on page 190
- "stream variable" on page 190
- "stream_error variable" on page 190
- "stream_parms variable" on page 190
- "table_order variable" on page 191
- "upload_ok variable" on page 191
- "upload_only variable" on page 191
- "user_data variable" on page 191
- "user_name variable" on page 191
- "version variable" on page 192

 Copyright © 2007, iAnywhere Solutions, Inc.

auth_parms variable

Synopsis

ul_wchar ** **ul_synch_info_w2::auth_parms**

Remarks

An array of authentication parameters in MobiLink events.

auth_status variable

Synopsis

ul_auth_status **ul_synch_info_w2::auth_status**

Remarks

The status of MobiLink user authentication. The MobiLink server provides this information to the client.

auth_value variable

Synopsis

ul_s_long **ul_synch_info_w2::auth_value**

Remarks

The MobiLink server provides this information to the client to determine the authentication status.

checkpoint_store variable

Synopsis

ul_bool **ul_synch_info_w2::checkpoint_store**

Remarks

Adds additional checkpoints of the database during synchronization to limit database growth during the synchronization process.

disable_concurrency variable

Synopsis

ul_bool **ul_synch_info_w2::disable_concurrency**

Remarks

Disallow database access from other threads during synchronization.

download_only variable

Synopsis

ul_bool **ul_synch_info_w2::download_only**

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Do not upload any changes from the UltraLite database during the current synchronization.

ignored_rows variable

Synopsis

ul_bool **ul_synch_info_w2::ignored_rows**

Remarks

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 variable

Synopsis

ul_synch_info_w2 * **ul_synch_info_w2::init_verify**

Remarks

Initialize verification.

keep_partial_download variable

Synopsis

ul_bool **ul_synch_info_w2::keep_partial_download**

Remarks

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 variable

Synopsis

ul_wchar * **ul_synch_info_w2::new_password**

Remarks

A string specifying a new MobiLink password associated with the user name. This parameter is optional.

num_auth_parms variable

Synopsis

ul_byte **ul_synch_info_w2::num_auth_parms**

Remarks

The number of authentication parameters being passed to authentication parameters in MobiLink events.

 Copyright © 2007, iAnywhere Solutions, Inc.

observer variable

Synopsis

ul_synch_observer_fn **ul_synch_info_w2::observer**

Remarks

A pointer to a callback function or event handler that monitors synchronization. This parameter is optional.

partial_download_retained variable

Synopsis

ul_bool **ul_synch_info_w2::partial_download_retained**

Remarks

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 variable

Synopsis

ul_wchar * **ul_synch_info_w2::password**

Remarks

A string specifying the existing MobiLink password associated with the user name. This parameter is optional.

ping variable

Synopsis

ul_bool **ul_synch_info_w2::ping**

Remarks

Confirm communications between the UltraLite client and the MobiLink server. When this parameter is set to true, no synchronization takes place.

publication variable

Synopsis

ul_publication_mask **ul_synch_info_w2::publication**

Remarks

An OR'd list of publications for which the last download time is retrieved. The set is supplied as a mask. This parameter is optional.

Copyright © 2007, iAnywhere Solutions, Inc.

resume_partial_download variable

Synopsis

ul_bool **ul_synch_info_w2::resume_partial_download**

Remarks

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 variable

Synopsis

ul_bool **ul_synch_info_w2::send_column_names**

Remarks

Instructs the application that column names should be sent to the Mobilikn server in the upload.

send_download_ack variable

Synopsis

ul_bool **ul_synch_info_w2::send_download_ack**

Remarks

Instructs the MobiLink server whether or not the client will provide a download acknowledgment.

stream variable

Synopsis

const char * **ul_synch_info_w2::stream**

Remarks

The MobiLink network protocol to use for synchronization.

stream_error variable

Synopsis

ul_stream_error **ul_synch_info_w2::stream_error**

Remarks

The structure to hold communications error reporting information.

stream_parms variable

Synopsis

ul_wchar * **ul_synch_info_w2::stream_parms**

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

The options to configure the network protocol you selected.

table_order variable

Synopsis

ul_wchar * **ul_synch_info_w2::table_order**

Remarks

The table order required for priority synchronization, if the UltraLite default table ordering is not suitable for your deployment. This parameter is optional.

upload_ok variable

Synopsis

ul_bool **ul_synch_info_w2::upload_ok**

Remarks

The status of data uploaded to the MobiLink server. This field reports true if upload succeeded.

upload_only variable

Synopsis

ul_bool **ul_synch_info_w2::upload_only**

Remarks

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 variable

Synopsis

ul_void * **ul_synch_info_w2::user_data**

Remarks

Make application-specific information available to the synchronization observer. This parameter is optional.

user_name variable

Synopsis

ul_wchar * **ul_synch_info_w2::user_name**

Remarks

A string that the MobiLink server uses to identify a unique MobiLink user.

Copyright © 2007, iAnywhere Solutions, Inc.

version variable

Synopsis

ul_wchar * **ul_synch_info_w2::version**

Remarks

The version string allows an UltraLite application to choose from a set of synchronization scripts.

 Copyright © 2007, iAnywhere Solutions, Inc.

ul_synch_result struct

Syntax

public **ul_synch_result**

Remarks

A structure to hold the synchronization result, so that appropriate action can be taken in the application.

Members

All members of ul_synch_result, including all inherited members.

- "auth_status variable" on page 193
- "auth_value variable" on page 193
- "ignored_rows variable" on page 193
- "partial_download_retained variable" on page 194
- "sql_code variable" on page 194
- "status variable" on page 194
- "stream_error variable" on page 194
- "timestamp variable" on page 194
- "upload_ok variable" on page 194

auth_status variable

Synopsis

ul_auth_status **ul_synch_result::auth_status**

Remarks

The synchronization authentication status.

auth_value variable

Synopsis

ul_s_long **ul_synch_result::auth_value**

Remarks

The value used by the MobiLink server to determine the auth_status result.

ignored_rows variable

Synopsis

ul_bool **ul_synch_result::ignored_rows**

Remarks

Set to true if uploaded rows were ignored; false otherwise.

partial_download_retained variable

Synopsis

ul_bool **ul_synch_result::partial_download_retained**

Remarks

The value that tells you that a partial download was retained. See keep_partial_download.

sql_code variable

Synopsis

an_sql_code **ul_synch_result::sql_code**

Remarks

The SQL code from the last synchronization.

status variable

Synopsis

ul_synch_status **ul_synch_result::status**

Remarks

The status information used by the observer function. See observer.

stream_error variable

Synopsis

ul_stream_error **ul_synch_result::stream_error**

Remarks

The communication stream error information.

timestamp variable

Synopsis

SQLDATETIME **ul_synch_result::timestamp**

Remarks

The time and date of the last synchronization.

upload_ok variable

Synopsis

ul_bool **ul_synch_result::upload_ok**

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Set to true if the upload was successful; false otherwise.

ul_synch_stats struct

Syntax

public **ul_synch_stats**

Remarks

Reports the statistics of the synchronization stream.

Members

All members of ul_synch_stats, including all inherited members.

- "bytes variable" on page 196
- "deletes variable" on page 196
- "inserts variable" on page 196
- "padding variable" on page 197
- "updates variable" on page 197

bytes variable

Synopsis

ul_u_long **ul_synch_stats::bytes**

Remarks

The number of bytes currently sent.

deletes variable

Synopsis

ul_u_short **ul_synch_stats::deletes**

Remarks

The number of deleted rows current sent.

inserts variable

Synopsis

ul_u_short **ul_synch_stats::inserts**

Remarks

The number of rows currently inserted.

Copyright © 2007, iAnywhere Solutions, Inc.

padding variable

Synopsis

ul_u_short **ul_synch_stats::padding**

Remarks

The manual structure alignment.

updates variable

Synopsis

ul_u_short **ul_synch_stats::updates**

Remarks

The number of updated rows currently sent.

ul_synch_status struct

Syntax

public **ul_synch_status**

Remarks

Returns synchronization progress monitoring data.

Members

All members of ul_synch_status, including all inherited members.

- "flags variable" on page 198
- "info variable" on page 198
- "received variable" on page 198
- "sent variable" on page 199
- "state variable" on page 199
- "stop variable" on page 199
- "tableCount variable" on page 199
- "tableIndex variable" on page 199

flags variable

Synopsis

ul_u_short **ul_synch_status::flags**

Remarks

Returns the current synchronization flags indicating additional information relating to the current state.

info variable

Synopsis

ul_synch_info_a * **ul_synch_status::info**

Remarks

A pointer to the ul_synch_info_a struct structure. See ul_synch_info_a struct.

received variable

Synopsis

ul_synch_stats **ul_synch_status::received**

Remarks

Returns download statistics.

 Copyright © 2007, iAnywhere Solutions, Inc.

sent variable

Synopsis

ul_synch_stats **ul_synch_status::sent**

Remarks

Returns upload statistics.

state variable

Synopsis

ul_synch_state **ul_synch_status::state**

Remarks

One of the many supported states. See ul_synch_state.

stop variable

Synopsis

ul_bool **ul_synch_status::stop**

Remarks

A boolean that cancel synchronization. A value of true means that synchronization is canceled.

tableCount variable

Synopsis

ul_u_short **ul_synch_status::tableCount**

Remarks

Returns the number of tables being synchronized. For each table there is a sending and receiving phase, so this number may be more than the number of tables being synchronized.

tableIndex variable

Synopsis

ul_u_short **ul_synch_status::tableIndex**

Remarks

The current table which is being uploaded or downloaded, starting at 0. This number may skip values when not all tables are being synchronized.

ULSqlca class

Syntax

public **ULSqlca**

Base classes

- "ULSqlcaBase class" on page 202

Remarks

The ULSqlcaBase class contains a SQLCA structure, so an external one is not required.

This class is used in most C++ component applications. You must initialize the SQLCA before you call any other functions. Each thread requires its own SQLCA.

Members

All members of ULSqlca, including all inherited members.

- "Finalize function" on page 202
- "GetCA function" on page 203
- "GetParameter function" on page 203
- "GetParameter function" on page 203
- "GetParameterCount function" on page 204
- "GetSQLCode function" on page 204
- "GetSQLCount function" on page 205
- "GetSQLErrorOffset function" on page 205
- "Initialize function" on page 205
- "LastCodeOK function" on page 206
- "LastFetchOK function" on page 206
- "ULSqlca function" on page 200
- "~ULSqlca function" on page 200

ULSqlca function

Synopsis

ULSqlca::ULSqlca()

Remarks

This function is the SQLCA constructor.

~ULSqlca function

Synopsis

ULSqlca::~ULSqlca()

 Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

This function is the SQLCA destructor.

ULSqlcaBase class

Syntax

public **ULSqlcaBase**

Derived classes

- ♦ "ULSqlca class" on page 200
- ♦ "ULSqlcaWrap class" on page 207

Remarks

Defines the communication area between the interface library and the application.

Use a subclass of this class (which is typically the ULSqlca class) to create your communication area. This API always requires an underlying SQLCA object. You must initialize the SQLCA before you call any other functions. Each thread requires its own SQLCA.

Members

All members of ULSqlcaBase, including all inherited members.

- ♦ "Finalize function" on page 202
- ♦ "GetCA function" on page 203
- ♦ "GetParameter function" on page 203
- ♦ "GetParameter function" on page 203
- ♦ "GetParameterCount function" on page 204
- ♦ "GetSQLCode function" on page 204
- ♦ "GetSQLCount function" on page 205
- ♦ "GetSQLErrorOffset function" on page 205
- ♦ "Initialize function" on page 205
- ♦ "LastCodeOK function" on page 206
- ♦ "LastFetchOK function" on page 206

Finalize function

Synopsis

void **ULSqlcaBase::Finalize()**

Remarks

Finalizes this SQLCA.

Until you initialize the communication area again, you cannot use it.

Copyright © 2007, iAnywhere Solutions, Inc.

GetCA function

Synopsis

SQLCA * **ULSqlcaBase::GetCA()**

Remarks

Gets the SQLCA structure for direct access to additional fields.

Returns

- A raw SQLCA structure.

GetParameter function

Synopsis

size_t **ULSqlcaBase::GetParameter(**
  ul_u_long *parm_num*,
  char * *buffer*,
  size_t *size*
)

Parameters

- **parm_num** A 1-based parameter number.
- **buffer** The buffer to receive parameter string.
- **size** The size, in chars, of the buffer.

Remarks

Gets the error parameter string.

The output parameter string is always null-terminated, even if the buffer is too small and the parameter is truncated. The parameter number is 1-based.

Returns

- If the function succeeds, the return value is the buffer size required to hold the entire parameter string.
- If the function fails, the return value is zero. The function fails if an invalid (out of range) parameter number is given.

GetParameter function

Synopsis

size_t **ULSqlcaBase::GetParameter(**
  ul_u_long *parm_num*,
  ul_wchar * *buffer*,

```
  size_t size
)
```

Parameters

- **parm_num** A 1-based parameter number.
- **buffer** The buffer to receive parameter string.
- **size** The size, in ul_wchars, of the buffer.

Remarks

Gets the error parameter string.

The output parameter string is always null-terminated, even if the buffer is too small and the parameter is truncated. The parameter number is 1-based.

Returns

- If the function succeeds, the return value is the buffer size required to hold the entire parameter string.
- If the function fails, the return value is zero. The function fails if an invalid (out of range) parameter number is given.

GetParameterCount function

Synopsis

ul_u_long **ULSqlcaBase::GetParameterCount()**

Remarks

Gets the error parameter count for last operation.

Returns

- The number of parameters for the current error.

GetSQLCode function

Synopsis

an_sql_code **ULSqlcaBase::GetSQLCode()**

Remarks

Gets the error code (SQLCODE) for last operation.

Returns

- The sqlcode value.

 Copyright © 2007, iAnywhere Solutions, Inc.

GetSQLCount function

Synopsis

an_sql_code **ULSqlcaBase::GetSQLCount()**

Remarks

Gets the sql count variable (SQLCOUNT) for the last operation.

Returns

The number of rows affected by an INSERT, DELETE, or UPDATE operation. 0 if none are affected.

GetSQLErrorOffset function

Synopsis

ul_s_long **ULSqlcaBase::GetSQLErrorOffset()**

Remarks

Gets the error offset in dynamic SQL statement.

Returns

- When applicable, the return value is the offset into the associated dynamic SQL statement (which is then passed to the PrepareStatement function) that corresponds to the current error.
- When not applicable, the return value is -1.

Initialize function

Synopsis

bool **ULSqlcaBase::Initialize()**

Remarks

Initializes this SQLCA.

You must initialize the SQLCA before any other operations occur.

Returns

- True if the SQLCA was initialized.
- False if the SQLCA failed to initialize. This method can fail if basic interface library initialization fails. Library failure occurs if system resources are depleted.

Copyright © 2007, iAnywhere Solutions, Inc.

LastCodeOK function

Synopsis

bool **ULSqlcaBase::LastCodeOK()**

Remarks

Tests the error code for the last operation.

Returns

- TRUE if the sqlcode is SQLE_NOERROR or a warning.
- FALSE if sqlcode indicates an error.

LastFetchOK function

Synopsis

bool **ULSqlcaBase::LastFetchOK()**

Remarks

Tests the error code for last fetch operation.

Use this function only immediately after performing a fetch operation.

Returns

- TRUE if the sqlcode indicates that a row was fetched successfully by the last operation.
- FALSE if the sqlcode indicates the row was not fetched.

 Copyright © 2007, iAnywhere Solutions, Inc.

ULSqlcaWrap class

Syntax

public **ULSqlcaWrap**

Base classes

- “ULSqlcaBase class” on page 202

Remarks

The ULSqlcaBase class which attaches to an existing SQLCA object.

This can be used with a previously-initialized SQLCA object (in which case, you would not call the Initialize function again). You must initialized the SQLCA before you call any other functions. Each thread requires its own SQLCA.

Members

All members of ULSqlcaWrap, including all inherited members.

- “Finalize function” on page 202
- “GetCA function” on page 203
- “GetParameter function” on page 203
- “GetParameter function” on page 203
- “GetParameterCount function” on page 204
- “GetSQLCode function” on page 204
- “GetSQLCount function” on page 205
- “GetSQLErrorOffset function” on page 205
- “Initialize function” on page 205
- “LastCodeOK function” on page 206
- “LastFetchOK function” on page 206
- “ULSqlcaWrap function” on page 207
- “~ULSqlcaWrap function” on page 208

ULSqlcaWrap function

Synopsis

```
ULSqlcaWrap::ULSqlcaWrap(
 SQLCA * sqlca
)
```

Parameters

- **sqlca** The SQLCA object to use.

Remarks

The constructor.

You can initialize the given SQLCA object before creating this object. In this case, do not call Initialize function again.

~ULSqlcaWrap function

Synopsis

ULSqlcaWrap::~ULSqlcaWrap()

Remarks

The destructor.

Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_Connection class

Syntax

public **UltraLite_Connection**

Base classes

- "UltraLite_SQLObject_iface class" on page 258
- "UltraLite_Connection_iface class" on page 210

Remarks

Represents a connection to an UltraLite database.

UltraLite_Connection_iface class

Syntax

public **UltraLite_Connection_iface**

Derived classes

♦ "UltraLite_Connection class" on page 209

Remarks

The connection interface.

Members

All members of UltraLite_Connection_iface, including all inherited members.

- "ChangeEncryptionKey function" on page 211
- "Checkpoint function" on page 211
- "Commit function" on page 211
- "CountUploadRows function" on page 211
- "GetConnectionNum function" on page 212
- "GetDatabaseID function" on page 212
- "GetDatabaseProperty function" on page 212
- "GetLastDownloadTime function" on page 212
- "GetLastIdentity function" on page 213
- "GetNewUUID function" on page 213
- "GetPublicationMask function" on page 213
- "GetSchema function" on page 214
- "GetSqlca function" on page 214
- "GetSuspend function" on page 214
- "GetSynchResult function" on page 214
- "GetUtilityULValue function" on page 215
- "GlobalAutoincUsage function" on page 215
- "GrantConnectTo function" on page 215
- "InitSynchInfo function" on page 215
- "InitSynchInfo function" on page 216
- "OpenTable function" on page 216
- "OpenTableWithIndex function" on page 216
- "PrepareStatement function" on page 217
- "ResetLastDownloadTime function" on page 217
- "RevokeConnectFrom function" on page 217
- "Rollback function" on page 218
- "RollbackPartialDownload function" on page 218
- "SetDatabaseID function" on page 218
- "SetDatabaseOption function" on page 218
- "SetSuspend function" on page 219
- "SetSynchInfo function" on page 219
- "SetSynchInfo function" on page 219

 Copyright © 2007, iAnywhere Solutions, Inc.

- "Shutdown function" on page 220
- "StartSynchronizationDelete function" on page 220
- "StopSynchronizationDelete function" on page 220
- "StrToUUID function" on page 220
- "Synchronize function" on page 221
- "Synchronize function" on page 221
- "Synchronize function" on page 221
- "UUIDToStr function" on page 222
- "UUIDToStr function" on page 222

ChangeEncryptionKey function

Synopsis

bool **UltraLite_Connection_iface::ChangeEncryptionKey(**
  const ULValue & *new_key*
)

Parameters

- **new_key** The new encryption key value for the database.

Remarks

Changes the encryption key.

Checkpoint function

Synopsis

bool **UltraLite_Connection_iface::Checkpoint()**

Remarks

Checkpoints the database.

Commit function

Synopsis

bool **UltraLite_Connection_iface::Commit()**

Remarks

Commits the current transaction.

CountUploadRows function

Synopsis

ul_u_long **UltraLite_Connection_iface::CountUploadRows(**
  ul_publication_mask *mask*,
  ul_u_long *threshold*
)

Parameters

- **mask** The set of publications to consider.
- **threshold** The limit on the number of rows to count.

Remarks

Determines the number of rows that need to be uploaded.

GetConnectionNum function

Synopsis

ul_connection_num **UltraLite_Connection_iface::GetConnectionNum()**

Remarks

Gets the connection number.

GetDatabaseID function

Synopsis

ul_u_long **UltraLite_Connection_iface::GetDatabaseID()**

Remarks

Gets the database ID used for global autoincrement columns.

GetDatabaseProperty function

Synopsis

ULValue **UltraLite_Connection_iface::GetDatabaseProperty(**
  ul_database_property_id *id*
)

Parameters

- **id** The ID of the property being requested.

Remarks

Gets the Database Property.

Returns

- The value of the requested property.

GetLastDownloadTime function

Synopsis

bool **UltraLite_Connection_iface::GetLastDownloadTime(**
  ul_publication_mask *mask*,

 Copyright © 2007, iAnywhere Solutions, Inc.

DECL_DATETIME * *value*
)

Parameters

- **mask** The publication mask.
- **value** The last download time.

Remarks

Gets the time of the last download.

GetLastIdentity function

Synopsis

ul_u_big **UltraLite_Connection_iface::GetLastIdentity()**

Remarks

Gets the @@identity value.

GetNewUUID function

Synopsis

bool **UltraLite_Connection_iface::GetNewUUID(**
p_ul_binary *uuid*
)

Parameters

- **uuid** The new UUID value.

Remarks

Creates a new UUID.

GetPublicationMask function

Synopsis

ul_publication_mask **UltraLite_Connection_iface::GetPublicationMask(**
const ULValue & *pub_id*
)

Parameters

- **pub_id** The publication name or ordinal.

Remarks

Gets the publication mask for a given publication ID.

A publication mask is an array of OR'd publications. 0 is returned if the publication is not found.

Copyright © 2007, iAnywhere Solutions, Inc.

GetSchema function

Synopsis

UltraLite_DatabaseSchema * **UltraLite_Connection_iface::GetSchema()**

Remarks

Gets the database schema.

GetSqlca function

Synopsis

ULSqlcaBase const & **UltraLite_Connection_iface::GetSqlca()**

Remarks

Gets the communication area associated with this connection.

GetSuspend function

Synopsis

bool **UltraLite_Connection_iface::GetSuspend()**

Remarks

Gets the Suspend property.

Returns

- True if this connection is suspended.
- False if the connection is not suspended.

GetSynchResult function

Synopsis

bool **UltraLite_Connection_iface::GetSynchResult(**
 ul_synch_result * *synch_result*
)

Parameters

- **synch_result** A pointer to the ul_synch_result struct structure that holds the synchronization results.

Remarks

Gets the result of the last synchronization.

 Copyright © 2007, iAnywhere Solutions, Inc.

GetUtilityULValue function

Synopsis

ULValue **UltraLite_Connection_iface::GetUtilityULValue()**

Remarks

Gets a new ULValue class.

A ULValue class object must be bound to a connection in order for many of its methods to succeed.

GlobalAutoincUsage function

Synopsis

ul_u_short **UltraLite_Connection_iface::GlobalAutoincUsage()**

Remarks

Gets the percent of the global autoincrement values used by the counter.

GrantConnectTo function

Synopsis

bool **UltraLite_Connection_iface::GrantConnectTo(**
 const ULValue & *uid*,
 const ULValue & *pwd*
)

Parameters

- **uid** The user ID for which access to connect is granted.
- **pwd** The password for the autorized user ID.

Remarks

To create a new user, specify both a new user ID and a password.

To change a password, specify an existing user ID, but set a new password for that user.

InitSynchInfo function

Synopsis

void **UltraLite_Connection_iface::InitSynchInfo(**
 ul_synch_info_a * *info*
)

Parameters

- **info** A pointer to the ul_synch_info structure that holds the synchronization parameters.

Remarks

Initializes the synchronization information structure.

InitSynchInfo function

Synopsis

```
void UltraLite_Connection_iface::InitSynchInfo(
  ul_synch_info_w2 * info
)
```

Parameters

- **info** A pointer to the ul_synch_info structure that holds the synchronization parameters.

Remarks

Initializes the synchronization information structure.

OpenTable function

Synopsis

```
UltraLite_Table * UltraLite_Connection_iface::OpenTable(
  const ULValue & table_id,
  const ULValue & persistent_name
)
```

Parameters

- **table_id** The table name or ordinal.
- **persistent_name** The instance name used for suspending.

Remarks

Opens a table.

When the application first opens a table, the cursor position is set to BeforeFirst().

OpenTableWithIndex function

Synopsis

```
UltraLite_Table * UltraLite_Connection_iface::OpenTableWithIndex(
  const ULValue & table_id,
  const ULValue & index_id,
  const ULValue & persistent_name
)
```

Parameters

- **table_id** The table name or ordinal.
- **index_id** The index name or ordinal.

 Copyright © 2007, iAnywhere Solutions, Inc.

- **persistent_name** The instance name used for suspending.

Remarks

Opens a table, with the named index to order the rows.

When the application first opens a table, the cursor position is set to BeforeFirst().

PrepareStatement function

Synopsis

UltraLite_PreparedStatement * **UltraLite_Connection_iface::PrepareStatement(**
const ULValue & *sql*,
const ULValue & *persistent_name*
)

Parameters

- **sql** The SQL statement as a string.
- **persistent_name** The instance name used for suspending.

Remarks

Prepares a SQL statement.

ResetLastDownloadTime function

Synopsis

bool **UltraLite_Connection_iface::ResetLastDownloadTime(**
ul_publication_mask *mask*
)

Parameters

- **mask** The set of publications to reset.

Remarks

Resets the time of the last download.

RevokeConnectFrom function

Synopsis

bool **UltraLite_Connection_iface::RevokeConnectFrom(**
const ULValue & *uid*
)

Parameters

- **uid** The user ID for whom the authority to connect is being revoked.

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Deletes an existing user.

Rollback function

Synopsis

bool **UltraLite_Connection_iface::Rollback()**

Remarks

Rolls back the current transaction.

RollbackPartialDownload function

Synopsis

bool **UltraLite_Connection_iface::RollbackPartialDownload()**

Remarks

Rolls back a partial download.

SetDatabaseID function

Synopsis

bool **UltraLite_Connection_iface::SetDatabaseID(**
  ul_u_long *value*
)

Parameters

- **value** The database ID, which determines the starting value for global autoincrement columns.

Remarks

Sets the database ID used for global autoincrement columns.

SetDatabaseOption function

Synopsis

bool **UltraLite_Connection_iface::SetDatabaseOption(**
  ul_database_option_id *id*,
  const ULValue & *value*
)

Parameters

- **id** The ID of the option being set.
- **value** The new value of the option.

Remarks

Sets the specified Database Option.

 Copyright © 2007, iAnywhere Solutions, Inc.

SetSuspend function

Synopsis

void **UltraLite_Connection_iface::SetSuspend(**
  bool *suspend*
)

Parameters

- **suspend** Set to true to suspend the connection, so that its state can be restored when the database is reopened.

Remarks

Sets the Suspend property.

The connection name (or lack thereof) identifies suspended connetions.

Returns

- True if the connection is suspended.

SetSynchInfo function

Synopsis

bool **UltraLite_Connection_iface::SetSynchInfo(**
  ul_synch_info_a * *info*
)

Parameters

- **info** A pointer to the ul_synch_info structure that holds the synchronization parameters.

Remarks

Attaches a ul_synch_info struct to the current database for use on a subsequent synchronization.

Synchronization information is saved in the database. This replaces any previously saved synchronization information by performing an auto-commit. If you supply a null pointer, you clear any currently saved synchronization information.

SetSynchInfo function

Synopsis

bool **UltraLite_Connection_iface::SetSynchInfo(**
  ul_synch_info_w2 * *info*
)

Parameters

- **info** A pointer to the ul_synch_info structure that holds the synchronization parameters.

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Attaches a ul_synch_info struct to the current database for use on a subsequent synchronization.

Shutdown function

Synopsis

void **UltraLite_Connection_iface::Shutdown()**

Remarks

Destroys this connection and any remaining associated objects.

If you do not set the connection to suspend, this connection is rolled back.

StartSynchronizationDelete function

Synopsis

bool **UltraLite_Connection_iface::StartSynchronizationDelete()**

Remarks

Sets START SYNCHRONIZATION DELETE for this connection.

StopSynchronizationDelete function

Synopsis

bool **UltraLite_Connection_iface::StopSynchronizationDelete()**

Remarks

Sets STOP SYNCHRONIZATION DELETE for this connection.

StrToUUID function

Synopsis

bool **UltraLite_Connection_iface::StrToUUID(**
  p_ul_binary *dst*,
  size_t *len*,
  const ULValue & *src*
)

Parameters

- **dst** The UUID value being returned.
- **len** The length of the ul_binary array.
- **src** A string holding the UUID value to be converted.

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Converts a string to a UUID.

Synchronize function

Synopsis

bool **UltraLite_Connection_iface::Synchronize(**
 ul_synch_info_a * *info*
)

Parameters

- **info** A pointer to the ul_synch_info structure that holds the synchronization parameters.

Remarks

Synchronizes the database.

Example:

```
 info;ul_synch_info
conn->InitSynchInfo( &info );
info. = UL_TEXT( user_name"user_name" );
info. = UL_TEXT( version"test" );
conn->Synchronize( &info );
```

Synchronize function

Synopsis

bool **UltraLite_Connection_iface::Synchronize(**
 ul_synch_info_w2 * *info*
)

Parameters

- **info** A pointer to the ul_synch_info structure that holds the synchronization parameters.

Remarks

Synchronizes the database.

See Synchronize function.

Synchronize function

Synopsis

bool **UltraLite_Connection_iface::Synchronize()**

Remarks

Synchronizes the database with the synch info previously stored in the database by SetSynchInfo function.

Copyright © 2007, iAnywhere Solutions, Inc.

Example:

```
 info;ul_synch_info
conn->InitSynchInfo( &info );
info. = UL_TEXT( user_name"user_name" );
info. = UL_TEXT( version"test" );
conn.SetSynchInfo( &info );
// ...
conn->Synchronize();
```

UUIDToStr function

Synopsis

bool **UltraLite_Connection_iface::UUIDToStr(**
  char * *dst*,
  size_t *len*,
  p_ul_binary *src*
)

Parameters

- **dst** The string being returned.
- **len** The length of the ul_binary array.
- **src** The UUID value to be converted to a string.

Remarks

Converts a UUID to an ANSI string.

UUIDToStr function

Synopsis

bool **UltraLite_Connection_iface::UUIDToStr(**
  ul_wchar * *dst*,
  size_t *len*,
  p_ul_binary *src*
)

Parameters

- **dst** The Unicode string being returned.
- **len** The length of the ul_binary array.
- **src** The UUID value to be converted to a string.

Remarks

Converts a UUID to a Unicode string.

 Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_Cursor_iface class

Syntax

public **UltraLite_Cursor_iface**

Derived classes

- "UltraLite_ResultSet class" on page 250
- "UltraLite_Table class" on page 265

Remarks

Represents a bi-directional cursor in an UltraLite database.

Cursors are sets of rows from either a table or the result set from a query.

Members

All members of UltraLite_Cursor_iface, including all inherited members.

- "AfterLast function" on page 223
- "BeforeFirst function" on page 224
- "Delete function" on page 224
- "First function" on page 224
- "Get function" on page 224
- "GetRowCount function" on page 224
- "GetState function" on page 225
- "GetStreamReader function" on page 225
- "GetStreamWriter function" on page 225
- "GetSuspend function" on page 226
- "IsNull function" on page 226
- "Last function" on page 226
- "Next function" on page 226
- "Previous function" on page 227
- "Relative function" on page 227
- "Set function" on page 227
- "SetDefault function" on page 228
- "SetNull function" on page 228
- "SetSuspend function" on page 228
- "Update function" on page 229
- "UpdateBegin function" on page 229

AfterLast function

Synopsis

bool **UltraLite_Cursor_iface::AfterLast()**

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Moves the cursor after the last row.

BeforeFirst function

Synopsis

bool **UltraLite_Cursor_iface::BeforeFirst()**

Remarks

Moves the cursor before the first row.

Delete function

Synopsis

bool **UltraLite_Cursor_iface::Delete()**

Remarks

Deletes the current row and moves it to the next valid row.

First function

Synopsis

bool **UltraLite_Cursor_iface::First()**

Remarks

Moves the cursor to the first row.

Get function

Synopsis

ULValue **UltraLite_Cursor_iface::Get(**
 const ULValue & *column_id*
)

Parameters

- **column_id** The name or ordinal of the column.

Remarks

Fetches a value from a column.

GetRowCount function

Synopsis

ul_u_long **UltraLite_Cursor_iface::GetRowCount()**

 Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Gets the number of rows in the table.

Calling this method is equivalent to executing "SELECT COUNT(*) FROM table".

GetState function

Synopsis

UL_RS_STATE **UltraLite_Cursor_iface::GetState()**

Remarks

Gets the internal state of the cursor.

See the UL_RS_STATE enumeration in ulglobal.h

GetStreamReader function

Synopsis

UltraLite_StreamReader * **UltraLite_Cursor_iface::GetStreamReader(**
const ULValue & *id*
)

Parameters

- **id** A column identifier, which may be either a 1-based ordinal number or a column name.

Remarks

Gets a stream reader object for reading string or binary column data in chunks.

GetStreamWriter function

Synopsis

UltraLite_StreamWriter * **UltraLite_Cursor_iface::GetStreamWriter(**
const ULValue & *column_id*
)

Parameters

- **column_id** A column identifier, which may be either a 1-based ordinal number or a column name.

Remarks

Gets a stream writer for streaming string/binary data into a column.

GetSuspend function

Synopsis

bool **UltraLite_Cursor_iface::GetSuspend()**

Remarks

Gets the value of the Suspend property.

Returns

- True if this cursor is suspended.
- False if it is not.

IsNull function

Synopsis

bool **UltraLite_Cursor_iface::IsNull(**
const ULValue & *column_id*
)

Parameters

- **column_id** The name or ordinal of the column.

Remarks

Checks if a column is NULL.

Last function

Synopsis

bool **UltraLite_Cursor_iface::Last()**

Remarks

Moves the cursor to the last row.

Next function

Synopsis

bool **UltraLite_Cursor_iface::Next()**

Remarks

Moves the cursor forward one row.

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

 Copyright © 2007, iAnywhere Solutions, Inc.

evaluating the SELECT expressions. In this case, errors are also returned when retrieving the column values.

- False, if it fails to move forward. For example, there may not be a next row. In this case, the resulting cursor position is AfterLast function.

Previous function

Synopsis

bool **UltraLite_Cursor_iface::Previous()**

Remarks

Moves the cursor back one row.

On failure, the resulting cursor position is BeforeFirst function.

Relative function

Synopsis

bool **UltraLite_Cursor_iface::Relative(**
  ul_fetch_offset *offset*
)

Parameters

- **offset** The number of rows to move.

Remarks

Moves the cursor by offset rows from the current cursor position.

Set function

Synopsis

bool **UltraLite_Cursor_iface::Set(**
  const ULValue & *column_id*,
  const ULValue & *value*
)

Parameters

- **column_id** A 1-based ordinal number identifying the column.
- **value** The value to which the column is set.

Remarks

Sets a column value.

SetDefault function

Synopsis

bool **UltraLite_Cursor_iface::SetDefault(**
  const ULValue & *column_id*
)

Parameters

- **column_id** A 1-based ordinal number identifying the column.

Remarks

Sets column(s) to their default value.

SetNull function

Synopsis

bool **UltraLite_Cursor_iface::SetNull(**
  const ULValue & *column_id*
)

Parameters

- **column_id** A 1-based ordinal number identifying the column.

Remarks

Sets a column to null.

SetSuspend function

Synopsis

void **UltraLite_Cursor_iface::SetSuspend(**
  bool *suspend*
)

Parameters

- **suspend** True suspends the connection. This allows you to restore the database's state when the database is reopened.

Remarks

Sets the value of the Suspend property.

Use the persistent name parameter when opening the associated object to identify suspended cursors. If you do not supply a persistent name parameter for this cursor, you cannot suspend it.

Returns

- True, if the cursor is suspended and restored when the application reopens the database.
- False, if the cursor is not suspended.

 Copyright © 2007, iAnywhere Solutions, Inc.

Update function

Synopsis

bool **UltraLite_Cursor_iface::Update()**

Remarks

Updates the current row.

The table must be in update mode for this operation to succeed. Use UpdateBegin function to switch to update mode;

UpdateBegin function

Synopsis

bool **UltraLite_Cursor_iface::UpdateBegin()**

Remarks

Selects update mode for setting columns.

Columns in the primary key may not be modified when in update mode.

Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_DatabaseManager class

Syntax

public **UltraLite_DatabaseManager**

Base classes

- "UltraLite_DatabaseManager_iface class" on page 231

Remarks

Manages synchronization listeners and allows you to drop (delete) UltraLite databases.

 Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_DatabaseManager_iface class

Syntax

public **UltraLite_DatabaseManager_iface**

Derived classes

- "UltraLite_DatabaseManager class" on page 230

Remarks

Manages connections and databases.

Creating a database and establishing a connection to it is a necessary first step in using UltraLite. You should ensure that you are connected properly before attempting any DML with the database.

Members

All members of UltraLite_DatabaseManager_iface, including all inherited members.

- "CreateDatabase function" on page 231
- "DropDatabase function" on page 232
- "OpenConnection function" on page 232
- "Shutdown function" on page 233

CreateDatabase function

Synopsis

```
bool UltraLite_DatabaseManager_iface::CreateDatabase(
  ULSqlcaBase & sqlca,
  ULValue const & access_parms,
  void const * coll,
  ULValue const & create_parms,
  void * reserved
)
```

Parameters

- **sqlca** The initialized sqlca.
- **access_parms** Connection parameters used to access database
- **coll** Collation sequence
- **create_parms** Parameters used to create the database
- **reserved** Reserved (not used currently)

Remarks

Creates a new database.

DropDatabase function

Synopsis

bool **UltraLite_DatabaseManager_iface::DropDatabase(**
ULSqlcaBase & *sqlca*,
const ULValue & *parms_string*
)

Parameters

- **sqlca** The initialized sqlca.
- **parms_string** The database identification parameters.

Remarks

Erases an existing database that is already stopped.

You cannot erase a running database.

OpenConnection function

Synopsis

UltraLite_Connection * **UltraLite_DatabaseManager_iface::OpenConnection(**
ULSqlcaBase & *sqlca*,
ULValue const & *parms_string*
)

Parameters

- **sqlca** The initialized sqlca to associate with the new connection.
- **parms_string** The connection string.

Remarks

Opens a new connection to an existing database.

The given sqlca is associated with the new connection.

- SQLE_CONNECTION_ALREADY_EXISTS - A connection with the given SQLCA and name (or no name) already exists. Before connecting you must disconnect the existing connection, or specify a different connection name with the CON parameter.
- SQLE_INVALID_LOGON - You supplied an invalid user ID or an incorrect password.
- SQLE_INVALID_SQL_IDENTIFIER - An invalid identifier was supplied through the C language interface. For example, you may have supplied a NULL string for a cursor name.
- SQLE_TOO_MANY_CONNECTIONS - You exceeded the number of concurrent database connections.

To get error information, use the associated ULSqlca class object. Possible errors include:

 Copyright © 2007, iAnywhere Solutions, Inc.

Returns

- If the function succeeds, the application returns a new connection object is returned.
- If the function fails, NULL is returned.

Shutdown function

Synopsis

void **UltraLite_DatabaseManager_iface::Shutdown(**
 ULSqlcaBase & *sqlca*
)

Parameters

- **sqlca** The initialized sqlca.

Remarks

Closes all databases and releases the database manager.

Any remaining associated objects are destroyed. After calling this function, the database manager can no longer be used (nor can any other previously obtained objects).

UltraLite_DatabaseSchema class

Syntax

public **UltraLite_DatabaseSchema**

Base classes

- "UltraLite_SQLObject_iface class" on page 258
- "UltraLite_DatabaseSchema_iface class" on page 235

Remarks

Represents the schema of an UltraLite database.

 Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_DatabaseSchema_iface class

Syntax

public **UltraLite_DatabaseSchema_iface**

Derived classes

- "UltraLite_DatabaseSchema class" on page 234

Remarks

DatabaseSchema interface.

Members

All members of UltraLite_DatabaseSchema_iface, including all inherited members.

- "GetCollationName function" on page 235
- "GetPublicationCount function" on page 235
- "GetPublicationID function" on page 236
- "GetPublicationMask function" on page 236
- "GetPublicationName function" on page 236
- "GetTableCount function" on page 237
- "GetTableName function" on page 237
- "GetTableSchema function" on page 238
- "IsCaseSensitive function" on page 238

GetCollationName function

Synopsis

ULValue **UltraLite_DatabaseSchema_iface::GetCollationName()**

Remarks

Gets the name of the current collation sequence.

Returns

- A ULValue class that contains a string.

GetPublicationCount function

Synopsis

ul_publication_count **UltraLite_DatabaseSchema_iface::GetPublicationCount()**

Remarks

Gets the number of publications in the database.

Publication IDs range from 1 to GetPublicationCount function

GetPublicationID function

Synopsis

ul_u_short **UltraLite_DatabaseSchema_iface::GetPublicationID(**
const ULValue & *pub_id*
)

Parameters

- **pub_id** A 1-based ordinal number.

Remarks

Gets a 1-based id for the publication given its name.

GetPublicationMask function

Synopsis

ul_publication_mask **UltraLite_DatabaseSchema_iface::GetPublicationMask(**
const ULValue & *pub_id*
)

Parameters

- **pub_id** A 1-based ordinal number.

Remarks

Gets the publication mask for a given publication name.

A mask defines the set of publications that is created by or'ing the individual publications into a group. Publication masks are not publication IDs.

Returns

- 0 if the publication is not found.

GetPublicationName function

Synopsis

ULValue **UltraLite_DatabaseSchema_iface::GetPublicationName(**
const ULValue & *pub_id*
)

Parameters

- **pub_id** A 1-based ordinal number.

 Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Gets the name of a publication given, its 1-based index ID.

Publication masks are not publication IDs. Instead, a mask defines the set of publications that is created by or'ing the individual publications into a group.

GetTableCount function

Synopsis

ul_table_num **UltraLite_DatabaseSchema_iface::GetTableCount()**

Remarks

Returns the number of tables in the database.

Returns

- An integer that represents the number of tables.
- 0 if the connection is not open.

GetTableName function

Synopsis

ULValue **UltraLite_DatabaseSchema_iface::GetTableName(**
  ul_table_num *tableID*
)

Parameters

- **tableID** A 1-based ordinal number.

Remarks

Gets the name of a table given its 1-based table ID.

Table IDs may change during a schema upgrade. To correctly identify a table, either access it by name, or refresh any cached IDs after a schema upgrade. The ULValue class object returned is empty if the table does not exist.

Returns

- The name of the table identified by the specified table ID.

GetTableSchema function

Synopsis

UltraLite_TableSchema * **UltraLite_DatabaseSchema_iface::GetTableSchema(**
  const ULValue & *table_id*
)

Parameters

- **table_id**   A 1-based ordinal number.

Remarks

Gets a TableSchema object given a 1-based table ID or name.

Returns

- UL_NULL if the table does not exist.

IsCaseSensitive function

Synopsis

bool **UltraLite_DatabaseSchema_iface::IsCaseSensitive()**

Remarks

Gets the case sensitivity of the database.

Database case sensitivity affects how indexes on tables and result sets are sorted.

Returns

- True if the database is case sensitive.
- False otherwise.

Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_IndexSchema class

Syntax

public **UltraLite_IndexSchema**

Base classes

- "UltraLite_SQLObject_iface class" on page 258
- "UltraLite_IndexSchema_iface class" on page 240

Remarks

Represents the schema of an UltraLite table index.

UltraLite_IndexSchema_iface class

Syntax

public **UltraLite_IndexSchema_iface**

Derived classes

- "UltraLite_IndexSchema class" on page 239

Remarks

Represents an IndexSchema interface.

Members

All members of UltraLite_IndexSchema_iface, including all inherited members.

- "GetColumnCount function" on page 240
- "GetColumnName function" on page 240
- "GetID function" on page 241
- "GetName function" on page 241
- "GetReferencedIndexName function" on page 241
- "GetReferencedTableName function" on page 242
- "GetTableName function" on page 242
- "IsColumnDescending function" on page 242
- "IsForeignKey function" on page 242
- "IsForeignKeyCheckOnCommit function" on page 243
- "IsForeignKeyNullable function" on page 243
- "IsPrimaryKey function" on page 243
- "IsUniqueIndex function" on page 244
- "IsUniqueKey function" on page 244

GetColumnCount function

Synopsis

ul_column_num **UltraLite_IndexSchema_iface::GetColumnCount()**

Remarks

Gets the number of columns in the index.

GetColumnName function

Synopsis

ULValue **UltraLite_IndexSchema_iface::GetColumnName(**
  ul_column_num *col_id_in_index*
)

 Copyright © 2007, iAnywhere Solutions, Inc.

Parameters

- **col_id_in_index** The 1-based ordinal number indicating the position of the column in the index.

Remarks

Gets the name of the column given the position of the column in the index.

Returns

- An emptry ULValue class object if the column does not exist.
- SQLE_COLUMN_NOT_FOUND if the column name does not exist.

GetID function

Synopsis

ul_index_num **UltraLite_IndexSchema_iface::GetID()**

Remarks

Gets the index ID.

Returns

- The ID of the index.

GetName function

Synopsis

ULValue **UltraLite_IndexSchema_iface::GetName()**

Remarks

Gets the name of the index.

GetReferencedIndexName function

Synopsis

ULValue **UltraLite_IndexSchema_iface::GetReferencedIndexName()**

Remarks

Gets the associated primary index name.

This function is for foreign keys only.

Returns

- An empty ULValue class object if the index is not a foreign key.

Copyright © 2007, iAnywhere Solutions, Inc.

GetReferencedTableName function

Synopsis

ULValue **UltraLite_IndexSchema_iface::GetReferencedTableName()**

Remarks

Gets the associated primary table name.

This method is for foreign keys only.

Returns

- An empty ULValue class object if the index is not a foreign key.

GetTableName function

Synopsis

ULValue **UltraLite_IndexSchema_iface::GetTableName()**

Remarks

Gets the name of the table containing the index.

IsColumnDescending function

Synopsis

```
bool UltraLite_IndexSchema_iface::IsColumnDescending(
  const ULValue & column_name
)
```

Parameters

- **column_name** The column name.

Remarks

Determines if the column is in descending order.

Returns

- True if the column is in descending order.
- Sets SQLE_COLUMN_NOT_FOUND if the column name does not exist.

IsForeignKey function

Synopsis

bool **UltraLite_IndexSchema_iface::IsForeignKey()**

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Checks whether the index is a foreign key.

Columns in a foreign key may reference another table's non-null, unique index.

Returns

- True if the index is a foreign key.
- False if the index is not a foreign key.

IsForeignKeyCheckOnCommit function

Synopsis

bool **UltraLite_IndexSchema_iface::IsForeignKeyCheckOnCommit()**

Remarks

Checks whether referential integrity for the foreign key is performed on commits or on inserts and updates.

Returns

- True if this foreign key checks referential integrity on commit.
- False if this foreign key checks referential integrity on insert.

IsForeignKeyNullable function

Synopsis

bool **UltraLite_IndexSchema_iface::IsForeignKeyNullable()**

Remarks

Checks whether the foreign key is nullable.

Returns

- True if the index is a unique foreign key constraint.
- False if the foreign key is not nullable

IsPrimaryKey function

Synopsis

bool **UltraLite_IndexSchema_iface::IsPrimaryKey()**

Remarks

Checks whether the index is a primary key.

Copyright © 2007, iAnywhere Solutions, Inc.

Columns in the primary key may not be null.

Returns

- True if the index is a primary key.
- False if the index is not a primary key.

IsUniqueIndex function

Synopsis

bool **UltraLite_IndexSchema_iface::IsUniqueIndex()**

Remarks

Checks whether the index is unique.

Returns

- True if the index is unique.
- False if the index is not unique.

IsUniqueKey function

Synopsis

bool **UltraLite_IndexSchema_iface::IsUniqueKey()**

Remarks

Checks whether the index is a unique key.

Columns in a unique key may not be null.

Returns

- True if the index is a primary key or a unique constraint.
- False if the index is not either a primary key or a a unique constraint.

Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_PreparedStatement class

Syntax

public **UltraLite_PreparedStatement**

Base classes

- "UltraLite_SQLObject_iface class" on page 258
- "UltraLite_PreparedStatement_iface class" on page 246

Remarks

Prepares a statement with placeholders, and then assigns values to the placeholders after executing the statement.

UltraLite_PreparedStatement_iface class

Syntax

public **UltraLite_PreparedStatement_iface**

Derived classes

- "UltraLite_PreparedStatement class" on page 245

Remarks

PreparedStatement interface.

Members

All members of UltraLite_PreparedStatement_iface, including all inherited members.

- "ExecuteQuery function" on page 246
- "ExecuteStatement function" on page 246
- "GetPlan function" on page 247
- "GetPlan function" on page 247
- "GetSchema function" on page 247
- "GetStreamWriter function" on page 248
- "HasResultSet function" on page 248
- "SetParameter function" on page 248
- "SetParameterNull function" on page 249

ExecuteQuery function

Synopsis

UltraLite_ResultSet * **UltraLite_PreparedStatement_iface::ExecuteQuery()**

Remarks

Executes a SQL SELECT statement as a query.

Returns

- The result set of the query, as a set of rows.

ExecuteStatement function

Synopsis

ul_s_long **UltraLite_PreparedStatement_iface::ExecuteStatement()**

Remarks

Executes a statement that does not return a result set, such as a SQL INSERT, DELETE or UPDATE statement.

 Copyright © 2007, iAnywhere Solutions, Inc.

GetPlan function

Synopsis

size_t **UltraLite_PreparedStatement_iface::GetPlan(**
  char * *buffer*,
  size_t *size*
)

Parameters

- **buffer** The buffer in which to receive the plan description.
- **size** The size, in ASCII characters, of the buffer.

Remarks

Gets a text-based description of query execution plan.

Returns

- A string describing the access plan UltraLite will use to execute a query. This function is intended primarily for use during development.

GetPlan function

Synopsis

size_t **UltraLite_PreparedStatement_iface::GetPlan(**
  ul_wchar * *buffer*,
  size_t *size*
)

Parameters

- **buffer** The buffer to receive the plan description.
- **size** The size, in ul_wchars, of the buffer.

Remarks

Gets a text-based description of query execution plan in wide characters.

Returns

- A string describing the access plan UltraLite will use to execute a query. This function is intended primarily for use during development.

GetSchema function

Synopsis

UltraLite_ResultSetSchema * **UltraLite_PreparedStatement_iface::GetSchema()**

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Gets the schema for the result set.

GetStreamWriter function

Synopsis

UltraLite_StreamWriter * **UltraLite_PreparedStatement_iface::GetStreamWriter(**
ul_column_num *parameter_id*
)

Parameters

- **parameter_id** A column identifier, which may be either a 1-based ordinal number or a column name.

Remarks

Gets a stream writer for streaming string/binary data into a parameter.

HasResultSet function

Synopsis

bool **UltraLite_PreparedStatement_iface::HasResultSet()**

Remarks

Determines if the SQL statement has a result set.

Returns

- True if a result set is generated when this statement is executed.
- False if no result set is generated.

SetParameter function

Synopsis

void **UltraLite_PreparedStatement_iface::SetParameter(**
ul_column_num *parameter_id*,
ULValue const & *value*
)

Parameters

- **parameter_id** The 1-based ordinal of the parameter.
- **value** The value to set the parameter.

Remarks

Sets a parameter for the SQL statement.

Copyright © 2007, iAnywhere Solutions, Inc.

SetParameterNull function

Synopsis

```
void UltraLite_PreparedStatement_iface::SetParameterNull(
  ul_column_num parameter_id
)
```

Parameters

- **parameter_id** The 1-based ordinal of the parameter.

Remarks

Sets a parameter to null.

UltraLite_ResultSet class

Syntax

public **UltraLite_ResultSet**

Base classes

- "UltraLite_SQLObject_iface class" on page 258
- "UltraLite_ResultSet_iface class" on page 251
- "UltraLite_Cursor_iface class" on page 223

Remarks

Represents an editable result set in an UltraLite database.

An editable result set allows you to perform positioned updates and deletes.

 Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_ResultSet_iface class

Syntax

public **UltraLite_ResultSet_iface**

Derived classes

- "UltraLite_ResultSet class" on page 250

Remarks

ResultSet interface.

Members

All members of UltraLite_ResultSet_iface, including all inherited members.

- "DeleteNamed function" on page 251
- "GetSchema function" on page 251

DeleteNamed function

Synopsis

bool **UltraLite_ResultSet_iface::DeleteNamed(**
  const ULValue & *table_name*
)

Parameters

- **table_name** A table name or its correlation (required when the database has multiple columns that share the same table nam.

Remarks

Deletes the current row and moves it to the next valid row.

GetSchema function

Synopsis

UltraLite_ResultSetSchema * **UltraLite_ResultSet_iface::GetSchema()**

Remarks

Gets the schema for this result set.

Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_ResultSetSchema class

Syntax

public **UltraLite_ResultSetSchema**

Base classes

- "UltraLite_SQLObject_iface class" on page 258
- "UltraLite_RowSchema_iface class" on page 253

Remarks

Retrieves schema information about a result set.

For example, column names, total number of columns, column scales, column sizes, and column SQL types.

Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_RowSchema_iface class

Syntax

public **UltraLite_RowSchema_iface**

Derived classes

- ♦ "UltraLite_ResultSetSchema class" on page 252
- ♦ "UltraLite_TableSchema class" on page 272

Remarks

RowSchema interface.

Members

All members of UltraLite_RowSchema_iface, including all inherited members.

- ♦ "GetBaseColumnName function" on page 253
- ♦ "GetColumnCount function" on page 254
- ♦ "GetColumnID function" on page 254
- ♦ "GetColumnName function" on page 254
- ♦ "GetColumnPrecision function" on page 255
- ♦ "GetColumnScale function" on page 256
- ♦ "GetColumnSize function" on page 256
- ♦ "GetColumnSQLName function" on page 255
- ♦ "GetColumnSQLType function" on page 256
- ♦ "GetColumnType function" on page 257

GetBaseColumnName function

Synopsis

ULValue **UltraLite_RowSchema_iface::GetBaseColumnName(**
  ul_column_num *column_id*
)

Parameters

- ♦ **column_id** A 1-based ordinal number.

Remarks

Gets the combined base and column name of a column of a result set, even if this column has a correlation name or alias.

Returns

- ♦ A combined ULValue class object.
- ♦ An empty name if the column is not part of a table.
- ♦ SQLE_COLUMN_NOT_FOUND is set if the column name does not exist.

GetColumnCount function

Synopsis

ul_column_num **UltraLite_RowSchema_iface::GetColumnCount()**

Remarks

Gets the number of columns in the table.

GetColumnID function

Synopsis

ul_column_num **UltraLite_RowSchema_iface::GetColumnID(**
const ULValue & *column_name*
)

Parameters

- **column_name** The column name.

Remarks

Gets the 1-based column ID.

Returns

- 0 if the column does not exist.
- Sets SQLE_COLUMN_NOT_FOUND if the column name does not exist.

GetColumnName function

Synopsis

ULValue **UltraLite_RowSchema_iface::GetColumnName(**
ul_column_num *column_id*
)

Parameters

- **column_id** A 1-based ordinal number.

Remarks

Gets the name of a column given its 1-based ID.

This will be the alias or correlation name for SELECT statements.

Returns

- An empty ULValue class object if the column does not exist.
- Sets SQLE_COLUMN_NOT_FOUND if the column name does not exist

Copyright © 2007, iAnywhere Solutions, Inc.

GetColumnPrecision function

Synopsis

```
size_t UltraLite_RowSchema_iface::GetColumnPrecision(
  const ULValue & column_id
)
```

Parameters

- **column_id** A 1-based ordinal number.

Remarks

Gets the precision of a numeric column.

Returns

- 0 if the column is not a numeric type or if the column does not exist.
- Sets SQLE_COLUMN_NOT_FOUND if the column name does not exist.
- Sets SQLE_DATATYPE_NOT_ALLOWED if the column type is not a numeric column.

GetColumnSQLName function

Synopsis

```
ULValue UltraLite_RowSchema_iface::GetColumnSQLName(
  ul_column_num column_id
)
```

Parameters

- **column_id** A 1-based ordinal number.

Remarks

Gets the SQL name for a column in a result set.

If the column has an alias, then that name is used. Otherwise, if the column in the result set corresponds to a column in a table, then the column name is used. Otherwise, the combined name is empty.

Returns

- A combined ULValue class object.
- SQLE_COLUMN_NOT_FOUND is set if the column name does not exist.

GetColumnSQLType function

Synopsis

ul_column_sql_type **UltraLite_RowSchema_iface::GetColumnSQLType(**
const ULValue & *column_id*
)

Parameters

- **column_id** A 1-based ordinal number.

Remarks

Gets the SQL type of a column.

See the ul_column_sql_type in ulprotos.h.

Returns

- UL_SQLTYPE_BAD_INDEX if the column does not exist.

GetColumnScale function

Synopsis

size_t **UltraLite_RowSchema_iface::GetColumnScale(**
const ULValue & *column_id*
)

Parameters

- **column_id** A 1-based ordinal number.

Remarks

Gets the scale of a numeric column.

Returns

- 0 if the column is not a numeric type or if the column does not exist.
- Sets SQLE_COLUMN_NOT_FOUND if the column name does not exist.
- Sets SQLE_DATATYPE_NOT_ALLOWED if the column type is not a numeric.

GetColumnSize function

Synopsis

size_t **UltraLite_RowSchema_iface::GetColumnSize(**
const ULValue & *column_id*
)

 Copyright © 2007, iAnywhere Solutions, Inc.

Parameters

- **column_id** A 1-based ordinal number.

Remarks

Gets the size of the column.

Returns

- 0 if the column does not exist or if the column type does not have a variable length.
- Sets SQLE_COLUMN_NOT_FOUND if the column name does not exist.
- Sets SQLE_DATATYPE_NOT_ALLOWED the column type is not either UL_SQLTYPE_CHAR or UL_SQLTYPE_BINARY.

GetColumnType function

Synopsis

ul_column_storage_type **UltraLite_RowSchema_iface::GetColumnType(**
const ULValue & *column_id*
)

Parameters

- **column_id** A 1-based ordinal number.

Remarks

Gets the type of a column.

See the ul_column_storage_type enum in ulprotos.h.

Returns

- UL_TYPE_BAD_INDEX if the column does not exist.

Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_SQLObject_iface class

Syntax

public **UltraLite_SQLObject_iface**

Derived classes

- "UltraLite_Connection class" on page 209
- "UltraLite_DatabaseSchema class" on page 234
- "UltraLite_IndexSchema class" on page 239
- "UltraLite_PreparedStatement class" on page 245
- "UltraLite_ResultSet class" on page 250
- "UltraLite_ResultSetSchema class" on page 252
- "UltraLite_StreamReader class" on page 260
- "UltraLite_StreamWriter class" on page 264
- "UltraLite_Table class" on page 265
- "UltraLite_TableSchema class" on page 272

Remarks

The SQLObject interface.

Members

All members of UltraLite_SQLObject_iface, including all inherited members.

- "AddRef function" on page 258
- "GetConnection function" on page 258
- "GetIFace function" on page 259
- "Release function" on page 259

AddRef function

Synopsis

ul_ret_void **UltraLite_SQLObject_iface::AddRef()**

Remarks

Increases the internal reference count for an object.

Note that you must match each call to this function with a call to Release function in order to free the object.

GetConnection function

Synopsis

UltraLite_Connection * **UltraLite_SQLObject_iface::GetConnection()**

 Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Gets the Connection object.

Returns

♦ The connection associated with this object.

GetIFace function

Synopsis

ul_void * **UltraLite_SQLObject_iface::GetIFace(**
  ul_iface_id *iface*
)

Parameters

♦ **iface** Reserved for future use.

Remarks

Reserved for future use.

Release function

Synopsis

ul_u_long **UltraLite_SQLObject_iface::Release()**

Remarks

Releases a reference to an object.

The object is freed once you have removed all references. You must call this function at least once. If you use AddRef function you also need a matching call from each AddRef function.

UltraLite_StreamReader class

Syntax

public **UltraLite_StreamReader**

Base classes

- "UltraLite_SQLObject_iface class" on page 258
- "UltraLite_StreamReader_iface class" on page 261

Remarks

Represents an UltraLite StreamReader.

 Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_StreamReader_iface class

Syntax

public **UltraLite_StreamReader_iface**

Derived classes

- “UltraLite_StreamReader class” on page 260

Remarks

StreamReader interface.

This interface supports reading/retrieving of VARCHAR and BINARY columns.

Members

All members of UltraLite_StreamReader_iface, including all inherited members.

- “GetByteChunk function” on page 261
- “GetLength function” on page 262
- “GetStringChunk function” on page 262
- “GetStringChunk function” on page 262
- “SetReadPosition function” on page 263

GetByteChunk function

Synopsis

bool **UltraLite_StreamReader_iface::GetByteChunk(**
ul_byte * *data*,
size_t *buffer_len*,
size_t * *len_retn*,
bool * *morebytes*
)

Parameters

- **data** A pointer to an array of bytes.
- **buffer_len** The length of the buffer, or array. The buffer_len must be greater than or equal to 0.
- **len_retn** An output parameter. The length returned.
- **morebytes** An output parameter. True if there are more bytes to read.

Remarks

Gets a byte chunk from current StreamReader offset by copying buffer_len bytes in to buffer data.

The bytes are read from where the last read left off unless you use SetReadPosition function.

Copyright © 2007, iAnywhere Solutions, Inc.

GetLength function

Synopsis

size_t **UltraLite_StreamReader_iface::GetLength(**
 bool *fetch_as_chars*
)

Parameters

- **fetch_as_chars** False for byte length, true for char length.

Remarks

Gets the length of a string/binary value.

Returns

- The number of bytes for binary values (fetch_as_chars is ignored for binaries).
- The number of characters or bytes for string values.

GetStringChunk function

Synopsis

bool **UltraLite_StreamReader_iface::GetStringChunk(**
 ul_wchar * *str*,
 size_t *buffer_len*,
 size_t * *len_retn*,
 bool * *morebytes*
)

Parameters

- **str** A pointer to an array ofwide characters.
- **buffer_len** The length of the buffer.
- **len_retn** An output parameter. The length returned.
- **morebytes** An output parameter. True if there are more characters to read.

Remarks

Gets a string chunk from current StreamReader offset by copying buffer_len wide characters in to buffer str.

Characters are read from where the last read left off unless you use SetReadPosition function.

GetStringChunk function

Synopsis

bool **UltraLite_StreamReader_iface::GetStringChunk(**
 char * *str*,

Copyright © 2007, iAnywhere Solutions, Inc.

size_t *buffer_len*,
size_t * *len_retn*,
bool * *morebytes*
)

Parameters

- **str** A pointer to an array of characters.
- **buffer_len** The length of the buffer, or array. The buffer_len must be greater than or equal to 0.
- **len_retn** An output parameter. The length returned.
- **morebytes** An output parameter. True if there are more characters to read.

Remarks

Gets a string chunk from current StreamReader offset by copying buffer_len bytes in to buffer str.

The characters are read from where the last read left off unless you use SetReadPosition function.

SetReadPosition function

Synopsis

bool **UltraLite_StreamReader_iface::SetReadPosition(**
size_t *offset*,
bool *offset_in_chars*
)

Parameters

- **offset** The offset.
- **offset_in_chars** True if offset is in characters. False if the offset is in bytes.

Remarks

Sets the offset in the data for the next read.

UltraLite_StreamWriter class

Syntax

public **UltraLite_StreamWriter**

Base classes

- "UltraLite_SQLObject_iface class" on page 258

Remarks

Represents an UltraLite StreamWriter.

 Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_Table class

Syntax

public **UltraLite_Table**

Base classes

- "UltraLite_SQLObject_iface class" on page 258
- "UltraLite_Table_iface class" on page 266
- "UltraLite_Cursor_iface class" on page 223

Remarks

Represents a table in an UltraLite database.

UltraLite_Table_iface class

Syntax

public **UltraLite_Table_iface**

Derived classes

- "UltraLite_Table class" on page 265

Remarks

Represents a table interface.

Members

All members of UltraLite_Table_iface, including all inherited members.

- "DeleteAllRows function" on page 266
- "Find function" on page 267
- "FindBegin function" on page 267
- "FindFirst function" on page 267
- "FindLast function" on page 268
- "FindNext function" on page 268
- "FindPrevious function" on page 268
- "GetSchema function" on page 269
- "Insert function" on page 269
- "InsertBegin function" on page 269
- "Lookup function" on page 270
- "LookupBackward function" on page 270
- "LookupBegin function" on page 270
- "LookupForward function" on page 271
- "TruncateTable function" on page 271

DeleteAllRows function

Synopsis

bool **UltraLite_Table_iface::DeleteAllRows()**

Remarks

Deletes all rows from table.

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 function.

 Copyright © 2007, iAnywhere Solutions, Inc.

Returns

- ♦ True on success.
- ♦ False on failure. For example, the table is not open, or there was an SQL error, and so on.

Find function

Synopsis

bool **UltraLite_Table_iface::Find(**
  ul_column_num *ncols*
)

Parameters

- ♦ **ncols** For composite indexes, the number of columns to use in the lookup.

Remarks

This function is the equivalent to FindFirst function.

Does an exact match lookup based on the current index scanning forward through the table.

FindBegin function

Synopsis

bool **UltraLite_Table_iface::FindBegin()**

Remarks

Prepares to perform a new Find on a table by entering find mode.

You may only set columns in the index that the table was opened with.

FindFirst function

Synopsis

bool **UltraLite_Table_iface::FindFirst(**
  ul_column_num *ncols*
)

Parameters

- ♦ **ncols** For composite indexes, the number of columns to use in the lookup.

Remarks

Does an exact match lookup based on the current index scanning forward through the table.

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. If no row matches the index value, the cursor position is AfterLast() and the function returns false.

FindLast function

Synopsis

bool **UltraLite_Table_iface::FindLast(**
ul_column_num *ncols*
)

Parameters

- **ncols** For composite indexes, the number of columns to use in the lookup.

Remarks

Does an exact match lookup based on the current index scanning backward through the table.

To specify the value to search for, set the column value for each column in the index. The cursor is left positioned on the first row that exactly matches the index value. If no row matches the index value, the cursor position is BeforeFirst() and the function returns false.

FindNext function

Synopsis

bool **UltraLite_Table_iface::FindNext(**
ul_column_num *ncols*
)

Parameters

- **ncols** For composite indexes, the number of columns to use in the lookup.

Remarks

Gets the next row that exactly matches the index.

Returns

- False if no more rows match the index. In this case the cursor is positioned after the last row.

FindPrevious function

Synopsis

bool **UltraLite_Table_iface::FindPrevious(**
ul_column_num *ncols*
)

Copyright © 2007, iAnywhere Solutions, Inc.

Parameters

- **ncols** For composite indexes, the number of columns to use in the lookup.

Remarks

Gets the previous row that exactly matches the index.

Returns

- False if no more rows match the index. In this case the cursor is positioned before the first row.

GetSchema function

Synopsis

UltraLite_TableSchema * **UltraLite_Table_iface::GetSchema()**

Remarks

Gets a schema object for this table.

Insert function

Synopsis

bool **UltraLite_Table_iface::Insert()**

Remarks

Inserts a new row into the table.

The table must be in insert mode for this operation to succeed. Use InsertBegin function to switch to insert mode.

InsertBegin function

Synopsis

bool **UltraLite_Table_iface::InsertBegin()**

Remarks

Selects insert mode for setting columns.

All columns may be modified in this mode.

Lookup function

Synopsis

bool **UltraLite_Table_iface::Lookup(**
  ul_column_num *ncols*
)

Parameters

- **ncols** For composite indexes, the number of columns to use in the lookup.

Remarks

This function is the equivalent to LookupForward.

Does a lookup based on the current index scanning forward through the table.

If resulting cursor position is AfterLast(), the return value is false.

LookupBackward function

Synopsis

bool **UltraLite_Table_iface::LookupBackward(**
  ul_column_num *ncols*
)

Parameters

- **ncols** For composite indexes, the number of columns to use in the lookup.

Remarks

Does a lookup based on the current index scanning backward through the table.

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.

Returns

- If resulting cursor position is BeforeFirst(), the return value is false.

LookupBegin function

Synopsis

bool **UltraLite_Table_iface::LookupBegin()**

Remarks

Prepares to perform a new Find on a table by entering lookup mode.

 Copyright © 2007, iAnywhere Solutions, Inc.

You may only set columns in the index that the table was opened with.

LookupForward function

Synopsis

bool **UltraLite_Table_iface::LookupForward(**
  ul_column_num *ncols*
)

Parameters

- **ncols** For composite indexes, the number of columns to use in the lookup.

Remarks

Does a lookup based on the current index scanning forward through the table.

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.

Returns

- If resulting cursor position is AfterLast(), the return value is false.

TruncateTable function

Synopsis

bool **UltraLite_Table_iface::TruncateTable()**

Remarks

Truncates the table and temporarily activates STOP SYNCHRONIZATION DELETE.

UltraLite_TableSchema class

Syntax

public **UltraLite_TableSchema**

Base classes

- "UltraLite_SQLObject_iface class" on page 258
- "UltraLite_TableSchema_iface class" on page 273
- "UltraLite_RowSchema_iface class" on page 253

Remarks

Represents a table schema.

 Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite_TableSchema_iface class

Syntax

public **UltraLite_TableSchema_iface**

Derived classes

- "UltraLite_TableSchema class" on page 272

Remarks

TableSchema interface.

Members

All members of UltraLite_TableSchema_iface, including all inherited members.

- "GetColumnDefault function" on page 273
- "GetGlobalAutoincPartitionSize function" on page 274
- "GetID function" on page 274
- "GetIndexCount function" on page 275
- "GetIndexName function" on page 275
- "GetIndexSchema function" on page 275
- "GetName function" on page 276
- "GetOptimalIndex function" on page 276
- "GetPrimaryKey function" on page 276
- "GetPublicationPredicate function" on page 277
- "GetUploadUnchangedRows function" on page 277
- "InPublication function" on page 277
- "IsColumnAutoinc function" on page 278
- "IsColumnCurrentDate function" on page 278
- "IsColumnCurrentTime function" on page 279
- "IsColumnCurrentTimestamp function" on page 279
- "IsColumnGlobalAutoinc function" on page 279
- "IsColumnInIndex function" on page 280
- "IsColumnNewUUID function" on page 281
- "IsColumnNullable function" on page 281
- "IsNeverSynchronized function" on page 281

GetColumnDefault function

Synopsis

ULValue **UltraLite_TableSchema_iface::GetColumnDefault(**
  const ULValue & *column_id*
)

Parameters

- **column_id** The ID number of the column. The value must be a 1-based ordinal number. The first column in the table has an ID value of one.

Remarks

Gets the default value for the column if it exists.

Returns

- The default contained as a string.
- Is empty if the column does not contain a default value. Sets SQLE_COLUMN_NOT_FOUND if the column name does not exist.

The ULValue class object returned has:

GetGlobalAutoincPartitionSize function

Synopsis

bool **UltraLite_TableSchema_iface::GetGlobalAutoincPartitionSize(**
const ULValue & *column_id*,
ul_u_big * *size*
)

Parameters

- **column_id** The ID number of the column. The value must be a 1-based ordinal number. The first column in the table has an ID value of one.
- **size** An output parameter. The partition size for the column. All global autoincrement columns in a given table share the same global autoincrement partition.

Remarks

Gets the partition size.

Returns

- The partition size for a global autoincrement column.

GetID function

Synopsis

ul_table_num **UltraLite_TableSchema_iface::GetID()**

Remarks

Gets the table ID.

 Copyright © 2007, iAnywhere Solutions, Inc.

GetIndexCount function

Synopsis

ul_index_num **UltraLite_TableSchema_iface::GetIndexCount()**

Remarks

Gets the number of indexes in the table.

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.

Returns

- The number of indexes in the table.

GetIndexName function

Synopsis

ULValue **UltraLite_TableSchema_iface::GetIndexName(**
  ul_index_num *index_id*
)

Parameters

- **index_id** A 1-based ordinal number.

Remarks

Gets the index name given its 1-based ID.

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.

Returns

- A ULValue class object is empty if the index does not exist.

GetIndexSchema function

Synopsis

UltraLite_IndexSchema * **UltraLite_TableSchema_iface::GetIndexSchema(**
  const ULValue & *index_id*
)

Parameters

- **index_id** The name or ID number identifying the index.

Remarks

Gets an IndexSchema object with the given name or ID.

Copyright © 2007, iAnywhere Solutions, Inc.

Returns

- UL_NULL if the index does not exist.

GetName function

Synopsis

ULValue **UltraLite_TableSchema_iface::GetName()**

Remarks

Gets the name of the table.

Returns

- The name of the table as a string.

GetOptimalIndex function

Synopsis

ULValue **UltraLite_TableSchema_iface::GetOptimalIndex(**
  const ULValue & *column_id*
)

Parameters

- **column_id** The ID number of the column. The value must be a 1-based ordinal number. The first column in the table has an ID value of one.

Remarks

Determines the best index to use for searching for a column value.

Returns

The name of the index.

GetPrimaryKey function

Synopsis

UltraLite_IndexSchema * **UltraLite_TableSchema_iface::GetPrimaryKey()**

Remarks

Gets the primary key for the table.

 Copyright © 2007, iAnywhere Solutions, Inc.

GetPublicationPredicate function

Synopsis

ULValue **UltraLite_TableSchema_iface::GetPublicationPredicate(**
 const ULValue & *publication_name*
)

Parameters

♦ **publication_name** The name of the publication.

Remarks

Gets the publication predicate as a string.

Returns

♦ The publication predicate string for the specified publication.

♦ Sets SQLE_PUBLICATION_NOT_FOUND if the publication does not exist.

GetUploadUnchangedRows function

Synopsis

bool **UltraLite_TableSchema_iface::GetUploadUnchangedRows()**

Remarks

Checks whether the database has been configured to upload rows that have not changed.

Tables set to upload unchanged as well as changed rows are sometimes referred to as allsync tables.

Returns

♦ True if the table is marked to always upload all rows during synchronization.

♦ False if the table is marked to upload only changed rows.

InPublication function

Synopsis

bool **UltraLite_TableSchema_iface::InPublication(**
 const ULValue & *publication_name*
)

Parameters

♦ **publication_name** The name of the publication.

Remarks

Checks whether the table is contained in the named publication.

Returns

- True if the table is contained in the publication.
- False if the table is not in the publication.
- Sets SQLE_PUBLICATION_NOT_FOUND if the publication does not exist.

IsColumnAutoinc function

Synopsis

```
bool UltraLite_TableSchema_iface::IsColumnAutoinc(
  const ULValue & column_id
)
```

Parameters

- **column_id** The ID number of the column. The value must be a 1-based ordinal number. The first column in the table has an ID value of one.

Remarks

Checks whether the specified column's default is set to autoincrement.

Returns

- True if the column default is set to be autoincremented.
- False if the column is not autoincremented.
- Sets SQLE_COLUMN_NOT_FOUND if the column name does not exist.

IsColumnCurrentDate function

Synopsis

```
bool UltraLite_TableSchema_iface::IsColumnCurrentDate(
  const ULValue & column_id
)
```

Parameters

- **column_id** The ID number of the column. The value must be a 1-based ordinal number. The first column in the table has an ID value of one.

Remarks

Checks whether the specified column's default is set to the current date.

Returns

- True if the column has a current date default.
- False if the column does not default to the current date.

Copyright © 2007, iAnywhere Solutions, Inc.

IsColumnCurrentTime function

Synopsis

bool **UltraLite_TableSchema_iface::IsColumnCurrentTime(**
 const ULValue & *column_id*
)

Parameters

- **column_id** The ID number of the column. The value must be a 1-based ordinal number. The first column in the table has an ID value of one.

Remarks

Checks whether the specified column's default is set to the current time.

Returns

- True if the column has a current time default.
- False if it does not.

IsColumnCurrentTimestamp function

Synopsis

bool **UltraLite_TableSchema_iface::IsColumnCurrentTimestamp(**
 const ULValue & *column_id*
)

Parameters

- **column_id** The ID number of the column. The value must be a 1-based ordinal number. The first column in the table has an ID value of one.

Remarks

Checks whether the specified column's default is set to the current timestamp.

Returns

- True if the column has a current timestamp default.
- False if it does not.

IsColumnGlobalAutoinc function

Synopsis

bool **UltraLite_TableSchema_iface::IsColumnGlobalAutoinc(**
 const ULValue & *column_id*
)

Parameters

- **column_id** The ID number of the column. The value must be a 1-based ordinal number. The first column in the table has an ID value of one.

Remarks

Checks whether the specified column's default is set to autoincrement.

Returns

- True if the column is autoincremented.
- False if it is not autoincremente. SQLE_COLUMN_NOT_FOUND is set if the column name does not exist.

IsColumnInIndex function

Synopsis

```
bool UltraLite_TableSchema_iface::IsColumnInIndex(
  const ULValue & column_id,
  const ULValue & index_id
)
```

Parameters

- **column_id** A 1-based ordinal number identifying the column. You can get the column_id by calling GetColumnCount function.
- **index_id** A 1-based ordinal number identifying the index. You can get the number of indexes in a table by calling GetIndexCount function.

Remarks

Checks whether the table is contained in the named index.

Returns

- True if the column is contained in the index.
- False if the column is not contained in the index.
- Sets SQLE_COLUMN_NOT_FOUND if the column name does not exist.
- Sets SQLE_INDEX_NOT_FOUND if the index does not exist.

.

 Copyright © 2007, iAnywhere Solutions, Inc.

IsColumnNewUUID function

Synopsis

bool **UltraLite_TableSchema_iface::IsColumnNewUUID(**
 const ULValue & *column_id*
)

Parameters

- **column_id** The ID number of the column. The value must be a 1-based ordinal number. The first column in the table has an ID value of one.

Remarks

Checks whether the specified column's default is set to a new UUID.

Returns

- True if the column has a new UUID default.
- False if the column does not default to a new UUID.

IsColumnNullable function

Synopsis

bool **UltraLite_TableSchema_iface::IsColumnNullable(**
 const ULValue & *column_id*
)

Parameters

- **column_id** The ID number of the column. The value must be a 1-based ordinal number. The first column in the table has an ID value of one. The specified column is the first column in the index, but the index may have more than one column.

Remarks

Checks whether the specified column is nullable.

Returns

- True if the column is nullable.
- False if it is not nullable.
- Sets SQLE_COLUMN_NOT_FOUND if the column name does not exist.

IsNeverSynchronized function

Synopsis

bool **UltraLite_TableSchema_iface::IsNeverSynchronized()**

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

Checks whether the table is marked as never being synchronized.

Returns

- True if the table is omitted from syncronization. Tables marked as never being synchronized are never synchronized, even if they are included in a publication. These tables are sometimes referred to as nosync tables.
- False if the table is included as a synchronizable table.

Copyright © 2007, iAnywhere Solutions, Inc.

ULValue class

Syntax

public **ULValue**

Remarks

ULValue class class.

The ULValue class is a wrapper for the data types stored in an UltraLite cursor. This class allows you to store data without having to worry about the data type, and is used to pass values to and from the UltraLite C++ Component.

ULValue class contains many constructors and cast operators, so you can use ULValue class seamlessly (in most cases) without explicitly instantiating a ULValue class.

You can construct the object or assign it from any basic C++ data type. You can also cast it into any basic C++ data type.

```
 x( 5 );       ULValue// Example of ULValue's constructor
 y = 5;        ULValue// Example of ULValue's assignment operator
int z = y;            // Example of ULValue's cast operator
```

This sample works for strings as well:

```
 x( UL_TEXT( ULValue"hello" ) );
 y = UL_TEXT( ULValue"hello" );
y.( buffer, BUFFER_LEN );    GetString// NOTE, there is no cast operator
```

You do not need to explicitly construct a ULValue class object as the compiler does this automatically in many cases. For example, to fetch a value from a column, you can use the following:

```
int x = table->Get( UL_TEXT( "my_column" ) );
```

The table->Get() call returns a ULValue class object. C++ automatically calls the cast operator in order to convert it to an integer. Similarly, the table->Get() call takes a ULValue class parameter as the column identifier. This determines which column to fetch. C++ automatically converts the "my_column" literal string into a ULValue class object.

Members

All members of ULValue, including all inherited members.

- "GetBinary function" on page 284
- "GetBinary function" on page 285
- "GetBinaryLength function" on page 285
- "GetCombinedStringItem function" on page 286
- "GetCombinedStringItem function" on page 286
- "GetString function" on page 286
- "GetString function" on page 287
- "GetStringLength function" on page 287
- "InDatabase function" on page 288
- "IsNull function" on page 288

Copyright © 2007, iAnywhere Solutions, Inc.

- "operator bool function" on page 296
- "operator DECL_DATETIME function" on page 296
- "operator double function" on page 296
- "operator float function" on page 296
- "operator int function" on page 297
- "operator long function" on page 297
- "operator short function" on page 297
- "operator ul_s_big function" on page 297
- "operator ul_u_big function" on page 297
- "operator unsigned char function" on page 297
- "operator unsigned int function" on page 298
- "operator unsigned long function" on page 298
- "operator unsigned short function" on page 298
- "operator= function" on page 298
- "SetBinary function" on page 289
- "SetString function" on page 289
- "SetString function" on page 289
- "StringCompare function" on page 290
- "ULValue function" on page 290
- "ULValue function" on page 290
- "ULValue function" on page 291
- "ULValue function" on page 291
- "ULValue function" on page 291
- "ULValue function" on page 291
- "ULValue function" on page 292
- "ULValue function" on page 292
- "ULValue function" on page 292
- "ULValue function" on page 293
- "ULValue function" on page 293
- "ULValue function" on page 293
- "ULValue function" on page 293
- "ULValue function" on page 294
- "ULValue function" on page 294
- "ULValue function" on page 294
- "ULValue function" on page 295
- "ULValue function" on page 295
- "ULValue function" on page 295
- "ULValue function" on page 295
- "~ULValue function" on page 298

GetBinary function

Synopsis

void **ULValue::GetBinary(**
  p_ul_binary *bin*,
  size_t *len*
)

 Copyright © 2007, iAnywhere Solutions, Inc.

Parameters

- **bin** The binary structure to receive bytes.
- **len** The length of the buffer.

Remarks

Retrieves the current value into a binary buffer, casting as required.

If the buffer is too small, then the value is truncated. Up to len characters are copied to the given buffer.

GetBinary function

Synopsis

```
void ULValue::GetBinary(
  ul_byte * dst,
  size_t len,
  size_t * retr_len
)
```

Parameters

- **dst** The buffer to receive bytes.
- **len** The length of the buffer.
- **retr_len** An output parameter. The actual number of bytes returned.

Remarks

Retrieves the current value into a binary buffer, casting as required If the buffer is too small, then the value is truncated.

Up to len bytes are copied to the given buffer. The number of bytes actually copied is returned in retr_len.

GetBinaryLength function

Synopsis

```
size_t ULValue::GetBinaryLength()
```

Remarks

Gets the length of a Binary value.

Returns

- The number of bytes necessary to hold the binary value returned by GetBinary function.

GetCombinedStringItem function

Synopsis

void **ULValue::GetCombinedStringItem(**
 ul_u_short *selector*,
 char * *dst*,
 size_t *len*
)

Parameters

- **selector** Selected internal value.
- **dst** The buffer to receive string value.
- **len** The length, in bytes, of dst.

Remarks

Retrieves parts of a combined name into a string buffer, casting as required.

If the value is not combined, an empty string is copied. The output string is always null-terminated. If the buffer is too small, then the value is truncated. Up to len characters are copied to the given buffer, including the null terminator.

GetCombinedStringItem function

Synopsis

void **ULValue::GetCombinedStringItem(**
 ul_u_short *selector*,
 ul_wchar * *dst*,
 size_t *len*
)

Parameters

- **selector** Selected internal value.
- **dst** The buffer to receive string value.
- **len** The length, in wide chars, of dst.

Remarks

Gets selected portion of a combined String value.

GetString function

Synopsis

void **ULValue::GetString(**
 char * *dst*,
 size_t *len*
)

 Copyright © 2007, iAnywhere Solutions, Inc.

Parameters

- **dst** The buffer to receive string value.
- **len** The length, in bytes, of dst.

Remarks

Retrieves the current value into a string buffer, casting as required.

The output string is always null-terminated. If the buffer is too small then the value is truncated. Up to len characters are copied to the given buffer, including the null terminator.

GetString function

Synopsis

```
void ULValue::GetString(
  ul_wchar * dst,
  size_t len
)
```

Parameters

- **dst** The buffer to receive string value.
- **len** The length, in wide chars, of dst.

Remarks

Gets a String value.

Retrieves the current value into a string buffer, casting as required.

GetStringLength function

Synopsis

```
size_t ULValue::GetStringLength(
  bool fetch_as_chars
)
```

Parameters

- **fetch_as_chars** False for byte length, true for char length.

Remarks

Gets the length of a String.

Intended usage is as follows:

Copyright © 2007, iAnywhere Solutions, Inc.

```
len = v.GetStringLength();
dst = new char[ len ];
( dst, len ); GetString
```

For wide character apps the usage is:

```
len = v.GetStringLength( true );
dst = new ul_wchar[ len ];
( dst, len ); GetString
```

Returns

- The number of bytes or characters required to hold the string returned by one of the GetString function methods, including the null-terminator.

InDatabase function

Synopsis

bool **ULValue::InDatabase()**

Remarks

Checks if value is in the database.

Returns

- True if this object is referencing a cursor field.
- False if it is not.

IsNull function

Synopsis

bool **ULValue::IsNull()**

Remarks

Checks if the ULValue class ojbect is empty.

Returns

- True if this object is either an empty ULValue class object, or if it references a cursor field that is set to NULL.
- False otherwise.

Copyright © 2007, iAnywhere Solutions, Inc.

SetBinary function

Synopsis

void **ULValue::SetBinary(**
ul_byte * *src*,
size_t *len*
)

Parameters

- **src** A buffer of bytes.
- **len** The length of the buffer.

Remarks

Sets the value to reference the binary buffer provided.

No bytes are copied from the provided buffer until the value is used.

SetString function

Synopsis

void **ULValue::SetString(**
const char * *val*,
size_t *len*
)

Parameters

- **val** A pointer to the null-terminated string representation of this ULValue class.
- **len** The length of the string.

Remarks

Casts a ULValue class to a string.

SetString function

Synopsis

void **ULValue::SetString(**
const ul_wchar * *val*,
size_t *len*
)

Parameters

- **val** A pointer to the null-terminated unicode string representation of this ULValue class.
- **len** The length of the string.

Remarks

Casts a ULValue class to a UNICODE string.

StringCompare function

Synopsis

ul_compare **ULValue::StringCompare(**
 const ULValue & *value*
)

Parameters

- **value** The comparison string.

Remarks

Compares strings, or string representations of ULValue class objects.

Returns

- 0 if the strings are equal.
- -1 if the current value compares less than value.
- 1 if the current value compares greater than value.
- -3 if the sqlca of either ULValue class object is not set.
- -2 if the string representation of either ULValue class object is UL_NULL.

ULValue function

Synopsis

ULValue::ULValue()

Remarks

Constructs a ULValue class.

ULValue function

Synopsis

ULValue::ULValue(
 const ULValue & *vSrc*
)

Parameters

- **vSrc** A value to be treated as a ULValue class.

Remarks

Constructs a ULValue class, copying from an existing one.

 Copyright © 2007, iAnywhere Solutions, Inc.

ULValue function

Synopsis

ULValue::ULValue(
bool *val*
)

Parameters

- **val** A boolean value to be treated as a ULValue class.

Remarks

Constructs a ULValue class from a bool.

ULValue function

Synopsis

ULValue::ULValue(
short *val*
)

Parameters

- **val** A short value to be treated as a ULValue class.

Remarks

Constructs a ULValue class from a short.

ULValue function

Synopsis

ULValue::ULValue(
long *val*
)

Parameters

- **val** A long value to be treated as a ULValue class.

Remarks

Constructs a ULValue class from a long.

ULValue function

Synopsis

ULValue::ULValue(
int *val*
)

Copyright © 2007, iAnywhere Solutions, Inc.

Parameters

- **val** An INTEGER value to be treated as a ULValue class.

Remarks

Construct a ULValue class from an int.

ULValue function

Synopsis

```
ULValue::ULValue(
 unsigned int val
)
```

Parameters

- **val** An unsigned INTEGER value to be treated as a ULValue class.

Remarks

Constructs a ULValue class from an unsigned INTEGER.

ULValue function

Synopsis

```
ULValue::ULValue(
 float val
)
```

Parameters

- **val** A FLOAT value to be treated as a ULValue class.

Remarks

Constructs a ULValue class from a FLOAT.

ULValue function

Synopsis

```
ULValue::ULValue(
 double val
)
```

Parameters

- **val** A DOUBLE value to be treated as a ULValue class.

Remarks

Constructs a ULValue class from a DOUBLE.

 Copyright © 2007, iAnywhere Solutions, Inc.

ULValue function

Synopsis

ULValue::ULValue(
unsigned char *val*
)

Parameters

- **val** An unsigned CHAR to be treated as a ULValue class.

Remarks

Constructs a ULValue class from an unsigned CHAR.

ULValue function

Synopsis

ULValue::ULValue(
unsigned short *val*
)

Parameters

- **val** An unsigned SHORT to be treated as a ULValue class.

Remarks

Construct a ULValue class from an unsigned SHORT.

ULValue function

Synopsis

ULValue::ULValue(
unsigned long *val*
)

Parameters

- **val** An unsigned LONG to be treated as a ULValue class.

Remarks

Construct a ULValue class from an unsigned LONG.

ULValue function

Synopsis

ULValue::ULValue(
const ul_u_big & *val*
)

Copyright © 2007, iAnywhere Solutions, Inc.

Parameters

- **val** A ul_u_big value to be treated as a ULValue class.

Remarks

Construct a ULValue class from a ul_u_big.

ULValue function

Synopsis

```
ULValue::ULValue(
 const ul_s_big & val
)
```

Parameters

- **val** A ul_s_big value to be treated as a ULValue class.

Remarks

Construct a ULValue class from a ul_s_big.

ULValue function

Synopsis

```
ULValue::ULValue(
 const p_ul_binary val
)
```

Parameters

- **val** A ul_binary value to be treated as a ULValue class.

Remarks

Construct a ULValue class from a ul_binary.

ULValue function

Synopsis

```
ULValue::ULValue(
 DECL_DATETIME & val
)
```

Parameters

- **val** A DATETIME value to be treated as a ULValue class.

Remarks

Construct a ULValue class from a datetime.

 Copyright © 2007, iAnywhere Solutions, Inc.

ULValue function

Synopsis

ULValue::ULValue(
const char * *val*
)

Parameters

- **val** A pointer to a string to be treated as a ULValue class.

Remarks

Construct a ULValue class from a STRING.

ULValue function

Synopsis

ULValue::ULValue(
const ul_wchar * *val*
)

Parameters

- **val** A pointer to a UNICODE string to be treated as a ULValue class.

Remarks

Construct a ULValue class from a UNICODE string.

ULValue function

Synopsis

ULValue::ULValue(
const char * *val*,
size_t *len*
)

Parameters

- **val** A buffer holding the string to be treated as a ULValue class.
- **len** The length of the buffer.

Remarks

Construct a ULValue class from a buffer of characters.

ULValue function

Synopsis

ULValue::ULValue(
const ul_wchar * *val*,

Copyright © 2007, iAnywhere Solutions, Inc.

size_t *len*
)

Parameters

- **val** A buffer holding the string to be treated as a ULValue class.
- **len** The length of the buffer.

Remarks

Construct a ULValue class from a buffer of unicode characters.

operator DECL_DATETIME function

Synopsis

ULValue::operator DECL_DATETIME()

Remarks

Cast a ULValue class to a datetime.

operator bool function

Synopsis

ULValue::operator bool()

Remarks

Cast a ULValue class to a boolean.

operator double function

Synopsis

ULValue::operator double()

Remarks

Cast a ULValue class to a double.

operator float function

Synopsis

ULValue::operator float()

Remarks

Cast a ULValue class to a float.

 Copyright © 2007, iAnywhere Solutions, Inc.

operator int function

Synopsis

ULValue::operator int()

Remarks

Cast a ULValue class to an int.

operator long function

Synopsis

ULValue::operator long()

Remarks

Cast a ULValue class to a long.

operator short function

Synopsis

ULValue::operator short()

Remarks

Cast a ULValue class to a short.

operator ul_s_big function

Synopsis

ULValue::operator ul_s_big()

Remarks

Cast a ULValue class to a signed big int.

operator ul_u_big function

Synopsis

ULValue::operator ul_u_big()

Remarks

Cast a ULValue class to an unsigned big int.

operator unsigned char function

Synopsis

ULValue::operator unsigned char()

Remarks

Cast a ULValue class to a char.

operator unsigned int function

Synopsis

ULValue::operator unsigned int()

Remarks

Cast a ULValue class to an unsigned int.

operator unsigned long function

Synopsis

ULValue::operator unsigned long()

Remarks

Cast a ULValue class to an unsigned long.

operator unsigned short function

Synopsis

ULValue::operator unsigned short()

Remarks

Cast a ULValue class to an unsigned short.

operator= function

Synopsis

ULValue & **ULValue::operator=(**
const ULValue & *other*
)

Parameters

- **other** The value to be assigned to a ULValue class.

Remarks

Override the = operator for ULValues.

~ULValue function

Synopsis

ULValue::~ULValue()

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

The destructor for ULValue class.

CHAPTER 13

Embedded SQL API Reference

Contents

Introduction to embedded SQL API 303
db_fini function 304
db_init function 305
db_start_database function 306
db_stop_database function 307
ULChangeEncryptionKey function 308
ULCheckpoint function 309
ULClearEncryptionKey function 310
ULCountUploadRows function 311
ULDropDatabase function 312
ULGetDatabaseID function 313
ULGetDatabaseProperty function 314
ULGetLastDownloadTime function 315
ULGetSynchResult function 316
ULGlobalAutoincUsage function 318
ULGrantConnectTo function 319
ULHTTPSStream function (deprecated) 320
ULHTTPStream function (deprecated) 321
ULInitSynchInfo function 322
ULIsSynchronizeMessage function 323
ULResetLastDownloadTime function 324
ULRetrieveEncryptionKey function 325
ULRevokeConnectFrom function 326
ULRollbackPartialDownload function 327
ULSaveEncryptionKey function 328
ULSetDatabaseID function 329
ULSetDatabaseOptionString function 330
ULSetDatabaseOptionULong 331
ULSetSynchInfo function 332

ULSocketStream function (deprecated) 333
ULSynchronize function 334

 Copyright © 2007, iAnywhere Solutions, Inc.

Introduction to embedded SQL API

This chapter 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 37.

Use the EXEC SQL INCLUDE SQLCA command to include prototypes for the functions in this chapter.

db_fini function

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 function" on page 305

Copyright © 2007, iAnywhere Solutions, Inc.

db_init function

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.

In most cases, 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 function" on page 304

db_start_database function

Starts a database if the database is not already running.

Syntax

unsigned int **db_start_database(** SQLCA * *sqlca*, char * *parms* **);**

Parameters

sqlca A pointer to a SQLCA structure.

parms A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD**=*value*. Typically, only a file name is required. For example:

```
"DBF=c:\\db\\mydatabase.db"
```

Returns

- True if the database was already running or was successfully started. In this case, SQLCODE is set to 0.
- Error information is also returned in the SQLCA.

Remarks

Required when developing applications that combine embedded SQL and the C++ component.

See also

- "UltraLite Connection String Parameters Reference" [*UltraLite - Database Management and Reference*]
- "Initializing the SQL Communications Area" on page 41

 Copyright © 2007, iAnywhere Solutions, Inc.

db_stop_database function

Stop a database.

Syntax

unsigned int **db_stop_database(** SQLCA * *sqlca***,** char * *parms* **);**

Parameters

sqlca A pointer to a SQLCA structure.

parms A null-terminated string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD**=*value*. Typically, only a database file name is needed. For example,

```
"DBF=c:\\db\\mydatabase.db"
```

Returns

- TRUE if there were no errors.

Remarks

This function is not commonly needed, as UltraLite automatically stops the database when all connections are closed. However, this function may be useful when developing applications that combine embedded SQL and the C++ component.

This function does not stop a database that has existing connections.

See also

- "UltraLite Connection String Parameters Reference" [*UltraLite - Database Management and Reference*]
- "Initializing the SQL Communications Area" on page 41

ULChangeEncryptionKey function

Changes the encryption key for an UltraLite database.

Syntax

ul_bool **ULChangeEncryptionKey(** SQLCA **sqlca*, ul_char **new_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.

See also

♦ "Encrypting data" on page 61

 Copyright © 2007, iAnywhere Solutions, Inc.

ULCheckpoint function

Performs a checkpoint operation: flushing any pending committed transactions to the database. 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. For more information, see: "Flushing single or grouped transactions" [*UltraLite - Database Management and Reference*].

Syntax

void **ULCheckpoint(** SQLCA * *sqlca***);**

Remarks

The ULCheckpoint function ensures that all pending committed transactions have been written to the database storage.

See also

- "UltraLite transaction processing and isolation levels" [*UltraLite - Database Management and Reference*]

ULClearEncryptionKey function

Clears the encryption key.

Syntax

ul_bool **ULClearEncryptionKey(**
ul_u_long * *creator***,**
ul_u_long * *feature-num* **);**

Parameters

creator A pointer to the creator ID of the feature that holds the encryption key. The default value is NULL.

feature-num A pointer to the feature number that holds the encryption key. If feature-num is NULL, the application uses the UltraLite default value of 100.

Remarks

On the Palm OS the encryption key is saved in dynamic memory as a Palm feature. Features are indexed by creator and a feature number.

See also

- "ULRetrieveEncryptionKey function" on page 325
- "ULSaveEncryptionKey function" on page 328

 Copyright © 2007, iAnywhere Solutions, Inc.

ULCountUploadRows function

Counts the number of rows that need to be uploaded for synchronization.

Syntax

ul_u_long **ULCountUploadRows (**
SQLCA * *sqlca*,
ul_publication_mask *publication-mask*,
ul_u_long *threshold* **);**

Parameters

sqlca A pointer to the SQLCA.

publication-mask A set of publications to check. A value of 0 indicates that the entire database needs to be checked. Otherwise, you need to supply a publication mask of OR'd publications. For example:

```
UL_PUB_PUB1 | UL_PUB_PUB2
```

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).
- 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 set of publications or in the whole database.

Remarks

Use this function to prompt users to synchronize.

Example

The following call checks the entire database for the number of rows to be synchronized:

```
count = ULCountUploadRows( sqlca, 0, 0 );
```

The following call checks publications PUB1 and PUB2 for a maximum of 1000 rows:

```
count = ULCountUploadRows( sqlca,
UL_PUB_PUB1 | UL_PUB_PUB2, 1000 );
```

The following call checks to see if any rows need to be synchronized:

```
count = ULCountUploadRows( sqlca, UL_SYNC_ALL, 1 );
```

See also

- “PublicationMask synchronization parameter” [*MobiLink - Client Administration*]

ULDropDatabase function

Deletes the UltraLite database file.

Syntax

ul_bool **ULDropDatabase (** SQLCA * *sqlca*, ul_char * *store-parms* **);**

Parameters

sqlca A pointer to the SQLCA.

store-parms A null-terminated connection string containing a semicolon-delimited list of parameter settings, each of the form **KEYWORD**=*value*.

Returns

- **ul_true** The database was successfully deleted.
- **ul_false** The database could not be deleted. A detailed error message is defined by the sqlcode field in the SQLCA. The usual reason for failure is that an incorrect file name was supplied or that access to the file was denied (perhaps because it is opened by an application).

Remarks

Only call this function when:

- You do not have an open database connection.
- Either before you have called db_init but after you have called db_fini.

On the Palm OS, only call this function only when:

- You are not connected to the database.
- After have called ULEnable functions.

Caution
This function deletes the database file and all data in it. This operation is not recoverable. Therefore, use this function with care.

Example

The following call deletes the UltraLite database file *myfile.udb*.

```
if( ULDropDatabase(&sqlca, UL_TEXT("file_name=myfile.udb") ) ){
    // success
};
```

 Copyright © 2007, iAnywhere Solutions, Inc.

ULGetDatabaseID function

Gets the current database ID.

Syntax

ul_u_long **ULGetDatabaseID(** SQLCA * *sqlca* **)**

Parameters

sqlca A pointer to the SQLCA.

Returns

- The value set by the last call to SetDatabaseID.
- UL_INVALID_DATABASE_ID if the ID was never set.

Remarks

The current database ID used for global autoincrement.

See also

- "UltraLite global_database_id option" [*UltraLite - Database Management and Reference*]

ULGetDatabaseProperty function

Obtains the value of a database property.

Syntax

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.

See also

- "UltraLite Database Settings Reference" [*UltraLite - Database Management and Reference*]

 Copyright © 2007, iAnywhere Solutions, Inc.

ULGetLastDownloadTime function

Obtains the last time a specified publication was downloaded.

Syntax

ul_bool **ULGetLastDownloadTime (**
SQLCA * *sqlca*,
ul_publication_mask *publication-mask*,
DECL_DATETIME * *value* **);**

Parameters

sqlca A pointer to the SQLCA.

publication-mask A set of publications for which the last download time is retrieved. A value of 0 indicates that all publications in the database requires a download timestamp. The set is supplied as a mask. For example, the following mask corresponds to publications PUB1 and PUB2.:

```
UL_PUB_PUB1 | UL_PUB_PUB2
```

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** The *value* is successfully populated by the last download time of the publication that was specified by *publication-mask*.
- **false** The *publication-mask* specifies more than one publication or that the publication is undefined. The contents of *value* are not meaningful.

Examples

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_PUB_PUB1, &dt );
```

The following call populates the dt structure with the date and time that the entire database was last downloaded. It uses the special UL_SYNC_ALL publication mask.

```
ret = ULGetLastDownloadTime( &sqlca, UL_SYNC_ALL, &dt );
```

See also

- “PublicationMask synchronization parameter” [*MobiLink - Client Administration*]
- “UL_SYNC_ALL macro” on page 173
- “UL_SYNC_ALL_PUBS macro” on page 174

ULGetSynchResult function

Stores the results of the most recent synchronization, so that appropriate action can be taken in the application:

Syntax

ul_bool **ULGetSynchResult(** ul_synch_result * *synch-result* **);**

Parameters

synch-result A structure to hold the synchronization result. It is defined in *ulglobal.h* as follows:

```
typedef struct {
   an_sql_code    sql_code;
   ul_stream_error   stream_error;
   ul_bool     upload_ok;
   ul_bool     ignored_rows;
   ul_auth_status auth_status;
   ul_s_long      auth_value;
   SQLDATETIME timestamp;
   ul_synch_status   status;
} ul_synch_result, * p_ul_synch_result;
```

The individual parameters include:

- **sql_code** The SQL code from the last synchronization. For a list of SQL codes, see "Error messages sorted by SQLSTATE" [*SQL Anywhere 10 - Error Messages*].
- **stream_error** A structure of type ul_stream error. See "Stream Error synchronization parameter" [*MobiLink - Client Administration*].
- **upload_ok** True if the upload was successful; false otherwise.
- **ignored_rows** True if uploaded rows were ignored; false otherwise.
- **auth_status** The synchronization authentication status. See "Authentication Status synchronization parameter" [*MobiLink - Client Administration*].
- **auth_value** The value used by the MobiLink server to determine the **auth_status** result. See "Authentication Value synchronization parameter" [*MobiLink - Client Administration*].
- **timestamp** The time and date of the last synchronization.
- **status** The status information used by the observer function. See "Observer synchronization parameter" [*MobiLink - Client Administration*].

Returns

- True if the operation succeeded.
- False if the operation failed.

Copyright © 2007, iAnywhere Solutions, Inc.

Remarks

The application must allocate a ul_synch_result object before passing it to ULGetSynchResult. The function fills the ul_synch_result with the result of the last synchronization. These results are stored persistently in the database.

The function is of particular use when synchronizing applications on the Palm OS using HotSync, as the synchronization takes place outside the application itself. The SQLCODE value set in the connection reflect the result of the connecting operation itself. The synchronization status and results are written to the HotSync log only. To obtain extended synchronization result information, call ULGetSynchResult when connected to the database.

Examples

The following code checks for success of the previous synchronization.

```
ul_synch_result synch_result;
memset( &synch_result, 0, sizeof( ul_synch_result ) );
db_init( &sqlca );
EXEC SQL CONNECT "dba" IDENTIFIED BY "sql";
if( !ULGetSynchResult( &sqlca, &synch_result ) ) {
    prMsg( "ULGetSynchResult failed" );
}
```

ULGlobalAutoincUsage function

Obtains the percent of the default values used in all the columns having global autoincrement defaults.

Syntax

ul_u_short **ULGlobalAutoincUsage(** SQLCA * *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 function” on page 329
- “UltraLite global_database_id option” [*UltraLite - Database Management and Reference*]

 Copyright © 2007, iAnywhere Solutions, Inc.

ULGrantConnectTo function

Grants access to an UltraLite database for a new or existing user ID with the given password.

Syntax

void **ULGrantConnectTo(**
SQLCA * *sqlca*,
ul_char * *userid*,
ul_char * *password* **);**

Parameters

sqlca A pointer to the SQLCA.

userid A character array that holds the user ID. The maximum length is 16 characters.

password 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

- "Authenticating users" on page 59
- "ULRevokeConnectFrom function" on page 326

Copyright © 2007, iAnywhere Solutions, Inc.

ULHTTPSStream function (deprecated)

Defines an UltraLite HTTPS stream suitable for synchronization via HTTP.

Syntax

ul_stream_defn **ULHTTPSStream(** void **);**

Remarks

The HTTPS stream uses TCP/IP as its underlying transport. UltraLite applications act as Web browsers and MobiLink acts as a web server.

> **Deprecated feature**
> This function is deprecated in SQL Anywhere 10. You can set the stream field of ul_synch_info to a string with the desired value. This function now returns the appropriate string value.

See also

- "ULSynchronize function" on page 334
- "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*]

 Copyright © 2007, iAnywhere Solutions, Inc.

ULHTTPStream function (deprecated)

Defines an UltraLite HTTP stream suitable for synchronization via HTTP.

Syntax

ul_stream_defn **ULHTTPStream(** void **);**

Remarks

The HTTP stream uses TCP/IP as its underlying transport. UltraLite applications act as web browsers and MobiLink acts as a web server. UltraLite applications send POST requests to send data to the server and GET requests to read data from the server.

> **Deprecated feature**
> This function is deprecated in SQL Anywhere 10. You can set the stream field of ul_synch_info to a string with the desired value. This function now returns the appropriate string value.

See also

- "ULSynchronize function" on page 334
- "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*]

ULInitSynchInfo function

Initializes the synchronization information structure.

Syntax

```
void ULInitSynchInfo(
   ul_synch_info *  synch_info );
```

Parameters

synch_info A synchronization structure. For more information about the members of this structure, see "Synchronization parameters for UltraLite" [*MobiLink - Client Administration*].

 Copyright © 2007, iAnywhere Solutions, Inc.

ULIsSynchronizeMessage function

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

ul_bool **ULIsSynchronizeMessage(** ul_u_long *uMsg* **);**

Remarks

You should include function in the WindowProc function of your application.

Applies to Windows CE for ActiveSync.

Example

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

- “Adding ActiveSync synchronization to your application” on page 102
- “ActiveSync on Windows CE” [*MobiLink - Client Administration*]

ULResetLastDownloadTime function

Resets the last download time so that the application resynchronizes previously downloaded data.

Syntax

void **ULResetLastDownloadTime(**
SQLCA * *sqlca*,
ul_publication_mask *publication-mask* **);**

Parameters

sqlca A pointer to the SQLCA.

publication-mask A set of OR'd publications supplied as a mask. Use 0 to reset all publications in the entire database. For example, the following mask corresponds to publications PUB1 and PUB2.:

```
UL_PUB_PUB1 | UL_PUB_PUB2
```

Example

The following function call resets the last download time for all tables:

```
ULResetLastDownloadTime( &sqlca, UL_SYNC_ALL );
```

See also

- "PublicationMask synchronization parameter" [*MobiLink - Client Administration*]
- "ULGetLastDownloadTime function" on page 315
- "Timestamp-based downloads" [*MobiLink - Server Administration*]

 Copyright © 2007, iAnywhere Solutions, Inc.

ULRetrieveEncryptionKey function

Retrieves the encryption key from memory.

Syntax

ul_bool **ULRetrieveEncryptionKey(**
ul_char * *key*,
ul_u_short *len*,
ul_u_long * *creator***,**
ul_u_long * *feature-num* **);**

Parameters

key A pointer to a buffer in which to hold the retrieved encryption key.

len The length of the buffer that holds the encryption key with a terminating null character.

creator A pointer to the creator ID of the feature holding the encryption key. The default value is NULL.

feature-num A pointer to the feature number holding the encryption key. If feature-num is NULL, the application uses the UltraLite default value of 100.

Returns

- True if the operation is successful.
- False if the operation failed. This occurs if the feature was not found or if the supplied buffer length is insufficient to hold the key plus a terminating null character.

Remarks

On Palm OS, the encryption key is saved in dynamic memory as a Palm feature. Features are indexed by creator and a feature number.

Applies to Palm OS.

See also

- "ULClearEncryptionKey function" on page 310
- "ULSaveEncryptionKey function" on page 328

ULRevokeConnectFrom function

Revokes access from an UltraLite database for a user ID.

Syntax

void **ULRevokeConnectFrom(** SQLCA * *sqlca***,** ul_char * *userid* **);**

Parameters

sqlca A pointer to the SQLCA.

userid A character array holding the user ID to be excluded from database access. The maximum length is 16 characters.

See also

- "Authenticating users" on page 59
- "ULGrantConnectTo function" on page 319

 Copyright © 2007, iAnywhere Solutions, Inc.

ULRollbackPartialDownload function

Rolls back the changes from a failed synchronization.

Syntax

void **ULRollbackPartialDownload (** SQLCA * *sqlca* **)**

Parameters

- **sqlca** A pointer to the SQL Communications Area. In the C++ API, use the Sqlca.GetCA method.

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.

See also

- "GetCA function" on page 203
- "Resuming failed downloads" [*MobiLink - Server Administration*]
- "Keep Partial Download synchronization parameter" [*MobiLink - Client Administration*]
- "Partial Download Retained synchronization parameter" [*MobiLink - Client Administration*]
- "Resume Partial Download synchronization parameter" [*MobiLink - Client Administration*]

ULSaveEncryptionKey function

Saves the encryption key in Palm dynamic memory.

Syntax

ul_bool **ULSaveEncryptionKey(**
ul_char * *key*,
ul_u_long * *creator***,**
ul_u_long * *feature-num* **);**

Parameters

key A pointer to the encryption key.

creator A pointer to the creator ID of the feature that holds the encryption key. The default value is NULL.

feature-num A pointer to the feature number that holds the encryption key. If feature-num is NULL, the application uses the UltraLite default value of 100.

Returns

- True if the operation is successful.
- False if the operation failed. This occurs if the feature was not found or if the supplied buffer length is insufficient to hold the key plus a terminating null character.

Remarks

On the Palm OS the encryption key is saved in dynamic memory as a Palm feature. Features are indexed by creator and a feature number. They are not backed up and are cleared on any reset of the device.

Applies to Palm OS applications.

See also

- "ULClearEncryptionKey function" on page 310
- "ULRetrieveEncryptionKey function" on page 325

 Copyright © 2007, iAnywhere Solutions, Inc.

ULSetDatabaseID function

Sets the database identification number.

Syntax

void **ULSetDatabaseID(** SQLCA * *sqlca*, ul_u_long *id* **);**

Parameters

sqlca A pointer to the SQLCA.

id A positive integer that uniquely identifies a particular database in a replication or synchronization setup.

See also

- “UltraLite global_database_id option” [*UltraLite - Database Management and Reference*]
- “ULGlobalAutoincUsage function” on page 318

ULSetDatabaseOptionString function

Sets a database option from a string value.

Syntax

void **ULSetDatabaseOptionString (**SQLCA * *sqlca***,**
*ul_database_option_id***,**
ul_char * *value* **);**

Parameters

sqlca A pointer to the SQLCA.

ul_database_option_id The identifier for the database option to be set.

value The value of the database option.

 Copyright © 2007, iAnywhere Solutions, Inc.

ULSetDatabaseOptionULong

Sets a numeric database option.

Syntax

void **ULSetDatabaseOptionULong (**SQLCA * *sqlca***,**
ul_database_option_id **,**
ul_u_long * *value* **);**

Parameters

sqlca A pointer to the SQLCA.

ul_database_option_id The identifier for the database option to be set.

value The value of the database option.

ULSetSynchInfo function

Stores the synchronization parameters for use with HotSync.

Syntax

```
ul_bool ULSetSynchInfo(
   SQLCA * sqlca,
   ul_synch_info *  synch_info );
```

Parameters

sqlca A pointer to the SQLCA.

synch_info A synchronization structure. For more information about the members of this structure, see “ul_synch_info_a struct” on page 179.

Remarks

Typically, you call ULSetSynchInfo just before closing the application with db_fini.

Applies to Palm OS applications with HotSync

See also

- “db_fini function” on page 304

 Copyright © 2007, iAnywhere Solutions, Inc.

ULSocketStream function (deprecated)

Defines an UltraLite socket stream suitable for synchronization via TCP/IP.

Syntax

ul_stream_defn **ULSocketStream(** void **);**

Remarks

This function is deprecated in SQL Anywhere 10. You can set the stream field of ul_synch_info to a string with the desired value. This function now returns the appropriate string value.

See also

- "ULSynchronize function" on page 334

ULSynchronize function

Initiates synchronization in an UltraLite application.

Syntax

void **ULSynchronize(**
SQLCA * *sqlca***,**
ul_synch_info * *synch_info* **);**

Parameters

sqlca A pointer to the SQLCA.

synch_info A synchronization structure. Synchronization specifics are controlled through a set of synchronization parameters. For more information about these parameters, see “ul_synch_info_a struct” on page 179.

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. Your application should test the SQLCODE return value of this function.

 Copyright © 2007, iAnywhere Solutions, Inc.

CHAPTER 14

UltraLite ODBC API Reference

Contents

Introduction to UltraLite ODBC API 336
SQLAllocHandle function 337
SQLBindCol function 338
SQLBindParameter function 339
SQLConnect function 340
SQLDescribeCol function 341
SQLDisconnect function 342
SQLEndTran function 343
SQLExecDirect function 344
SQLExecute function 345
SQLFetch function 346
SQLFetchScroll function 347
SQLFreeHandle function 348
SQLGetCursorName function 349
SQLGetData function 350
SQLGetDiagRec function 351
SQLGetInfo function 352
SQLNumResultCols function 353
SQLPrepare function 354
SQLRowCount function 355
SQLSetConnectionName function 356
SQLSetCursorName function 357
SQLSetSuspend function 358
SQLSynchronize function 359

Introduction to UltraLite ODBC API

This chapter describes the components of the ODBC interface supported by UltraLite.

This is not a comprehensive ODBC reference; it is intended as a quick reference to complement the main reference for ODBC, which is the Microsoft ODBC SDK documentation.

 Copyright © 2007, iAnywhere Solutions, Inc.

SQLAllocHandle function

Allocates a handle, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLAllocHandle(**
SQLSMALLINT *HandleType*,
SQLHANDLE *InputHandle*,
SQLHANDLE * *OutputHandle* **)**;

Parameters

- **HandleType** The type of handle to be allocated. UltraLite supports the following handle types:
  - SQL_HANDLE_ENV (environment handle)
  - SQL_HANDLE_DBC (connection handle)
  - SQL_HANDLE_STMT (statement handle)
- **InputHandle** The handle in whose context the new handle is to be allocated. For a connection handle, this is the environment handle; for a statement handle, this is the connection handle.
- **OutputHandle** A pointer to a buffer in which to return the new handle.

Remarks

ODBC uses handles to provide the context for database operations. An environment handle provides the context for communication with a data source, like the SQL Communications Area in other interfaces. A connection handle provides a context for all database operations. A statement handle manages result sets and data modification. A descriptor handle manages the handling of result set data types.

See also

- SQLAllocHandle in the Microsoft *ODBC Programmer's Reference*.

SQLBindCol function

Binds a result set column to an application data buffer, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLBindCol (**
SQLHSTMT *StatementHandle*,
SQLUSMALLINT *ColumnNumber*,
SQLSMALLINT *TargetType*,
SQLPOINTER *TargetValue*,
SQLLEN *BufferLength*,
SQLLEN * *StrLen_or_Ind* **)**;

Parameters

- **StatementHandle** A handle for the statement that is to return a result set.
- **ColumnNumber** The number of the column in the result set to bind to an application data buffer.
- **TargetType** The identifier of the data type of the *TargetValue* pointer.
- **TargetValue** A pointer to the data buffer to bind to the column.
- **BufferLength** The length of the *TargetValue* buffer in bytes.
- **StrLen_or_Ind** A pointer to the length or indicator buffer to bind to the column. For strings, the length buffer holds the length of the actual string that was returned, which may be less than the length allowed by the column.

Remarks

To exchange information between your application and the database, ODBC binds buffers in the application to database objects such as columns. SQLBindCol is used when executing a query to identify a buffer in your application as a place that UltraLite puts the value of a specified column.

See also

- SQLBindCol in the Microsoft *ODBC Programmer's Reference*.

 Copyright © 2007, iAnywhere Solutions, Inc.

SQLBindParameter function

Binds a buffer parameter to a parameter marker in a SQL statement, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLBindParameter (**
SQLHSTMT *StatementHandle*,
SQLUSMALLINT *ParameterNumber*,
SQLSMALLINT *ParamType*,
SQLSMALLINT *CType*,
SQLSMALLINT *SqlType*,
SQLULEN *ColDef*,
SQLSMALLINT *Scale*,
SQLPOINTER *rgbValue*,
SQLLEN *cbValueMax*,
SQLLEN * *StrLen_or_Ind* **)**;

Parameters

- **StatementHandle** A statement handle.
- **ParameterNumber** The number of the parameter marker in the statement, in sequential order counting from 1.
- **ParamType** The parameter type. One of the following:
  - SQL_PARAM_INPUT
  - SQL_PARAM_INPUT_OUTPUT
  - SQL_PARAM_OUTPUT
- **CType** The C data type of the parameter.
- **SQLType** The SQL data type of the parameter.
- **ColDef** The size of the column or expression of the parameter marker.
- **Scale** The number of decimal digits for the column or expression of the parameter marker.
- **rgbValue** A pointer to a buffer for the parameter's data.
- **cbValueMax** The length of the *rgbValue*rgbValue buffer.
- **StrLen_or_Ind** A pointer to a buffer for the parameter's length.

Remarks

To exchange information between your application and the database, ODBC binds buffers in the application to database objects such as columns. SQLBindParameter is used when executing a statement, to identify a buffer in your application as a place that UltraLite gets or sets the value of a specified parameter in a query.

See also

- SQLBindParameter in the Microsoft *ODBC Programmer's Reference*.

SQLConnect function

Connects to a database, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLConnect (**
SQLHDBC *ConnectionHandle*,
SQLTCHAR * *ServerName*,
SQLSMALLINT *NameLength1*,
SQLTCHAR * *UserName*,
SQLSMALLINT *NameLength2*,
SQLTCHAR * *Authentication*,
SQLSMALLINT *NameLength3* **)**;

Parameters

- **ConnectionHandle** The connection handle.
- **ServerName** A connection string that defines the database to which your application connects. UltraLite ODBC does not use ODBC data sources. Instead, supply a connection string containing the database connection parameters, together with optional other parameters.

  The following is an example of a ServerName parameter:

  ```
  (SQLTCHAR*)UL_TEXT(
      "dbf=customer.udb"
  )
  ```

  For a complete list of connection parameters, see "UltraLite Connection String Parameters Reference" [*UltraLite - Database Management and Reference*].
- **NameLength1** The length of **ServerName*.
- **UserName** The user ID to use when connecting. The user ID can alternatively be specified in the connection string supplied to the ServerName parameter.
- **NameLength2** The length of **UserName*.
- **Authentication** The password to use when connecting.The password can alternatively be specified in the connection string supplied to the ServerName parameter.
- **NameLength3** The length of **Authentication*.

Remarks

Connects to a database. For more information about UltraLite connection parameters, see "UltraLite Connection String Parameters Reference" [*UltraLite - Database Management and Reference*].

See also

- SQLConnect in the Microsoft *ODBC Programmer's Reference*.

 Copyright © 2007, iAnywhere Solutions, Inc.

SQLDescribeCol function

Returns the result descriptor for a column in the result set, for UltraLite ODBC.

The result descriptor includes the column name, column size, data type, number of decimal digits, and nullability.

Syntax

```
UL_FN_SPEC SQLRETURN UL_FN_MOD SQLDescribeCol (
   SQLHSTMT StatementHandle,
   SQLUSMALLINT ColumnNumber,
   SQLTCHAR * ColumnName,
   SQLSMALLINT BufferLength,
   SQLSMALLINT * NameLength,
   SQLSMALLINT * DataType,
   SQLULEN * ColumnSize,
   SQLSMALLINT * DecimalDigits,
   SQLSMALLINT * Nullable );
```

Parameters

- **StatementHandle** A statement handle.
- **ColumnNumber** The 1-based column number of result data.
- **ColumnName** A pointer to a buffer in which to return the column name.
- **BufferLength** The length of **ColumnName*, in characters.
- **NameLength** A pointer to a buffer in which to return the total number of bytes (excluding the null-termination byte) available to return in **ColumnName*.
- **DataType** A pointer to a buffer in which to return the SQL data type of the column.
- **ColumnSize** A pointer to a buffer in which to return the size of the column on the data source.
- **DecimalDigits** A pointer to a buffer in which to return the number of decimal digits of the column on the data source.
- **Nullable** A pointer to a buffer in which to return a value that indicates whether the column allows NULL values.

See also

- SQLDescribeCol in the Microsoft *ODBC Programmer's Reference*

SQLDisconnect function

Disconnects the application from a database, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLDisconnect (**
SQLHDBC *ConnectionHandle* **)**;

Parameters

- **ConnectionHandle** The handle for the connection to be closed.

Remarks

Once SQLDisconnect is called, no further operations can be carried out against the database without opening a new connection.

See also

- "SQLConnect function" on page 340
- SQLDisconnect in the Microsoft *ODBC Programmer's Reference*

Copyright © 2007, iAnywhere Solutions, Inc.

SQLEndTran function

Commits or rolls back a transaction, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLEndTran (**
SQLSMALLINT *HandleType*,
SQLHANDLE *Handle*,
SQLSMALLINT *CompletionType* **)**;

Parameters

- **HandleType** The type of handle to be allocated. UltraLite supports the following handle types:
  - SQL_HANDLE_ENV
  - SQL_HANDLE_DBC
  - SQL_HANDLE_STMT
- **Handle** The connection handle indicating the scope of the transaction.
- **CompletionType** One of the following two values:
  - SQL_COMMIT
  - SQL_ROLLBACK

See also

- SQLEndTran in the Microsoft *ODBC Programmer's Reference*

SQLExecDirect function

Executes a SQL statement, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLExecDirect (**
SQLHSTMT *StatementHandle*,
SQLTCHAR * *StatementText*,
SQLINTEGER *TextLength* **)**;

Parameters

- **StatementHandle** A statement handle.
- **StatementText** The text of the SQL statement.
- **TextLength** The length of **StatementText*.

Remarks

Unlike SQLExecute, the statement does not need to be prepared before being executed using SQLExecDirect.

SQLExecDirect has slower performance than SQLExecute for statements executed repeatedly.

See also

- SQLExecDirect in the Microsoft *ODBC Programmer's Reference*
- "SQLExecute function" on page 345

 Copyright © 2007, iAnywhere Solutions, Inc.

SQLExecute function

Executes a prepared SQL statement, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLExecute (**
SQLHSTMT *StatementHandle* **)**;

Parameters

- **StatementHandle** The handle for the statement to be executed.

Remarks

The statement must be prepared using SQLPrepare before it can be executed. If the statement has parameter markers, they must be bound to variables using SQLBindParameter before execution.

You can use SQLExecDirect to execute a statement without preparing it first. SQLExecDirect has slower performance than SQLExecute for statements executed repeatedly.

See also

- "SQLBindParameter function" on page 339
- "SQLPrepare function" on page 354
- SQLExecute in the Microsoft *ODBC Programmer's Reference*.
- "SQLExecDirect function" on page 344

SQLFetch function

Fetches the next row from a result set and returns data for all bound columns, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLFetch (**
SQLHSTMT *StatementHandle* **)**;

Parameters

- **StatementHandle** A statement handle.

Remarks

Before fetching rows, you must have bound the columns in the result set to buffers using SQLBindCol. To fetch a row other than the next row in the result set, use SQLFetchScroll.

See also

- "SQLFetchScroll function" on page 347
- "SQLBindCol function" on page 338

- SQLFetch in the Microsoft *ODBC Programmer's Reference*.

 Copyright © 2007, iAnywhere Solutions, Inc.

SQLFetchScroll function

Fetches the specified row from the result set and returns data for all bound columns, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLFetchScroll (**
SQLHSTMT *StatementHandle*,
SQLSMALLINT *FetchOrientation*,
SQLLEN *FetchOffset* **)**;

Parameters

- **StatementHandle** A statement handle.
- **FetchOrientation** The type of fetch.
- **FetchOffset** The number of the row to fetch. The interpretation depends on the value of *FetchOrientation*.

Remarks

Before fetching rows, you must have bound the columns in the result set to buffers using SQLBindCol. SQLFetchScroll is for use in those cases where the more straightforward SQLFetch is not appropriate.

See also

- "SQLFetch function" on page 346
- "SQLBindCol function" on page 338
- SQLFetchScroll in the Microsoft *ODBC Programmer's Reference*.

SQLFreeHandle function

Frees resources for a handle allocated, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLFreeHandle (**
SQLSMALLINT *HandleType*,
SQLHANDLE *Handle* **)**;

Parameters

- **HandleType** The type of handle to be allocated. UltraLite supports the following handle types:
  - SQL_HANDLE_ENV
  - SQL_HANDLE_DBC
  - SQL_HANDLE_STMT
- **Handle** The handle to be freed.

Remarks

SQLFreeHandle should be called for each handle allocated using SQLAllocHandle, when the handle is no longer needed.

See also

- "SQLAllocHandle function" on page 337
- SQLFreeHandle in the Microsoft *ODBC Programmer's Reference*.

Copyright © 2007, iAnywhere Solutions, Inc.

SQLGetCursorName function

Returns the name associated with a cursor for a specified statement, for UltraLite ODBC.

Syntax

```
UL_FN_SPEC SQLRETURN UL_FN_MOD SQLGetCursorName (
   SQLHSTMT StatementHandle,
   SQLTCHAR * CursorName,
   SQLSMALLINT BufferLength,
   SQLSMALLINT * NameLength );
```

Parameters

- **StatementHandle** A statement handle.
- **CursorName** A pointer to a buffer in which to return the name of the cursor associated with *StatementHandle*.
- **BufferLength** The length of **CursorName*.
- **NameLength** A pointer to memory in which to return the total number of bytes (excluding the null-termination character) available to return in **CursorName*.

See also

- "SQLSetCursorName function" on page 357
- SQLGetCursorName in the Microsoft *ODBC Programmer's Reference*.

SQLGetData function

Retrieves data for a single column in the result set, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLGetData (**
SQLHSTMT *StatementHandle*,
SQLUSMALLINT *ColumnNumber*,
SQLSMALLINT *TargetType*,
SQLPOINTER *TargetValue*,
SQLLEN *BufferLength*,
SQLLEN * *StrLen_or_Ind* **)**;

Parameters

- **StatementHandle** A statement handle.
- **ColumnNumber** The number of the column in the result set to bind.
- **TargetType** The output handle.
- **TargetValue** A pointer to the data buffer to bind to the column.
- **BufferLength** The length of the *TargetValue* buffer in bytes.
- **StrLen_or_Ind** A pointer to the length or indicator buffer to bind to the column.

Remarks

SQLGetData is typically used to retrieve variable-length data in parts.

See also

- SQLGetData in the Microsoft *ODBC Programmer's Reference*.

 Copyright © 2007, iAnywhere Solutions, Inc.

SQLGetDiagRec function

Returns the current values of multiple fields of a diagnostic status record, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLGetDiagRec (**
SQLSMALLINT *HandleType*,
SQLHANDLE *Handle*,
SQLSMALLINT *RecNumber*,
SQLTCHAR * *Sqlstate*,
SQLINTEGER * *NativeError*,
SQLTCHAR * *MessageText*,
SQLSMALLINT *BufferLength*,
SQLSMALLINT * *TextLength* **)**;

Parameters

- **HandleType** The type of handle to be allocated. UltraLite supports the following handle types:
  - SQL_HANDLE_ENV
  - SQL_HANDLE_DBC
  - SQL_HANDLE_STMT
- **Handle** The input handle
- **RecNumber** The output handle.
- **Sqlstate** The ANSI/ISO SQLSTATE value of the error. For a listing, see “Error messages sorted by SQLSTATE” [*SQL Anywhere 10 - Error Messages*].
- **NativeError** The SQLCODE value of the error. For a listing, see “Error messages sorted by SQL Anywhere SQLCODE” [*SQL Anywhere 10 - Error Messages*].
- **MessageText** The text of the error or status message.
- **BufferLength** The length of the **MessageText* buffer in bytes.
- **TextLength** A pointer to a buffer in which to return the total number of bytes (excluding the null-termination byte) available to return in **MessageText*.

See also

- SQLGetDiagRec in the Microsoft *ODBC Programmer's Reference*

SQLGetInfo function

Returns general information about the current ODBC driver and data source, for UltraLite ODBC.

Syntax

```
UL_FN_SPEC SQLRETURN UL_FN_MOD SQLGetInfo (
SQLHDBC ConnectionHandle,
SQLUSMALLINT InfoType,
SQLPOINTER * InfoValue,
SQLSMALLINT BufferLength,
SQLSMALLINT ODBCFAR * StringLength );
```

Parameters

- **ConnectionHandle** A connection handle.
- **InfoType** The type of information returned. The only type supported is SQL_DBMS_VER. The information returned is a character string identifying the current release of the software.
- **InfoValue** A pointer to a buffer in which to return the information.
- **BufferLength** The length of the **InfoValue* buffer in bytes.
- **StringLength** A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character for character data) available to return in **InfoValue*.

See also

- SQLGetInfo in the Microsoft *ODBC Programmer's Reference*.

Copyright © 2007, iAnywhere Solutions, Inc.

SQLNumResultCols function

Returns the number of columns in a result set, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLNumResultCols (**
SQLHSTMT *StatementHandle*,
SQLSMALLINT * *ColumnCount* **)**;

Parameters

- **StatementHandle** A statement handle.
- **ColumnCount** A pointer to a buffer in which to return the total number of columns in the result set.

See also

- SQLNumResultCols in the Microsoft *ODBC Programmer's Reference*.

SQLPrepare function

Prepares a SQL statement for execution, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLPrepare (**
SQLHSTMT *StatementHandle*,
SQLTCHAR * *StatementText*,
SQLINTEGER *TextLength* **)**;

Parameters

- **StatementHandle** A statement handle.
- **StatementText** A pointer to a buffer that holds the SQL statement text.
- **TextLength** The length of **StatementText*.

See also

- "SQLExecute function" on page 345
- SQLPrepare in the Microsoft *ODBC Programmer's Reference*.

Copyright © 2007, iAnywhere Solutions, Inc.

SQLRowCount function

Returns the number of rows affected by an INSERT, UPDATE, or DELETE operation, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLRowCount (**
SQLHSTMT *StatementHandle*,
SQLLEN * *RowCount* **)**;

Parameters

- **StatementHandle** A statement handle.
- **RowCount** A pointer to a buffer in which the number of rows is returned.

See also

- SQLRowCount in the Microsoft *ODBC Programmer's Reference*.

SQLSetConnectionName function

Sets a connection name for the suspend and restore operation, for UltraLite ODBC. This function is specific to UltraLite and is not part of the ODBC standard.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLSetConnectionName (**
SQLHSTMT *StatementHandle*,
SQLTCHAR * *ConnectionName*,
SQLSMALLINT *NameLength* **)**;

Parameters

- **StatementHandle** A statement handle.
- **ConnectionName** A pointer to a buffer holding the connection name.
- **NameLength** The length of **ConnectionName*

Remarks

SQLSetConnectionName is used to provide a connection name for use in the suspend and restore operation, together with SQLSetSuspend. Set the connection name before opening a connection to restore application state.

See also

- "Maintaining state in UltraLite Palm applications" on page 79
- "SQLSetSuspend function" on page 358

 Copyright © 2007, iAnywhere Solutions, Inc.

SQLSetCursorName function

Sets the name of a cursor associated with a SQL statement, for UltraLite ODBC.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLSetCursorName (**
SQLHSTMT *StatementHandle*,
SQLTCHAR * *CursorName*,
SQLSMALLINT *NameLength* **)**;

Parameters

- **StatementHandle** A statement handle.
- **CursorName** A pointer to a buffer holding the cursor name.
- **NameLength** The length of **CursorName*.

See also

- "SQLGetCursorName function" on page 349
- SQLSetCursorName in the Microsoft *ODBC Programmer's Reference*.

Copyright © 2007, iAnywhere Solutions, Inc.

SQLSetSuspend function

Indicates whether the state of open cursors should be saved on closing the application, for UltraLite ODBC. This function is specific to UltraLite and is not part of the ODBC standard.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLSetSuspend (**
SQLSMALLINT *HandleType*,
SQLHSTMT *StatementHandle*,
SQLSMALLINT *TrueFalse* **)**;

Parameters

- **HandleType** The type of handle to be allocated. UltraLite supports the following handle types:

  - SQL_HANDLE_ENV
  - SQL_HANDLE_DBC
  - SQL_HANDLE_STMT

- **StatementHandle** A statement handle.

- **TrueFalse** The output handle.

See also

- "Maintaining state in UltraLite Palm applications" on page 79

Copyright © 2007, iAnywhere Solutions, Inc.

SQLSynchronize function

Synchronizes data in the database using MobiLink synchronization, for UltraLite ODBC. This function is specific to UltraLite and is not part of the ODBC standard.

Syntax

UL_FN_SPEC SQLRETURN UL_FN_MOD **SQLSynchronize (**
SQLHDBC *ConnectionHandle*,
ul_synch_info * *SynchInfo* **)**;

Parameters

- ♦ **ConnectionHandle** A handle.
- ♦ **SynchInfo** The structure holding synchronization information. See "UltraLite Synchronization Parameters and Network Protocol options" [*MobiLink - Client Administration*].

Remarks

SQLSynchronize is an extension to ODBC. It initiates a MobiLink synchronization operation.

See also

- ♦ "UltraLite Synchronization Parameters and Network Protocol options" [*MobiLink - Client Administration*].
- ♦ MobiLink - Server Administration [*MobiLink - Server Administration*]

Index

Symbols

#define
  UltraLite applications, 173
16-bit signed integer UltraLite embedded SQL data type
  about, 46
32-bit signed integer UltraLite embedded SQL data type
  about, 46
4-byte floating point UltraLite embedded SQL data type
  about, 46
8-byte floating point UltraLite embedded SQL data type
  about, 46
~ULSqlca function [UL C++]
  UltraLite C++ Component API, 200
~ULSqlcaWrap function [UL C++]
  UltraLite C++ Component API, 208
~ULValue function [UL C++]
  UltraLite C++ Component API, 298

A

accessing data
  UltraLite C++ Table API, 23
ActiveSync
  class names, 99
  ULIsSynchronizeMessage function, 323
  UltraLite message, 173
  UltraLite MFC requirements, 103
  UltraLite synchronization for Windows CE, 102
  UltraLite versions for Windows CE, 102
  UltraLite Windows CE applications, 102
  WindowProc function, 102
AddRef function [UL C++]
  UltraLite C++ Component API, 258
AES encryption algorithm
  UltraLite embedded SQL databases, 61
AfterLast function [UL C++]
  UltraLite C++ Component API, 223
ANSI
  UltraLite C++ library for, 36
APIs
  UltraLite Table API, 23
applications
  building the sample UltraLite embedded SQL application, 132
  building UltraLite embedded SQL, 70
  compiling UltraLite embedded SQL, 70
  deploying UltraLite on Palm OS, 85
  preprocessing UltraLite embedded SQL, 70
  writing in UltraLite embedded SQL, 126
  writing UltraLite embedded SQL, 37
auth_parms variable [UL C++]
  UltraLite C++ Component API, 180, 187
auth_status variable [UL C++]
  UltraLite C++ Component API, 180, 187, 193
auth_value variable [UL C++]
  UltraLite C++ Component API, 180, 187, 193
AutoCommit mode
  UltraLite C++ development, 29

B

BeforeFirst function [UL C++]
  UltraLite C++ Component API, 224
binary UltraLite embedded SQL data type
  about, 47
bugs
  providing feedback, xvii
build processes
  embedded SQL applications, 70
  UltraLite embedded SQL applications, 70
building
  sample UltraLite embedded SQL application, 132
  UltraLite embedded SQL applications, 70
bytes variable [UL C++]
  UltraLite C++ Component API, 196

C

C++ API (see UltraLite C/C++ API)
C++ applications, ix
  (see also UltraLite C/C++)
casting
  UltraLite C++ API data types in, 25
ChangeEncryptionKey function [UL C++]
  UltraLite C++ Component API, 211
changeEncryptionKey method
  UltraLite embedded SQL, 61
character string UltraLite embedded SQL data type
  fixed length, 47
  variable length, 47
Checkpoint function [UL C++]
  UltraLite C++ Component API, 211
checkpoint_store variable [UL C++]

Copyright © 2007, iAnywhere Solutions, Inc.

UltraLite C++ Component API, 180, 187
class names
ActiveSync synchronization, 99
ClientParms registry entry
MobiLink conduit, 83
CLOSE statement
UltraLite embedded SQL, 56
CodeWarrior
converting projects, 76
creating UltraLite projects, 75
expanded mode UltraLite applications, 78
installing UltraLite plug-in, 75
stationery for UltraLite, 75
UltraLite C/C++ development, 75
using UltraLite plug-in, 77
columns
UltraLite C++ API modification of values , 25
UltraLite C++ API retrieval of values , 25
Commit function [UL C++]
UltraLite C++ Component API, 211
commit method
UltraLite C++ transactions, 29
commiting
UltraLite changes with embedded SQL, 65
committing
UltraLite C++ transactions, 29
communications errors
UltraLite embedded SQL, 65
compile options
UltraLite applications for Windows CE, 95
compiler directives
UltraLite applications, 173
UNDER_CE, 174
UNDER_PALM_OS, 174
compiler options
UltraLite C++ development, 35
UltraLite compiler for ODBC, 137
compilers
Palm OS, 74
UltraLite applications for Windows CE, 95
compiling
UltraLite applications for Windows CE, 95
UltraLite embedded SQL applications, 70
configuring
development tools for UltraLite embedded SQL, 71
CONNECT statement
UltraLite embedded SQL, 43
connecting
UltraLite C++ databases, 17
Connection object
UltraLite C++, 17
connections
SQLCAs in UltraLite C/C++, 12
UltraLite embedded SQL, 43
conventions
documentation, xii
file names in documentation, xiv
CountUploadRows function [UL C++]
UltraLite C++ Component API, 211
CreateDatabase function [UL C++]
UltraLite C++ Component API, 231
Creating UltraLite databases
database creation parameters, 154
ULGetCollation_ function (UltraLite C/C++), 154
creator IDs
about, 81
Palm OS applications, 81
cursors
UltraLite fetching multiple rows, 56
UltraLite order of rows, 57
UltraLite positioning, 57
UltraLite positioning after updates, 58
UltraLite repositioning, 58
UltraLite restoring state, 79
UltraLite saving state, 79
CustDB application
UltraLite building for Palm OS, 77
UltraLite building for Windows CE, 96

D

data manipulation
UltraLite C++ API with SQL, 19
UltraLite C++ Table API, 23
data types
UltraLite C++ API accessing in , 24
UltraLite C++ API conversion of, 25
UltraLite embedded SQL, 45
database files
UltraLite encrypting and obfuscating (embedded SQL), 61
UltraLite for Windows CE, 98
database schemas
UltraLite C++ API access, 30
DatabaseManager object
UltraLite C++, 17

DatabaseSchema object
UltraLite C++, 30
db_fini function [UL ESQL]
do not use on the Palm Computing Platform, 304
syntax, 304
db_init function [UL ESQL]
syntax, 305
db_start_database function [UL ESQL]
syntax, 306
db_stop_database function [UL ESQL]
syntax, 307
decimal UltraLite embedded SQL data type
about, 46
DECL_BINARY macro
UltraLite embedded SQL, 45
DECL_DATETIME macro
UltraLite embedded SQL, 45
DECL_DECIMAL macro
UltraLite embedded SQL, 45
DECL_FIXCHAR macro
UltraLite embedded SQL, 45
DECL_VARCHAR macro
UltraLite embedded SQL, 45
declaration section
UltraLite embedded SQL declaration of, 45
DECLARE statement
UltraLite embedded SQL, 56
declaring
UltraLite host variables, 45
Delete function [UL C++]
UltraLite C++ Component API, 224
DeleteAllRows function [UL C++]
UltraLite C++ Component API, 266
DeleteNamed function [UL C++]
UltraLite C++ Component API, 251
deletes variable [UL C++]
UltraLite C++ Component API, 196
deleting
UltraLite C++ API table rows, 28
dependencies
UltraLite embedded SQL, 71
deploying
UltraLite applications on Palm OS, 85
UltraLite for Windows CE, 99
UltraLite on Palm OS, 85
UltraLite to Palm OS, 85
descUltraLite ODBC interface
SQLDisconnect function, 342
developer community
newsgroups, xvii
development
UltraLite C++, 15
development platforms
UltraLite C++, 6
development process
UltraLite embedded SQL, 4
development tools
UltraLite embedded SQL, 71
directives
UltraLite applications, 173
disable_concurrency variable [UL C++]
UltraLite C++ Component API, 180, 187
DML
UltraLite C++, 19
documentation
conventions, xii
SQL Anywhere, x
download_only variable [UL C++]
UltraLite C++ Component API, 181, 187
DropDatabase function [UL C++]
UltraLite C++ Component API, 232
DT_BINARY UltraLite embedded SQL data type
about, 48
DT_LONGVARCHAR UltraLite embedded SQL data type
about, 48
dynamic libraries
UltraLite C++ applications, 35

E

embedded SQL, ix
(see also UltraLite embedded SQL)
embedded SQL library functions (see UltraLite embedded SQL library functions)
eMbedded Visual C++
UltraLite development requirement for Windows CE, 94
emulator
UltraLite for Windows CE, 99
encryption
changing keys in UltraLite embedded SQL, 61
changing UltraLite encryption keys (embedded SQL), 308
Palm OS, 80

storing the encryption key in UltraLite embedded SQL, 80
ULEnableStrongEncryption function in UltraLite C/C++, 164
UltraLite C++ development, 33
UltraLite databases using embedded SQL, 61
UltraLite embedded SQL databases, 61
error checking
UltraLite ODBC interface, 351
error handling
Callback function syntax (UltraLite C/C++), 149
ULRegisterErrorCallback (UltraLite C/C++), 171
UltraLite C++, 31
errors
communications errors in UltraLite embedded SQL, 65
UltraLite C++ API handling , 31
UltraLite codes, 41
UltraLite SQLCODE, 41
UltraLite sqlcode SQLCA field, 41
EXEC SQL
UltraLite embedded SQL development, 40
execute function
UltraLite ODBC, 143
ExecuteQuery function [UL C++]
UltraLite C++ Component API, 246
ExecuteStatement function [UL C++]
UltraLite C++ Component API, 246
expanded mode
Palm OS UltraLite applications, 78

F

feedback
documentation, xvii
providing, xvii
FETCH statement
UltraLite embedded SQL multi-row queries, 56
UltraLite embedded SQL single-row queries, 55
fetching
UltraLite embedded SQL, 55
Finalize function [UL C++]
UltraLite C++ Component API, 202
Find function [UL C++]
UltraLite C++ Component API, 267
find methods
UltraLite C++, 25
find mode
UltraLite C++, 24
FindBegin function [UL C++]
UltraLite C++ Component API, 267
FindFirst function [UL C++]
UltraLite C++ Component API, 267
finding out more and providing feed back
technical support, xvii
FindLast function [UL C++]
UltraLite C++ Component API, 268
FindNext function [UL C++]
UltraLite C++ Component API, 268
FindPrevious function [UL C++]
UltraLite C++ Component API, 268
First function [UL C++]
UltraLite C++ Component API, 224
flags variable [UL C++]
UltraLite C++ Component API, 198
functions
UltraLite embedded SQL, 302

G

generated databases
naming in UltraLite, 77
Get function [UL C++]
UltraLite C++ Component API, 224
GetBaseColumnName function [UL C++]
UltraLite C++ Component API, 253
GetBinary function [UL C++]
UltraLite C++ Component API, 284, 285
GetBinaryLength function [UL C++]
UltraLite C++ Component API, 285
GetByteChunk function [UL C++]
UltraLite C++ Component API, 261
GetCA function [UL C++]
UltraLite C++ Component API, 203
GetCollationName function [UL C++]
UltraLite C++ Component API, 235
GetColumnCount function [UL C++]
UltraLite C++ Component API, 240, 254
GetColumnDefault function [UL C++]
UltraLite C++ Component API, 273
GetColumnID function [UL C++]
UltraLite C++ Component API, 254
GetColumnName function [UL C++]
UltraLite C++ Component API, 240, 254
GetColumnPrecision function [UL C++]
UltraLite C++ Component API, 255

 Copyright © 2007, iAnywhere Solutions, Inc.

GetColumnScale function [UL C++]
UltraLite C++ Component API, 256
GetColumnSize function [UL C++]
UltraLite C++ Component API, 256
GetColumnSQLName function [UL C++]
UltraLite C++ Component API, 255
GetColumnSQLType function [UL C++]
UltraLite C++ Component API, 256
GetColumnType function [UL C++]
UltraLite C++ Component API, 257
GetCombinedStringItem function [UL C++]
UltraLite C++ Component API, 286
GetConnection function [UL C++]
UltraLite C++ Component API, 258
GetConnectionNum function [UL C++]
UltraLite C++ Component API, 212
GetDatabaseID function [UL C++]
UltraLite C++ Component API, 212
GetDatabaseProperty function [UL C++]
UltraLite C++ Component API, 212
GetGlobalAutoincPartitionSize function [UL C++]
UltraLite C++ Component API, 274
GetID function [UL C++]
UltraLite C++ Component API, 241, 274
GetIFace function [UL C++]
UltraLite C++ Component API, 259
GetIndexCount function [UL C++]
UltraLite C++ Component API, 275
GetIndexName function [UL C++]
UltraLite C++ Component API, 275
GetIndexSchema function [UL C++]
UltraLite C++ Component API, 275
GetLastDownloadTime function [UL C++]
UltraLite C++ Component API, 212
GetLastIdentity function [UL C++]
UltraLite C++ Component API, 213
GetLength function [UL C++]
UltraLite C++ Component API, 262
GetName function [UL C++]
UltraLite C++ Component API, 241, 276
GetNewUUID function [UL C++]
UltraLite C++ Component API, 213
GetOptimalIndex function [UL C++]
UltraLite C++ Component API, 276
GetParameter function [UL C++]
UltraLite C++ Component API, 203
GetParameterCount function [UL C++]
UltraLite C++ Component API, 204
GetPlan function [UL C++]
UltraLite C++ Component API, 247
GetPrimaryKey function [UL C++]
UltraLite C++ Component API, 276
GetPublicationCount function [UL C++]
UltraLite C++ Component API, 235
GetPublicationID function [UL C++]
UltraLite C++ Component API, 236
GetPublicationMask function [UL C++]
UltraLite C++ Component API, 213, 236
GetPublicationName function [UL C++]
UltraLite C++ Component API, 236
GetPublicationPredicate function [UL C++]
UltraLite C++ Component API, 277
GetReferencedIndexName function [UL C++]
UltraLite C++ Component API, 241
GetReferencedTableName function [UL C++]
UltraLite C++ Component API, 242
GetRowCount function [UL C++]
UltraLite C++ Component API, 224
GetSchema function [UL C++]
UltraLite C++ Component API, 214, 247, 251, 269
GetSqlca function [UL C++]
UltraLite C++ Component API, 214
GetSQLCode function [UL C++]
UltraLite C++ Component API, 204
GetSQLCount function [UL C++]
UltraLite C++ Component API, 205
GetSQLErrorOffset function [UL C++]
UltraLite C++ Component API, 205
GetState function [UL C++]
UltraLite C++ Component API, 225
GetStreamReader function [UL C++]
UltraLite C++ Component API, 225
GetStreamWriter function [UL C++]
UltraLite C++ Component API, 225, 248
GetString function [UL C++]
UltraLite C++ Component API, 286, 287
GetStringChunk function [UL C++]
UltraLite C++ Component API, 262
GetStringLength function [UL C++]
UltraLite C++ Component API, 287
GetSuspend function [UL C++]
UltraLite C++ Component API, 214, 226
GetSynchResult function [UL C++]
UltraLite C++ Component API, 214
GetTableCount function [UL C++]
UltraLite C++ Component API, 237

GetTableName function [UL C++]
  UltraLite C++ Component API, 237, 242
GetTableSchema function [UL C++]
  UltraLite C++ Component API, 238
getting help
  technical support, xvii
GetUploadUnchangedRows function [UL C++]
  UltraLite C++ Component API, 277
GetUtilityULValue function [UL C++]
  UltraLite C++ Component API, 215
global autoincrement
  ULGlobalAutoincUsage function, 318
  ULSetDatabaseID function (UltraLite embedded SQL), 329
global database identifier
  UltraLite embedded SQL, 329
GlobalAutoincUsage function [UL C++]
  UltraLite C++ Component API, 215
GrantConnectTo function [UL C++]
  UltraLite C++ Component API, 215
grantConnectTo method
  UltraLite C++ development, 32

H

handling errors
  Callback function syntax (UltraLite C/C++), 149
  ULRegisterErrorCallback (UltraLite C/C++), 171
HasResultSet function [UL C++]
  UltraLite C++ Component API, 248
help
  technical support, xvii
host variables
  UltraLite embedded SQL, 45
  UltraLite embedded SQL expressions, 50
  UltraLite scope, 49
  UltraLite usage, 48
HotSync synchronization
  Palm OS, 82
HTTP synchronization
  UltraLite for Palm OS, 84
HTTPS synchronization
  UltraLite for Palm OS, 84

I

iAnywhere developer community
  newsgroups, xvii
icons
  used in manuals, xv
ignored_rows variable [UL C++]
  UltraLite C++ Component API, 181, 188, 193
import libraries
  UltraLite C++, 35
INCLUDE statement
  UltraLite SQLCA, 41
InDatabase function [UL C++]
  UltraLite C++ Component API, 288
indexes
  UltraLite C++ API schema information in , 30
IndexSchema object
  UltraLite C++ development, 30
indicator variables
  UltraLite embedded SQL, 53
  UltraLite NULL, 53
info variable [UL C++]
  UltraLite C++ Component API, 198
init_verify variable [UL C++]
  UltraLite C++ Component API, 181, 188
Initialize function [UL C++]
  UltraLite C++ Component API, 205
InitSynchInfo function [UL C++]
  UltraLite C++ Component API, 215, 216
InPublication function [UL C++]
  UltraLite C++ Component API, 277
Insert function [UL C++]
  UltraLite C++ Component API, 269
insert mode
  UltraLite C++, 24
InsertBegin function [UL C++]
  UltraLite C++ Component API, 269
inserting
  UltraLite C++ API table rows, 27
inserts variable [UL C++]
  UltraLite C++ Component API, 196
install-dir
  documentation usage, xiv
installing
  Palm OS UltraLite, 85
  UltraLite plug-in for CodeWarrior, 75
  UltraLite Windows CE applications, 94
IsCaseSensitive function [UL C++]
  UltraLite C++ Component API, 238
IsColumnAutoinc function [UL C++]
  UltraLite C++ Component API, 278
IsColumnCurrentDate function [UL C++]
  UltraLite C++ Component API, 278

 Copyright © 2007, iAnywhere Solutions, Inc.

IsColumnCurrentTime function [UL C++]
UltraLite C++ Component API, 279
IsColumnCurrentTimestamp function [UL C++]
UltraLite C++ Component API, 279
IsColumnDescending function [UL C++]
UltraLite C++ Component API, 242
IsColumnGlobalAutoinc function [UL C++]
UltraLite C++ Component API, 279
IsColumnInIndex function [UL C++]
UltraLite C++ Component API, 280
IsColumnNewUUID function [UL C++]
UltraLite C++ Component API, 281
IsColumnNullable function [UL C++]
UltraLite C++ Component API, 281
IsForeignKey function [UL C++]
UltraLite C++ Component API, 242
IsForeignKeyCheckOnCommit function [UL C++]
UltraLite C++ Component API, 243
IsForeignKeyNullable function [UL C++]
UltraLite C++ Component API, 243
IsNeverSynchronized function [UL C++]
UltraLite C++ Component API, 281
IsNull function [UL C++]
UltraLite C++ Component API, 226, 288
IsPrimaryKey function [UL C++]
UltraLite C++ Component API, 243
IsUniqueIndex function [UL C++]
UltraLite C++ Component API, 244
IsUniqueKey function [UL C++]
UltraLite C++ Component API, 244

K

keep_partial_download variable [UL C++]
UltraLite C++ Component API, 181, 188

L

last download timestamp
resetting in UltraLite databases, 324
ULGetLastDownloadTime function, 315
Last function [UL C++]
UltraLite C++ Component API, 226
LastCodeOK function [UL C++]
UltraLite C++ Component API, 206
LastFetchOK function [UL C++]
UltraLite C++ Component API, 206
libraries
UltraLite applications for Palm OS, 78
UltraLite applications for Symbian OS, 88
UltraLite applications for Windows CE, 95
UltraLite compiling and linking in C++, 35
UltraLite DLL for Windows CE, 99
UltraLite import library for ODBC, 137
UltraLite linking example in C++, 109
UltraLite Unicode libraries, 35
library functions
Callback function syntax (UltraLite C/C++), 149
MLFileTransfer (UltraLite embedded SQL), 151
ULChangeEncryptionKey (UltraLite embedded SQL), 308
ULCheckpoint (UltraLite embedded SQL), 309
ULClearEncryptionKey (UltraLite embedded SQL), 310
ULCountUploadRows (UltraLite embedded SQL), 311
ULCreateDatabase (UltraLite embedded SQL), 154
ULDropDatabase (UltraLite embedded SQL), 312
ULEnableEccSyncEncryption (UltraLite C/C++), 156
ULEnableFileDB (UltraLite C/C++), 157
ULEnableFIPSStrongEncryption (UltraLite C/C++), 158
ULEnableHttpsSynchronization (UltraLite C/C++), 160
ULEnableHttpSynchronization (UltraLite C/C++), 159
ULEnablePalmRecordDB (UltraLite C/C++), 161
ULEnableRsaFipsSyncEncryption (UltraLite C/C++), 162
ULEnableRsaSyncEncryption (UltraLite C/C++), 163
ULEnableStrongEncryption (UltraLite C/C++), 164
ULEnableTcpipSynchronization (UltraLite C/C++), 165
ULEnableTlsSynchronization (UltraLite C/C++), 166
ULEnableUserAuthentication (UltraLite C/C++), 167
ULEnableZlibSyncCompression (UltraLite C/C++), 168
ULGetDatabaseID (UltraLite embedded SQL), 313
ULGetDatabaseProperty (UltraLite embedded SQL), 314
ULGetLastDownloadTime (UltraLite embedded SQL), 315

ULGetSynchResult (UltraLite embedded SQL), 316
ULGlobalAutoincUsage (UltraLite embedded SQL), 318
ULGrantConnectTo (UltraLite embedded SQL), 319
ULHTTPSStream (UltraLite embedded SQL), 320
ULHTTPStream (UltraLite embedded SQL), 321
ULInitDatabaseManager (UltraLite C/C++), 169
ULInitDatabaseManagerNoSQL (UltraLite C/C++), 170
ULInitSynchInfo UltraLite (embedded SQL), 322
ULIsSynchronizeMessage (UltraLite embedded SQL), 323
ULRegisterErrorCallback (UltraLite C/C++), 171
ULResetLastDownloadTime (UltraLite embedded SQL), 324
ULRetrieveEncryptionKey (UltraLite embedded SQL), 325
ULRevokeConnectFrom (UltraLite embedded SQL), 326
ULRollbackPartialDownload (UltraLite embedded SQL), 327
ULSaveEncryptionKey (UltraLite embedded SQL), 328
ULSetDatabaseID (UltraLite embedded SQL), 329
ULSetDatabaseOptionString (UltraLite embedded SQL), 330
ULSetDatabaseOptionULong (UltraLite embedded SQL), 331
ULSetSynchInfo UltraLite embedded SQL), 332
ULSocketStream (UltraLite embedded SQL), 333
ULSynchronize (UltraLite embedded SQL), 334
UltraLite embedded SQL, 302
linking
UltraLite applications for Symbian OS, 88
UltraLite applications for Windows CE, 95
UltraLite C++ applications, 35
Lookup function [UL C++]
UltraLite C++ Component API, 270
lookup methods
UltraLite C++, 25
lookup mode
UltraLite C++, 24
LookupBackward function [UL C++]
UltraLite C++ Component API, 270
LookupBegin function [UL C++]
UltraLite C++ Component API, 270
LookupForward function [UL C++]
UltraLite C++ Component API, 271

M

macros
UL_SYNC_ALL, 173
UL_SYNC_ALL_PUBS, 174
UL_TEXT, 174
UL_USE_DLL, 174
UltraLite applications, 173
makefiles
UltraLite embedded SQL, 71
UltraLite ODBC tutorial, 137
managing
UltraLite C++ transactions, 29
Metrowerks CodeWarrior
creating UltraLite projects, 75
MFC
UltraLite applications ActiveSync requirements, 103
MFC applications
UltraLite for Windows CE, 99
MLFileTransfer function [UL ESQL]
syntax, 151
modes
UltraLite C++, 24
moveFirst method (Table object)
UltraLite C++ data retrieval example, 21
moveNext method (Table object)
UltraLite C++ data retrieval example, 21
multi-row queries
UltraLite cursors, 56
multi-threaded applications
UltraLite C++, 18
UltraLite embedded SQL, 43

N

namespaces
UltraLite C++ example, 109
navigating
UltraLite C++ Table API, 23
navigating SQL result sets
UltraLite C++, 21
network protocols
UltraLite for Windows CE, 104
new_password variable [UL C++]
UltraLite C++ Component API, 181, 188

 Copyright © 2007, iAnywhere Solutions, Inc.

newsgroups
technical support, xvii
Next function [UL C++]
UltraLite C++ Component API, 226
NULL
UltraLite indicator variables, 53
Null terminated string UltraLite embedded SQL data type
about, 46
null-terminated TCHAR character string UltraLite SQL data type
about, 47
null-terminated UNICODE character string UltraLite SQL data type
about, 46
null-terminated WCHAR character string UltraLite SQL data type
about, 46
null-terminated wide character string UltraLite SQL data type
about, 46
num_auth_parms variable [UL C++]
UltraLite C++ Component API, 182, 188

O

obfuscating
UltraLite embedded SQL databases, 61
obfuscation
UltraLite C++ development, 33
UltraLite databases using embedded SQL, 62
UltraLite embedded SQL databases, 61
observer synchronization parameter
UltraLite embedded SQL example, 68
observer variable [UL C++]
UltraLite C++ Component API, 182, 189
ODBC
UltraLite about, 136
UltraLite data insertions, 142
UltraLite database connections, 140
UltraLite database creation for, 139
UltraLite database queries, 143
UltraLite development tutorial, 135
UltraLite makefile, 137
ODBC API
UltraLite introducing development for, 137
offsets
UltraLite C++ relative, 23
online books
PDF, x
open method (Table object)
UltraLite C++ data retrieval example, 21
OPEN statement
UltraLite embedded SQL, 56
openByIndex method (Table object)
UltraLite C++ data retrieval example, 21
OpenConnection function [UL C++]
UltraLite C++ Component API, 232
OpenTable function [UL C++]
UltraLite C++ Component API, 216
OpenTableWithIndex function [UL C++]
UltraLite C++ Component API, 216
operator bool function [UL C++]
UltraLite C++ Component API, 296
operator DECL_DATETIME function [UL C++]
UltraLite C++ Component API, 296
operator double function [UL C++]
UltraLite C++ Component API, 296
operator float function [UL C++]
UltraLite C++ Component API, 296
operator int function [UL C++]
UltraLite C++ Component API, 297
operator long function [UL C++]
UltraLite C++ Component API, 297
operator short function [UL C++]
UltraLite C++ Component API, 297
operator ul_s_big function [UL C++]
UltraLite C++ Component API, 297
operator ul_u_big function [UL C++]
UltraLite C++ Component API, 297
operator unsigned char function [UL C++]
UltraLite C++ Component API, 297
operator unsigned int function [UL C++]
UltraLite C++ Component API, 298
operator unsigned long function [UL C++]
UltraLite C++ Component API, 298
operator unsigned short function [UL C++]
UltraLite C++ Component API, 298
operator= function [UL C++]
UltraLite C++ Component API, 298
options
UltraLite compiler for ODBC, 137

P

packed decimal UltraLite embedded SQL data type

about, 46
padding variable [UL C++]
UltraLite C++ Component API, 197
Palm Computing Platform
file-based data store, 157
record-based data store, 161
version 4.0, 157, 161
Palm OS
creator IDs, 81
HotSync synchronization in UltraLite, 82
installing UltraLite applications, 85
platform requirements, 74
security, 84
UltraLite building CustDB application using CodeWarrior, 77
UltraLite C++ applications, 74
UltraLite HTTP synchronization in C/C++, 84
UltraLite runtime libraries for, 35
UltraLite TCP/IP synchronization in C/C++, 84
partial_download_retained variable [UL C++]
UltraLite C++ Component API, 182, 189, 194
password variable [UL C++]
UltraLite C++ Component API, 182, 189
passwords
UltraLite C++ API authentication in , 32
PATH environment variable
HotSync, 74
PDF
documentation, x
performance
UltraLite avoiding re-entry of encryption key, 80
UltraLite naming databases explicitly, 13
UltraLite using DLL for economical memory use, 95
UltraLite using INSERT statements , 65
permissions
UltraLite embedded SQL, 40
persistent storage
UltraLite for Windows CE, 98
ping variable [UL C++]
UltraLite C++ Component API, 182, 189
platform requirements
UltraLite for Windows CE, 94
platforms
supported in UltraLite C++, 6
prefix files
about, 77
prepare function
UltraLite ODBC, 143
prepared statements
UltraLite C++, 19
preparedStatement class
UltraLite C++, 19
PrepareStatement function [UL C++]
UltraLite C++ Component API, 217
preprocessing
UltraLite embedded SQL applications, 70
UltraLite embedded SQL development tool settings, 71
Previous function [UL C++]
UltraLite C++ Component API, 227
program structure
UltraLite embedded SQL, 40
protocols
UltraLite for Windows CE, 104
publication variable [UL C++]
UltraLite C++ Component API, 183, 189
publications
UltraLite C++ API schema information in , 30
PublicationSchema object
UltraLite C++ development, 30

Q

queries
UltraLite embedded SQL multi-row queries, 56
UltraLite embedded SQL single-row queries, 55
UltraLite ODBC, 143

R

received variable [UL C++]
UltraLite C++ Component API, 198
registry
ClientParms registry entry, 83
Relative function [UL C++]
UltraLite C++ Component API, 227
relative offset
UltraLite C++ Table API, 23
Release function [UL C++]
UltraLite C++ Component API, 259
Reopen method
UltraLite C/C++, 79
ResetLastDownloadTime function [UL C++]
UltraLite C++ Component API, 217
restartable downloads
UltraLite embedded SQL, 327

 Copyright © 2007, iAnywhere Solutions, Inc.

result set schemas
UltraLite C++, 22
result sets
UltraLite C++ navigation of, 21
resume_partial_download variable [UL C++]
UltraLite C++ Component API, 183, 190
RevokeConnectFrom function [UL C++]
UltraLite C++ Component API, 217
revokeConnectionFrom method
UltraLite C++ development, 32
Rollback function [UL C++]
UltraLite C++ Component API, 218
rollback method
UltraLite C++ transactions, 29
RollbackPartialDownload function [UL C++]
UltraLite C++ Component API, 218
rollbacks
UltraLite C++ transactions, 29
rows
accessing in UltraLite C++ API tutorial, 114
UltraLite C++ table access of current, 24
UltraLite C++ table navigation, 23
UltraLite deletions with C++ API, 28
UltraLite insertions with C++ API, 27
UltraLite updates with C++ API, 26
runtime libraries
UltraLite applications for Windows CE, 95
UltraLite C++, 35
UltraLite for C++, 35
runtime library
Symbian OS, 88
Windows CE, 174

S

sample application
UltraLite building for Palm OS, 77
UltraLite building for Windows CE, 96
samples-dir
documentation usage, xiv
saving state
UltraLite on Palm OS, 79
schemas
UltraLite C++ API access, 30
scrolling
UltraLite C++ Table API, 23
searching
UltraLite rows with C++, 25
security
changing the encryption key in UltraLite embedded SQL, 61
encryption on Palm, 80
obfuscation in UltraLite embedded SQL, 61
UltraLite C/C++ applications, 84
UltraLite database encryption, 61
SELECT statement
UltraLite C++ data retrieval example, 21
UltraLite embedded SQL single row, 55
send_column_names variable [UL C++]
UltraLite C++ Component API, 183, 190
send_download_ack variable [UL C++]
UltraLite C++ Component API, 183, 190
sent variable [UL C++]
UltraLite C++ Component API, 199
SET CONNECTION statement
multiple connections in UltraLite embedded SQL, 43
Set function [UL C++]
UltraLite C++ Component API, 227
SetBinary function [UL C++]
UltraLite C++ Component API, 289
SetDatabaseID function [UL C++]
UltraLite C++ Component API, 218
SetDatabaseOption function [UL C++]
UltraLite C++ Component API, 218
SetDefault function [UL C++]
UltraLite C++ Component API, 228
SetNull function [UL C++]
UltraLite C++ Component API, 228
SetParameter function [UL C++]
UltraLite C++ Component API, 248
SetParameterNull function [UL C++]
UltraLite C++ Component API, 249
SetReadPosition function [UL C++]
UltraLite C++ Component API, 263
SetString function [UL C++]
UltraLite C++ Component API, 289
SetSuspend function [UL C++]
UltraLite C++ Component API, 219, 228
SetSynchInfo function [UL C++]
UltraLite C++ Component API, 219
Shutdown function [UL C++]
UltraLite C++ Component API, 220, 233
simple encryption
UltraLite database simple encryption, 62
SQL Anywhere

documentation, x
SQL Communications Area
UltraLite C/C++, 12
UltraLite embedded SQL, 41
SQL preprocessor
UltraLite embedded SQL applications, 70
sql_code variable [UL C++]
UltraLite C++ Component API, 194
SQLAllocHandle function [UL ODBC]
syntax, 337
SQLBindCol function [UL ODBC]
syntax, 338
SQLBindParameter function [UL ODBC]
syntax, 339
SQLCA
UltraLite C/C++, 12
UltraLite embedded SQL, 41
UltraLite embedded SQL multiple SQLCA, 43
UltraLite fields, 41
sqlcabc SQLCA field
UltraLite embedded SQL, 41
sqlcaid SQLCA field
UltraLite embedded SQL, 41
SQLCODE
Callback function syntax (UltraLite C/C++), 149
UltraLite C++ error handling, 31
sqlcode SQLCA field
UltraLite embedded SQL, 41
SQLConnect function [UL ODBC]
syntax, 340
SQLDescribeCol function [UL ODBC]
syntax, 341
SQLDisconnect function [UL ODBC]
syntax, 342
SQLEndTran function [UL ODBC]
syntax, 343
sqlerrd SQLCA field
UltraLite embedded SQL, 42
sqlerrmc SQLCA field
UltraLite embedded SQL, 41
sqlerrml SQLCA field
UltraLite embedded SQL, 41
sqlerrp SQLCA field
UltraLite embedded SQL, 42
SQLExecDirect function [UL ODBC]
syntax, 344
SQLExecute function [UL ODBC]
syntax, 345
SQLFetch function [UL ODBC]
syntax, 346
SQLFetchScroll function [UL ODBC]
syntax, 347
SQLFreeHandle function [UL ODBC]
syntax, 348
SQLGetCursorName function [UL ODBC]
syntax, 349
SQLGetData function [UL ODBC]
syntax, 350
SQLGetDiagRec function [UL ODBC]
syntax, 351
SQLGetInfo function [UL ODBC]
syntax, 352
SQLNumResultCols function [UL ODBC]
syntax, 353
sqlpp utility
UltraLite embedded SQL applications, 70
SQLPrepare function [UL ODBC]
syntax, 354
SQLRowCount function [UL ODBC]
syntax, 355
SQLSetConnectionName function [UL ODBC]
syntax, 356
SQLSetCursorName function [UL ODBC]
syntax, 357
SQLSetSuspend function [UL ODBC]
syntax, 358
sqlstate SQLCA field
UltraLite embedded SQL, 42
SQLSynchronize function [UL ODBC]
syntax, 359
sqlwarn SQLCA field
UltraLite embedded SQL, 42
stack and heap sizing
Symbian OS, 88
StartSynchronizationDelete function [UL C++]
UltraLite C++ Component API, 220
state variable [UL C++]
UltraLite C++ Component API, 199
statements
UltraLite ODBC, 143
static libraries
UltraLite C++ applications, 35
status variable [UL C++]
UltraLite C++ Component API, 194
stop variable [UL C++]
UltraLite C++ Component API, 199

 Copyright © 2007, iAnywhere Solutions, Inc.

StopSynchronizationDelete function [UL C++]
UltraLite C++ Component API, 220
stream definition functions
ULHTTPSStream (UltraLite embedded SQL), 320
ULHTTPStream (UltraLite), 321
ULSetDatabaseID (embedded SQL), 329
ULSocketStream (UltraLite embedded SQL), 333
stream variable [UL C++]
UltraLite C++ Component API, 183, 190
stream_error variable [UL C++]
UltraLite C++ Component API, 184, 190, 194
stream_parms variable [UL C++]
UltraLite C++ Component API, 184, 190
string UltraLite embedded SQL data type
about, 46
fixed length, 47
variable length, 47
StringCompare function [UL C++]
UltraLite C++ Component API, 290
strings
UL_TEXT macro, 174
strong encryption
UltraLite databases, 164
UltraLite embedded SQL, 61
StrToUUID function [UL C++]
UltraLite C++ Component API, 220
support
newsgroups, xvii
supported platforms
UltraLite C++, 6
Symbian OS
development environments, 88
linking and runtime libraries, 88
stack and heap sizing, 88
UltraLite C++ applications, 88
synchronization
adding to UltraLite embedded SQL applications, 63
canceling in UltraLite embedded SQL, 66
HotSync in UltraLite, 82
initial in UltraLite embedded SQL, 65
invoking in UltraLite embedded SQL, 64
monitoring in UltraLite embedded SQL, 66
Palm OS in UltraLite, 82
troubleshooting in UltraLite embedded SQL, 316
ULEnableEccaSyncEncryption function in UltraLite C/C++, 156
ULEnableFIPSStrongEncryption function in UltraLite C/C++, 158
ULEnableHttpsSynchronization function in UltraLite C/C++, 160
ULEnableHttpSynchronization function in UltraLite C/C++, 159
ULEnableRsaFipsSyncEncryption in UltraLite C/C++, 162
ULEnableRsaSyncEncryption function in UltraLite C/C++, 163
ULEnableTcpipSynchronization function in UltraLite C/C++, 165
ULEnableTlsSynchronization function in UltraLite C/C++, 166
ULEnableZlibSyncCompression function in UltraLite C/C++, 168
UltraLite C++ API tutorial, 116
UltraLite embedded SQL, 63
UltraLite embedded SQL committing changes in , 65
UltraLite embedded SQL example, 64
UltraLite embedded SQL tutorial, 133
UltraLite for Windows CE introduction, 102
UltraLite for Windows CE menu contro forl, 104
UltraLite HTTP in C/C++, 84
UltraLite ODBC interface, 359
UltraLite TCP/IP in C/C++, 84
synchronization errors
communications errors in UltraLite embedded SQL, 65
synchronization functions
ULInitSynchInfo (UltraLite embedded SQL), 322
ULSetSynchInfo (UltraLite embedded SQL), 332
synchronization status
ULGetSynchResult function, 316
Synchronize function [UL C++]
UltraLite C++ Component API, 221
synchronizing
CodeWarrior encryption libraries, 77
UltraLite C/C++ applications, 34
synchronizing data
UltraLite about, 34

T

Table API
UltraLite C++ introduction, 23
Table object

UltraLite C++ data retrieval example, 21
table_order variable [UL C++]
UltraLite C++ Component API, 184, 191
tableCount variable [UL C++]
UltraLite C++ Component API, 199
tableIndex variable [UL C++]
UltraLite C++ Component API, 199
tables
UltraLite C++ API schema information in , 30
TableSchema object
UltraLite C++ development, 30
target platforms
UltraLite C++, 6
TCP/IP synchronization
UltraLite for Palm OS in C/C++, 84
technical support
newsgroups, xvii
threads
UltraLite C++ API multi-threadedapplications, 18
UltraLite embedded SQL, 43
timestamp structure UltraLite embedded SQL data type
about, 47
timestamp variable [UL C++]
UltraLite C++ Component API, 194
tips
UltraLite development, 65
transaction processing
UltraLite C++ management of, 29
transactions
commiting in UltraLite with embedded SQL, 65
UltraLite C++ management of, 29
troubleshooting
newsgroups, xvii
previous synchronization, 316
UltraLite C++ handling errors, 31
UltraLite C/C++ using ULRegisterErrorCallback, 171
UltraLite development, 65
UltraLite synchronization with embedded SQL, 65
UltraLite using reference expressions in SQL preprocessor, 50
TruncateTable function [UL C++]
UltraLite C++ Component API, 271
truncation
UltraLite FETCH, 54
tutorials
UltraLite C++ API, 107
UltraLite embedded SQL, 121
UltraLite ODBC, 135

U

UL_AS_SYNCHRONIZE macro
ActiveSync UltraLite messages, 173
UL_SYNC_ALL macro
about, 173
UL_SYNC_ALL_PUBS macro
about, 174
ul_synch_info structure
about, 64
ul_synch_info_a struct [UL C++]
UltraLite C++ Component API, 179
ul_synch_info_w2 struct [UL C++]
UltraLite C++ Component API, 186
ul_synch_result struct [UL C++]
UltraLite C++ Component API, 193
ul_synch_stats struct [UL C++]
UltraLite C++ Component API, 196
ul_synch_status struct [UL C++]
UltraLite C++ Component API, 198
ul_synch_status structure
UltraLite embedded SQL, 66
UL_TEXT macro
about, 174
UL_USE_DLL macro
about, 174
ULActiveSyncStream function
Windows CE usage, 102
ulbase.lib
UltraLite C++ development, 35
ULChangeEncryptionKey function [UL ESQL]
syntax, 308
using, 61
ULCheckpoint function [UL ESQL]
syntax, 309
ULClearEncryptionKey function [UL ESQL]
syntax, 310
using, 80
ULCountUploadRows function [UL ESQL]
syntax, 311
ULCreateDatabase function [UL ESQL]
syntax, 154
ULDropDatabase function [UL ESQL]
syntax, 312
ULEnableEccSyncEncryption function [UL C/C++]
syntax, 156

 Copyright © 2007, iAnywhere Solutions, Inc.

ULEnableFileDB function [UL C/C++]
syntax, 157
ULEnableFIPSStrongEncryption function [UL C/C++]
syntax, 158
ULEnableHttpsSynchronization function [UL C/C++]
syntax, 160
ULEnableHttpSynchronization function [UL C/C++]
syntax, 159
ULEnablePalmRecordDB function [UL C/C++]
syntax, 161
ULEnableRsaFipsSyncEncryption function [UL C/C+
+]
syntax, 162
ULEnableRsaSyncEncryption function [UL C/C++]
syntax, 163
ULEnableStrongEncryption function [UL C/C++]
syntax, 164
ULEnableTcpipSynchronization function [UL C/C++]
syntax, 165
ULEnableTlsSynchronization function [UL C/C++]
syntax, 166
ULEnableUserAuthentication function [UL C/C++]
syntax, 167
ULEnableZlibSyncCompression function [UL C/C++]
syntax, 168
ULGetCollation_ function (UltraLite C/C++)
Creating UltraLite databases, 154
ULGetDatabaseID function [UL ESQL]
syntax, 313
ULGetDatabaseProperty function [UL ESQL]
syntax, 314
ULGetLastDownloadTime function [UL ESQL]
syntax, 315
ULGetSynchResult function [UL ESQL]
syntax, 316
ULGlobalAutoincUsage function [UL ESQL]
syntax, 318
ULGrantConnectTo function [UL ESQL]
syntax, 319
ULHTTPSStream function [UL ESQL]
syntax, 320
ULHTTPStream function [UL ESQL]
syntax, 321
ULInitDatabaseManager function [UL C/C++]
connecting to a database, 17
syntax, 169
ULInitDatabaseManagerNoSQL function [UL C/C++]
connecting to a database, 18
syntax, 170
ULInitSynchInfo function [UL ESQL]
about, 64
syntax, 322
ULIsSynchronizeMessage function [UL ESQL]
ActiveSync usage, 102
syntax, 323
ULRegisterErrorCallback function [UL C/C++]
Callback function syntax (UltraLite C/C++), 149
syntax, 171
ULResetLastDownloadTime function [UL ESQL]
syntax, 324
ULRetrieveEncryptionKey function [UL ESQL]
syntax, 325
using, 80
ULRevokeConnectFrom function [UL ESQL]
syntax, 326
ULRollbackPartialDownload function [UL ESQL]
syntax, 327
ulrt.lib
UltraLite C++ development, 35
ulrt10.dll
UltraLite C++ development, 35
ulrtc.lib
UltraLite C++ development, 35
ULSaveEncryptionKey function [UL ESQL]
syntax, 328
using, 80
ULSetDatabaseID function [UL ESQL]
syntax, 329
ULSetDatabaseOptionString function [UL ESQL]
syntax, 330
ULSetDatabaseOptionULong function [UL ESQL]
syntax, 331
ULSetSynchInfo function [UL ESQL]
syntax, 332
using, 82
ULSocketStream function [UL ESQL]
syntax, 333
ULSqlca class [UL C++]
UltraLite C++ Component API, 200
ULSqlca function [UL C++]
UltraLite C++ Component API, 200
ULSqlcaBase class [UL C++]
UltraLite C++ Component API, 202
ULSqlcaWrap class [UL C++]
UltraLite C++ Component API, 207
ULSqlcaWrap function [UL C++]

UltraLite C++ Component API, 207
ULSynchronize function [UL ESQL]
serial port on Palm OS, 84
syntax, 334
UltraLite embedded SQL tutorial, 133
ULTable objects
reopening, 79
UltraLite applications
synchronization of C/C++ applications, 34
UltraLite C++
development, 15
UltraLite C/C++
about, 3
accessing schema information, 30
architecture, 7
authentication, 32
common features, 12
data manipulation with SQL, 19
encryption, 33
obfuscating UltraLite databases, 62
Palm OS, 79
Reopen methods, 79
supported platforms, 6
synchronizing data, 34
Table API introduction, 23
tutorials, 107
UltraLite C/C++ Common API
alphabetical listing, 147
UltraLite databases
connecting in UltraLite C++, 17
deploying on Palm OS, 85
encrypting in embedded SQL, 61
ODBC connections to, 140
ODBC creation of, 139
ODBC data insertions, 142
ODBC queries, 143
UltraLite C++ information access , 30
UltraLite for Windows CE, 98
UltraLite embedded SQL
authorization, 40
building CustDB application, 96
cursors, 56
developing applications, 37
fetching data, 55
functions, 302
host variables, 45
sample program, 126
synchronization, 63
tutorial, 121
using, 303
UltraLite embedded SQL library functions
MLFileTransfer, 151
ULChangeEncryptionKey, 308
ULCheckpoint, 309
ULClearEncryptionKey, 310
ULCountUploadRows, 311
ULCreateDatabase, 154
ULDropDatabase, 312
ULEnableEccSyncEncryption, 156
ULEnableFileDB, 157
ULEnableFIPSStrongEncryption, 158
ULEnableHttpsSynchronization, 160
ULEnableHttpSynchronization, 159
ULEnablePalmRecordDB, 161
ULEnableRsaFipsSyncEncryption, 162
ULEnableRsaSyncEncryption, 163
ULEnableStrongEncryption, 164
ULEnableTcpipSynchronization, 165
ULEnableTlsSynchronization, 166
ULEnableUserAuthentication, 167
ULEnableZlibSyncCompression, 168
ULGetDatabaseID, 313
ULGetDatabaseProperty, 314
ULGetLastDownloadTime, 315
ULGetSynchResult, 316
ULGlobalAutoincUsage, 318
ULGrantConnectTo, 319
ULHTTPSStream, 320
ULHTTPStream, 321
ULInitSynchInfo, 322
ULIsSynchronizeMessage, 323
ULResetLastDownloadTime, 324
ULRetrieveEncryptionKey, 325
ULRevokeConnectFrom, 326
ULRollbackPartialDownload function, 327
ULSaveEncryptionKey, 328
ULSetDatabaseID, 329
ULSetDatabaseOptionString, 330
ULSetDatabaseOptionULong, 331
ULSetSynchInfo, 332
ULSocketStream, 333
ULSynchronize, 334
UltraLite namespace
UltraLite C++, 16
UltraLite ODBC interface
alphabetical listing, 335

 Copyright © 2007, iAnywhere Solutions, Inc.

SQLAllocHandle function, 337
SQLBindCol function, 338
SQLBindParameter function, 339
SQLConnect function, 340
SQLDescribeCol function, 341
SQLEndTran function, 343
SQLExecDirect function, 344
SQLExecute function, 345
SQLFetch function, 346
SQLFetchScroll function, 347
SQLFreeHandle function, 348
SQLGetCursorName function, 349
SQLGetData function, 350
SQLGetDiagRec function, 351
SQLGetInfo function, 352
SQLNumResultCols function, 353
SQLPrepare function, 354
SQLRowCount function, 355
SQLSetConnectionName function, 356
SQLSetCursorName function, 357
SQLSetSuspend function, 358
SQLSynchronize function, 359
UltraLite plug-in
CodeWarrior installing, 75
CodeWarrior libraries for encrypted synchronization, 77
CodeWarrior project conversion, 76
CodeWarrior project creation, 75
CodeWarrior usage, 77
UltraLite projects
CodeWarrior, 75
UltraLite runtime
deploying Windows CE libraries, 99
UltraLite C++ libraries, 35
UltraLite SQL
ODBC queries, 143
UltraLite_ classes
using the UltraLite namespace, 16
UltraLite_Connection class [UL C++]
UltraLite C++ Component API, 209
UltraLite_Connection_iface class [UL C++]
UltraLite C++ Component API, 210
UltraLite_Cursor_iface class [UL C++]
UltraLite C++ Component API, 223
UltraLite_DatabaseManager class [UL C++]
UltraLite C++ Component API, 230
UltraLite_DatabaseManager_iface class [UL C++]
UltraLite C++ Component API, 231
UltraLite_DatabaseSchema class [UL C++]
UltraLite C++ Component API, 234
UltraLite_DatabaseSchema_iface class [UL C++]
UltraLite C++ Component API, 235
UltraLite_IndexSchema class [UL C++]
UltraLite C++ Component API, 239
UltraLite_IndexSchema_iface class [UL C++]
UltraLite C++ Component API, 240
UltraLite_PreparedStatement class [UL C++]
UltraLite C++ Component API, 245
UltraLite_PreparedStatement_iface class [UL C++]
UltraLite C++ Component API, 246
UltraLite_ResultSet class [UL C++]
UltraLite C++ Component API, 250
UltraLite_ResultSet_iface class [UL C++]
UltraLite C++ Component API, 251
UltraLite_ResultSetSchema class [UL C++]
UltraLite C++ Component API, 252
UltraLite_RowSchema_iface class [UL C++]
UltraLite C++ Component API, 253
UltraLite_SQLObject_iface class [UL C++]
UltraLite C++ Component API, 258
UltraLite_StreamReader class [UL C++]
UltraLite C++ Component API, 260
UltraLite_StreamReader_iface class [UL C++]
UltraLite C++ Component API, 261
UltraLite_StreamWriter class [UL C++]
UltraLite C++ Component API, 264
UltraLite_Table class [UL C++]
UltraLite C++ Component API, 265
UltraLite_Table_iface class [UL C++]
UltraLite C++ Component API, 266
UltraLite_TableSchema class [UL C++]
UltraLite C++ Component API, 272
UltraLite_TableSchema_iface class [UL C++]
UltraLite C++ Component API, 273
ULValue class [UL C++]
UltraLite C++ Component API, 283
ULValue function [UL C++]
UltraLite C++ Component API, 290, 291, 292, 293, 293, 294, 295
uncommitted transactions
UltraLite embedded SQL, 65
UNDER_CE compiler directive
about, 174
UNDER_PALM_OS compiler directive
about, 174
UNICODE characters

UltraLite C++ library for, 35
Update function [UL C++]
UltraLite C++ Component API, 229
update mode
UltraLite C++, 24
UpdateBegin function [UL C++]
UltraLite C++ Component API, 229
updates variable [UL C++]
UltraLite C++ Component API, 197
updating
UltraLite C++ API table rows, 26
upload_ok variable [UL C++]
UltraLite C++ Component API, 184, 191, 194
upload_only variable [UL C++]
UltraLite C++ Component API, 184, 191
user authentication
UltraLite C++ development, 32
UltraLite C/C++ applications, 167
UltraLite deprecated ULEnableUserAuthentication function (UltraLite C/C++), 167
UltraLite embedded SQL, 326
UltraLite embedded SQL applications, 59
UltraLite embedded SQL grant method for, 319
UltraLite embedded SQL revoke method for, 326
UltraLite ULGrantConnectTo (UltraLite embedded SQL), 319
user_data variable [UL C++]
UltraLite C++ Component API, 185, 191
user_name variable [UL C++]
UltraLite C++ Component API, 185, 191
UUIDToStr function [UL C++]
UltraLite C++ Component API, 222

V

values
UltraLite C++ API accessing in , 24
version variable [UL C++]
UltraLite C++ Component API, 185, 192
Visual C++
UltraLite for Windows CE development, 94

W

WindowProc function
ActiveSync, 323
ActiveSync usage, 102
Windows
UltraLite runtime libraries for, 35
Windows CE
UltraLite application development overview, 94
UltraLite application synchronization, 102
UltraLite building CustDB application using eMbedded Visual C++, 96
UltraLite class names, 99
UltraLite platform requirements, 94
UltraLite runtime libraries for, 35
UltraLite synchronization menu control, 104
UltraLite synchronization with ActiveSync, 99
winsock.lib
UltraLite Windows CE applications, 94

 Copyright © 2007, iAnywhere Solutions, Inc.