



UltraLite® M-Business Anywhere Programming

Published: October 2006

Copyright and trademarks

Copyright © 2006 iAnywhere Solutions, Inc. Portions copyright © 2006 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	v
SQL Anywhere documentation	vi
Documentation conventions	ix
Finding out more and providing feedback	xiii
Introduction to UltraLite for M-Business Anywhere	1
UltraLite for M-Business Anywhere features	2
UltraLite for M-Business Anywhere architecture	3
Understanding UltraLite for M-Business Anywhere Development	5
UltraLite for M-Business Anywhere Quick Start	6
Connecting to an UltraLite database	10
Maintaining connections and application state across pages	11
Persistent names in M-Business Anywhere applications	12
Database encryption and obfuscation	15
Working with data using SQL	16
Working with data using the table API	20
Accessing schema information	26
Handling errors	27
Authenticating users	28
Synchronizing data	29
Deploying UltraLite for M-Business Anywhere applications	32
Tutorial: A Sample Application for M-Business Anywhere	35
Introduction	36
Lesson 1: Create project architecture	37
Lesson 2: Create the application files	39
Lesson 3: Set up the M-Business Anywhere Server and Client	41
Lesson 4: Add startup code to your application	43
Lesson 5: Add inserts to your application	46

Lesson 6: Add navigation to your application	49
Lesson 7: Add updates and deletes to your application	50
Lesson 8: Add synchronization to your application	52
UltraLite for M-Business Anywhere API Reference	55
Data types in UltraLite for M-Business Anywhere	56
Class AuthStatusCode	57
Class Connection	58
Class ConnectionParms	66
Class CreationParms	68
Class DatabaseManager	70
Class DatabaseSchema	73
Class IndexSchema	78
Class PreparedStatement	81
Class PublicationSchema	91
Class ResultSet	92
Class ResultSetSchema	108
Class SQLError	112
Class SQLType	120
Class SyncParms	122
Class SyncResult	133
Class TableSchema	139
Class ULTable	150
Class UUID	175
Index	177

About This Manual

Subject

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.

M-Business Anywhere is the iAnywhere platform for developing and deploying mobile web-based applications. The previous name for the product was AvantGo M-Business Server.

Audience

This manual is intended for 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 comprehensive package that provides data management and data exchange, 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.
- ◆ **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, security, backup procedures, security, 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 those 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 applications.
- ◆ **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.

- ◆ **QAnywhere** This manual describes QAnywhere, which defines 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 provides context-sensitive help for the Connect dialog, the Query Editor, the MobiLink Monitor, 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.

On Windows, the PDF books are accessible from the online books 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 accessible on your installation CD.

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.

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
```

- ◆ **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 “[File Locations and Installation Settings](#)” [*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*.

After installation is complete, the environment variable `SQLANYXSAMP10` 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 “[The samples directory](#)” [*SQL Anywhere Server - Database Administration*].

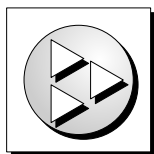
- ◆ **Environment variables** The documentation refers to setting environment variables. On Windows, environment variables are referred to using the syntax *%envvar%*. On Unix, Linux, and Mac OS X, environment variables are referred to using the syntax *\$envvar* or *\${envvar}*.

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

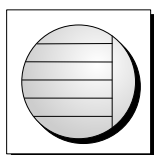
Graphic icons

The following icons are used in this documentation.

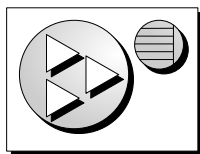
- ◆ A client application.



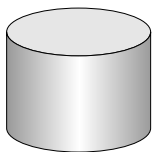
- ◆ A database server, such as SQL Anywhere.



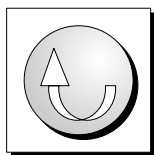
- ◆ An UltraLite application.



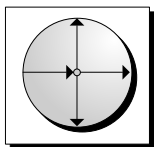
- ◆ A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



- ◆ 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.



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 iAnywhere Solutions 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 Solutions 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.

CHAPTER 1

Introduction to UltraLite for M-Business Anywhere

Contents

UltraLite for M-Business Anywhere features 2
UltraLite for M-Business Anywhere architecture 3

About this chapter

This chapter introduces UltraLite for M-Business Anywhere. It assumes that you are familiar with the features of UltraLite, as described in “[Introducing UltraLite](#)” [*UltraLite - Database Management and Reference*].

UltraLite for M-Business Anywhere features

UltraLite for M-Business Anywhere is a relational data management system for mobile devices. It has the performance, resource efficiency, robustness, and security required by business applications. UltraLite also provides synchronization with enterprise data stores.

System requirements and supported platforms

Development platforms

To develop applications using UltraLite for M-Business Anywhere, you require the following:

- ◆ M-Business Anywhere is the new name for AvantGo M-Business Server. This software requires M-Business Server 5.3 or later, and the corresponding M-Business Anywhere client.

Target platforms

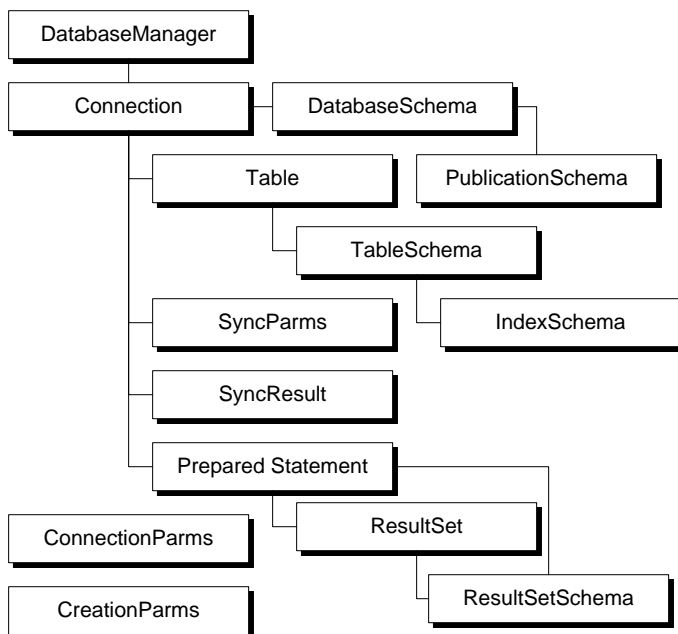
UltraLite for M-Business Anywhere supports the following target platforms:

- ◆ Windows CE 3.0 and higher, with Pocket PC on the ARM processor, including Windows Mobile 5.0.
- ◆ Palm OS version 5.0 and higher.
- ◆ Windows XP, starting with M-Business Anywhere 5.5.

☞ For more information, see the UltraLite table in [UltraLite Deployment Option for SQL Anywhere](#).

UltraLite for M-Business Anywhere architecture

The UltraLite programming interface exposes a set of objects for data manipulation using an UltraLite database. The following figure describes the object hierarchy.



The following list describes some of the more commonly-used high-level objects.

- ◆ **DatabaseManager** manages connections to UltraLite databases.

For more information, see [“Class DatabaseManager” on page 70](#).

- ◆ **ConnectionParms** holds a set of connection parameters.

For more information, see [“Class ConnectionParms” on page 66](#).

- ◆ **CreationParms** holds a set of database creation parameters.

For more information, see [“Class CreationParms” on page 68](#).

- ◆ **Connection** represents a database connection, and governs transactions.

For more information, see [“Class Connection” on page 58](#).

- ◆ **PreparedStatement, ResultSet, and ResultSetSchema** manage database requests and their results using SQL.

For more information, see [“Class PreparedStatement” on page 81](#), [“Class ResultSet” on page 92](#), and [“Class ResultSetSchema” on page 108](#).

- ◆ **Table** manages data using a table-based API.

For more information, see [“Class ULTable” on page 150](#).

- ◆ **SyncParms and SyncResult** manage synchronization through the MobiLink synchronization server.

For more information about synchronization with MobiLink, see [“UltraLite Clients” \[MobiLink - Client Administration\]](#).

CHAPTER 2

Understanding UltraLite for M-Business Anywhere Development

Contents

UltraLite for M-Business Anywhere Quick Start	6
Connecting to an UltraLite database	10
Maintaining connections and application state across pages	11
Persistent names in M-Business Anywhere applications	12
Database encryption and obfuscation	15
Working with data using SQL	16
Working with data using the table API	20
Accessing schema information	26
Handling errors	27
Authenticating users	28
Synchronizing data	29
Deploying UltraLite for M-Business Anywhere applications	32

About this chapter

This chapter describes application development using UltraLite for M-Business Anywhere.

UltraLite for M-Business Anywhere Quick Start

The following procedures describe how to run the supplied CustDB and Simple sample applications.

Before you start, ensure that you have M-Business Anywhere 6.0 or later installed and running, and that you have administrative privileges on the server. You must also have a supported handheld device.

◆ To install and run M-Business Anywhere samples

1. Copy the UltraLite for M-Business Anywhere sample files to your installation directory for deployment.

- a. Open a command prompt and change directory to the *samples-dir* \UltraLiteForMBusinessAnywhere\CustDB subdirectory of your SQL Anywhere installation.
- b. Run the following command:

```
build.bat deploy-dir
```

where *deploy-dir* is the directory where the CustDB and UltraLite files are to be deployed. For example, you may choose *C:\tutorial\mba*

The batch file copies the required files to the location you specify. To see what files are being copied, examine the file *samples-dir\UltraLiteForMBusinessAnywhere\CustDB\build.bat* using a text editor.

2. Create a virtual directory in your web server that points to the directory *deploy-dir* specified in step 1. The following instructions are for Microsoft IIS:
 - a. Open the IIS management tool.
 - b. Right-click your web site and choose New ► Virtual Directory. Name this virtual directory **CustDB** and specify your deployment directory *deploy-dir* as the content directory. Leave the other settings at their default values.
 - c. Right-click the new virtual directory and choose Properties. In the HTTP Headers tab, click File Types and register the following file extensions as the type application/octet-stream:
 - ◆ For Windows and Windows CE: cab, dll
 - ◆ For Palm OS: pdb, prc
 - ◆ udb
 - d. Make a note of the URL that accesses the file *main.htm* in this virtual directory. In a default installation this would be *http://localhost/CustDB/main.htm*.

3. Add users to M-Business Anywhere.

There are three ways to add new users to M-Business Anywhere: by creating new user profiles, by allowing users to self-register, and by importing a CSV file. These instructions describe how to create a new user profile. For more information, see the M-Business Anywhere documentation.

- a. Log in to M-Business Anywhere as an administrator.

The default administrator account settings are a user ID of **Admin** and an empty password.

- b. In the left panel, click Users.
 - c. Click Create User. The Create User page appears.
 - d. Type a unique user name in the User Name field.
 - e. Type the same password in the Password and the Confirm Password fields.
 - f. Click Save to add the user.
4. Deploy the M-Business Anywhere Client to a handheld device or PC.
- a. Click the Download Client Software Only link on the M-Business Anywhere login page. Run the installation to install the client.
 - b. On the handheld device or PC, configure M-Business Connect to synchronize with the M-Business Anywhere server.

Enter the user ID and password of the new M-Business user account you created.
 - c. Synchronize to the M-Business Anywhere server at least.

If you have connection problems at this stage, specify the IP number rather than the host name as the host to avoid name resolution issues with some versions of ActiveSync.

 For more information, see your M-Business Anywhere documentation.

5. Add a group to M-Business Anywhere.

The group will be used to test UltraLite for M-Business Anywhere.

- a. From a web browser, connect to M-Business Anywhere.

The default URL is *http://localhost* or *http://localhost:8091*.
 - b. Log in using the administrator account.
 - c. Click the Groups option in the left navigation panel, and click Create Group.
 - d. Name your group **UltraLite Samples**.
6. Configure the M-Business Anywhere channel settings.
- a. Add the user you created in step 3 to the group UltraLite Samples using the Users option in the left panel under Edit Group.
 - b. Use the group's "Channels" option in the left navigation panel to create the following channel:

Setting	Value
Channel Name	CustDB
Location	<i>http://localhost/CustDB/main.htm</i> or the URL from step 2.

Setting	Value
Size	1000
Link depth	3
Allow binary distribution	Yes (checked)
Hidden	No (unchecked)

After setting the Location value, click View to confirm that you have entered the value correctly.

7. Synchronize your client.

The initial synchronization downloads the UltraLite for M-Business Anywhere files onto your handheld device.

◆ **To verify your setup**

1. Check that the required files are present.

- ◆ On Pocket PC, after you have synchronized the device, check that the following files are in the *\Program Files\AvantGo\Pods* folder:

- ◆ *ulpod10.dll*
- ◆ *custdb.udb*

If any of these files is missing, you may have to manually copy them over to the device.

- ◆ On Palm OS, after you have synchronized your device, check Palm OS App Info for the presence of the following:

- ◆ *ulpod*
- ◆ *custdb*

If any of these are missing, you may have to use the Palm install utility to install the UltraLite for M-Business Anywhere runtime *.prc* and sample schemas *.pdb* files to the device.

- ◆ On Windows XP, after you have synchronized your device, check that the following files are in the *AvantGo\Pods* subdirectory of your *AvantGo Connect* folder:

- ◆ *ulpod10.dll*
- ◆ *custdb.udb*

If any of these files are missing, you may have to manually copy them over to the device.

2. Launch M-Business Client.

On your handheld device or PC, check that the About screen displays the UltraLite for M-Business Anywhere version number. This confirms that UltraLite for M-Business Anywhere is successfully installed.

3. Run the CustDB sample application.
 - a. Start the MobiLink server on your desktop.

From the Start menu, choose Programs ► SQL Anywhere 10 ► MobiLink ► Synchronization Server Sample.
 - b. Start the CustDB application on your M-Business Client.

The CustDB application is a link on your M-Business home page.
 - c. Enter your user ID.

The default value is **50**.
 - d. Synchronize.

Either answer Yes to the prompt "Do you have a network connection now?" or, in the CustDB application, click Synchronize. This synchronizes data with MobiLink, and is a separate operation from synchronizing with M-Business Anywhere.

You should now see data in the CustDB fields. You can now explore the CustDB application.

☞ For information about the CustDB sample application, see [“Exploring the CustDB Sample for MobiLink” \[MobiLink - Getting Started\]](#).

HotSync with multiple databases

Each UltraLite database on a Palm OS device must have a distinct creator ID to work properly with HotSync. In addition, an application with that creator ID must exist on the Palm OS device.

The HotSync Manager uses each application's creator ID as an identifier to handle synchronization. It hands each properly configured UltraLite application to the MobiLink conduit for synchronization. The conduit searches for and synchronizes a database with the same creator ID as the application.

☞ For information about configuring the conduit, see [“HotSync synchronization overview” \[MobiLink - Client Administration\]](#).

All UltraLite for M-Business Anywhere applications inherit the creator ID of the M-Business client, which is **AvGo**. This inheritance means that only one UltraLite database with creator ID **AvGo** can be synchronized, and that if you assign a distinct creator ID to your database, HotSync will not find it because there is no application with a corresponding creator ID.

This limitation is not an issue for the two sample applications (CustDB and Simple), as they share a common database schema. The only side effect is that when you synchronize the CustDB database, the Simple sample is also synchronized.

☞ For information on resolving this problem, see [“Registering the Palm creator ID” \[UltraLite - C and C++ Programming\]](#).

Connecting to an UltraLite database


UltraLite applications must connect to a database before carrying out operations on the data in it.

Here is the simplest way to establish a connection. Extensions to this technique are given in the following sections.

```
var DatabaseMgr;  
var Connection;  
DatabaseMgr = CreateObject("iAnywhere.UltraLite.DatabaseManager.CustDB");  
Connection = DatabaseMgr.openConnection("dbf=" + DatabaseMgr.directory + "\\  
mydb.udb");
```

Using the Connection object

The following properties of the Connection object govern global application behavior.

 For more information about the Connection object, see [“Class Connection” on page 58](#).

- ◆ **Commit behavior** By default, UltraLite applications are in AutoCommit mode. Each insert, update, or delete statement is committed to the database immediately. Set Connection.AutoCommit to false to build transactions into your application. Turning AutoCommit off and performing commits directly can improve the performance of your application.

For more information, see [“Method commit” on page 60](#).

- ◆ **User authentication** You can change the user ID and password for the application from the default values of DBA and sql by using the grantConnectTo and revokeConnectFrom methods.

For more information, see [“Authenticating users” on page 28](#).

- ◆ **Synchronization** A set of objects governing synchronization are accessed from the Connection object.

For more information, see [“Synchronizing data” on page 29](#).

- ◆ **Tables** UltraLite tables are accessed using the Connection.getTable method.

For more information, see [“Method getTable” on page 62](#).

Maintaining connections and application state across pages

The scope of a JavaScript variable is limited to one web page. Most web applications require multiple pages, and so a mechanism is needed for making some objects persistent across the pages of an application.

UltraLite for M-Business Anywhere provides persistence for the ULTable, ResultSet, and PreparedStatement objects. To make one of these objects persist across pages, supply a **persistent name** as a parameter when creating the object. You can use the persistent name on subsequent pages.

To carry a connection object from page to page, you reopen the connection on each page. One way to do this is to use the reOpen method. Another is to supply an open method on each page, perhaps by including a JavaScript file on each web page to initialize the settings. For examples of how to do this, see the sample files *Samples\UltraLiteForMBusinessAnywhere\CustDB\main.htm* and *Samples\UltraLiteForMBusinessAnywhere\Simple\main_page.htm*.

The requirement to reopen connections across pages provides a security feature for UltraLite applications. You can use it to require that the user confirm some information, perhaps the password, on moving from page to page.

If an UltraLite object is not needed in another web page, the application should issue a close method on the object to save memory.

See also

- ◆ [“Method reOpenConnection” on page 72](#)
- ◆ [“Class PreparedStatement” on page 81](#)
- ◆ [“Class ResultSet” on page 92](#)
- ◆ [“Class ULTable” on page 150](#)
- ◆ [“Class PreparedStatement” on page 81](#)

Persistent names in M-Business Anywhere applications

In HTML, when control transfers to a new page, all handles to allocated JavaScript objects from the old page are lost. For example, in *main.html*, you have a M-Business Anywhere database connection object:

```
conn = dbMgr.openConnection("...");
```

If you click a link in *main.html* and it takes you to a different page (for example: *insert.html*), you cannot find the object named "conn" in *insert.html*. To get the connection object back, you may have to call `dbMgr.openConnection("...")` again. However, you do not have to do this since the connection object is still in memory, you have only lost the JavaScript handle to it.

This is why there is a `persistName` argument in all the M-Business Anywhere API calls to `DataManager`, `Connection`, `ULTable`, `PreparedStatement`, or `ResultSet`. For example, when the M-Business Anywhere runtime receives a call from JavaScript for a UltraLite connection object, M-Business Anywhere first checks to see if a connection object exists in memory that has the same `persistName`. If the runtime can find a matching object, it will return that connection object. Otherwise, M-Business Anywhere goes through the normal procedure to make a new UltraLite database connection and return it.

Using persistent names

There are two types of hierarchy among M-Business Anywhere objects. They both start with `DatabaseManager` and `Connection`:

- ◆ `DatabaseManager` -> `Connection` -> `Table` (for table API)
- ◆ `DatabaseManager` -> `Connection` -> `PreparedStatement` -> `ResultSet` (for Dynamic SQL API)

To retrieve any of these M-Business Anywhere objects with a persistent name, you have to retrieve the top-level object with a persistent name, and then all the upper level M-Business Anywhere objects along the hierarchy tree until the one you want.

For example, if you want to retrieve a existing `ULTable` object from *insert.html*, you need to give a persistent name to `dbMgr`, `conn`, and `table` objects in *main.html*, and then use persistent names in *insert.html* to get all of them back:

Code segment for *main.html*:

```
var dbMgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.simple" );  
// "simple" is the persistent name here. A real database manager object is  
// allocated  
  
var conn = dbMgr.openConnection( "CON=simple_con;..." );  
// "simple_con" is the persistent name here. A real database connection is  
// made.  
  
var custTable = conn.getTable( "ULCustomer", "simpleCustTable" );  
// a real table is allocated
```

Code segment for *insert.html*:

```
var dbMgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.simple" );  
// "simple" is the persistent name here.  
// The allocated database manager object from main.html is returned
```

```
var conn = dbMgr.openConnection( "CON=simple_con;..." );
// "simple_con" is the persistent name here.
// The existing connection object from memory is returned.

var custTable = conn.getTable( "ULCustomer", "simpleCustTable" );
// the existing table object is returned.

var newTable = conn.getTable( "ULOrder", "simpleOrderTable" );
// since there is no order table from main.html,
// it does not exist in memory. A real order table object is allocated.
```

Using persistent names correctly

Put the commonly used code in a JavaScript file. Since most HTML pages of an M-Business Anywhere application need to refer to DatabaseManager, Connection, and some major ULTable objects, it is convenient to put the code that creates them (or retrieves them with a persistent name) in a common JavaScript file, and include this file at the top of HTML pages that use them. Both M-Business Anywhere "simple" and "CustDB" sample programs demonstrate how to do this.

Close the object if you do not plan to use the object from another page. If the M-Business Anywhere application only has one HTML page, then there is no need to have persistent names. The persistent name argument can be set to NULL. On the other hand, if each HTML page has many opened PreparedStatement and ResultSet objects, then the developer needs to balance between the convenience of having them in memory to retrieve them easily with a persistent name from another html page, and wasted memory usage because these objects are always around. For example, suppose you have 5 PreparedStatement objects and 10 ResultSet objects created in *main.html*. They are occupying a significant amount of memory. When the application jumps to *insert.html*, if you only need to refer to some of these objects with a persistent name, then the objects that are not needed anymore are wasting memory. If you try to create new PreparedStatement and ResultSet objects in *insert.html*, you may run out of memory. The solution is to explicitly close those PreparedStatement object or ResultSet object at the end of *main.html* if you are sure you do not need them from *insert.html*.

The state of each M-Business Anywhere object is preserved when it is retrieved with a persistent name. If you have a persistent ULTable object from page 1, when you call openTable method from page 2 using the same persistent name, you get the exact ULtable object back with the same state as the one from page 1. If the cursor is on the nth row of the table when you leave page 1, the cursor will be still on the nth row when you get it back in page 2. It will not be "before first row".

Be careful using the persistent name on ResultSet. When there are place holders on the PreparedStatement, you need to be very careful about whether you want to give a persistent name to the ResultSet. For example, in *main.html* you have the following code:

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" );

OrderStmt.setInt(1, 5000);

var OrderResultSet = OrderStmt.executeQuery( "order_query_result" );
```

Then from *insert.html*, you want the same ResultSet object, you must do the following:

```
var OrderStmt = Connection.prepareStatement(
```

```
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" );

//OrderStmt.setInt(1, 5000); // no need to do this since both the OrderStmt
and
OrderResultSet are retrieve from "cache" without any SQL statement being
actually executed

var OrderResultSet = OrderStmt.executeQuery( "order_query_result" );
```

This OrderResultSet object contains the same result as the "order_id" set to 5000.

However, consider a different situation. You want the same PreparedStatement because you want to do the same query on the Order Table. But you want to query with an order ID other than 5000. In this case, you can assign a persistent name to the PreparedStatement, but you don't need a persistent name on the ResultSet. Since the order id is different this time, the result set will be different from the previous one. In *main.html*, you still do the following:

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" ); // with persistent name

OrderStmt.setInt(1, 5000);

var OrderResultSet = OrderStmt.executeQuery( null ); // notice here, no
persistent name
```

In *insert.html*, you do the following to get a new ResultSet:

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" ); // get the prepared statement from memory with
persistent name

OrderStmt.setInt(1, 6000); // set a different place holder value

var OrderResultSet = OrderStmt.executeQuery( null ); // a real query is
executed
here!
```

In the example above, since the place holder value is different, or some other operation is performed on the Order table that you expect the returned result set will be different, you do not use persistent name on the ResultSet when calling `executeQuery`.

Database encryption and obfuscation

You can encrypt or obfuscate your UltraLite database using UltraLite for M-Business Anywhere.

☞ For more information about database encryption, see [“Security considerations” \[UltraLite - Database Management and Reference\]](#).

Encryption

UltraLite databases may be unencrypted or may employ either encryption or obfuscation. If you want the database to be encrypted or obfuscated that choice must be made when the database is created.

Encryption of an UltraLite database uses extremely strong industry-standard techniques to encrypt the data in the database. The encryption is based on a key phrase that is specified when the database is created. This key phrase must also be supplied when a connection is made to the database.

If an UltraLite database is encrypted, all connections to that database must specify the correct encryption key or the connection attempt fails.

☞ For more information about the EncryptionKey property, see [“Class ConnectionParms” on page 66](#) and [“Method changeEncryptionKey” on page 59](#).

Obfuscation

Obfuscation is a very weak form of encryption that simply masks the data in the database to discourage casual observation of the contents of the database by file or disk viewer programs. To obfuscate the database, set the creationParms.obfuscate boolean value to true. For example:

```
var create_parms = dbMgr.createCreationParms();
create_parms.obfuscate = true;
```

Example

You can change the encryption key by specifying a new encryption key on the Connection object. Before calling the changeEncryptionKey method, the application must make a connection to the encrypted database using the existing encryption key. In the following example code, "apricot" is the new encryption key.

```
conn.changeEncryptionKey("apricot")
```

Working with data using SQL

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

☞ For information about the Table API, see [“Working with data using the table API” on page 20](#).

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

- ◆ Inserting, deleting, and updating rows.
- ◆ Executing a query.
- ◆ Scrolling through the rows of a result set.

☞ This section does not describe the SQL language itself. For information about SQL features, see [SQL Anywhere Server - SQL Reference \[SQL Anywhere Server - SQL Reference\]](#).

Data manipulation: INSERT, UPDATE and DELETE

With UltraLite, you can perform SQL Data Manipulation Language operations and DDL operations. These operations are performed using the ExecuteStatement method, a member of the PreparedStatement class.

☞ For more information the PreparedStatement class, see [“Class PreparedStatement” on page 81](#).

Parameter markers in prepared statements

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

◆ To insert a row

1. Declare a PreparedStatement object.

```
var PrepStmt;
```

2. Assign an INSERT statement to your prepared statement object. In the following, TableName and ColumnName are the names of a table and column.

```
PrepStmt = conn.prepareStatement(  
    "INSERT into TableName(ColumnName) values (?)", null );
```

The null parameter indicates that the statement has no persistent name.

3. Assign parameter values to the statement.

```
var NewValue;  
NewValue = "Bob";  
PrepStmt.setStringParameter(1, NewValue);
```

- Execute the statement.

```
PrepStmt.executeStatement( null );
```

◆ To update a row

- Declare a PreparedStatement object.

```
var PrepStmt;
```

- Assign an UPDATE statement to your prepared statement object. In the following, TableName and ColumnName are the names of a table and column.

```
PrepStmt = conn.prepareStatement(
    "UPDATE TableName SET ColumnName = ? WHERE ID = ?", null);
```

The null parameter indicates that the statement has no persistent name.

- Assign parameter values to the statement using methods appropriate for the data type.

```
var NewValue;
NewValue = "Bob";
PrepStmt.setStringParameter(1, NewValue);
PrepStmt.setIntParameter(2, 6);
```

- Execute the statement

```
PrepStmt.executeStatement( );
```

◆ To delete a row

- Declare a PreparedStatement object.

```
var PrepStmt;
```

- Assign a DELETE statement to your prepared statement object.

```
PrepStmt = conn.prepareStatement(
    "DELETE FROM customer WHERE ID = ?", null );
```

The null parameter indicates that the statement has no persistent name.

- Assign parameter values for the statement.

```
var IDValue;
IDValue = 6;
PrepStmt.setIntParameter( 1, IDValue );
```

- Execute the statement.

```
PrepStmt.executeStatement( );
```

Data retrieval: SELECT

When you execute a SELECT statement, the PreparedStatement.executeQuery method returns a ResultSet object. The ResultSet class contains methods for navigating within a result set and methods to update data using the ResultSet.

☞ For more information about ResultSet objects, see [“Class ResultSet” on page 92](#).

Example

In the following code, the results of a query are accessed as a ResultSet. When first assigned, the ResultSet is positioned before the first row. The ResultSet.moveFirst method is then called to navigate to the first record in the result set.

```
var MyResultSet;
var PrepStmt;
PrepStmt = conn.prepareStatement("SELECT ID, Name FROM customer", null );
MyResultSet = PrepStmt.executeQuery( null );
MyResultSet.moveFirst();
```

Example

The following code demonstrates how to obtain column values for the current row. The example uses character data; similar methods are available for other data types.

The getString method uses the following syntax: MyResultSetName.getString(*Index*) where *Index* is the ordinal position of the column name in your SELECT statement.

```
if ( MyResultSet.getRowCount() == 0 ) {
} else {
    alert( MyResultSet.getString(1) );
    alert( MyResultSet.getString(2) );
    MyResultSet.moveRelative(0);
}
```

☞ For more information about navigating a result set, see [“Navigation with SQL” on page 19](#).

The following procedure uses a SELECT statement to retrieve information from the database. The results of the query are assigned to a ResultSet object.

◆ To perform a select statement

1. Declare a PreparedStatement object.

```
var OrderStmt;
```

2. Assign a prepared statement to your PreparedStatement object.

```
OrderStmt = Connection.prepareStatement(
    "SELECT order_id, disc, quant, notes, status, c.cust_id,
    cust_name, p.prod_id, prod_name, price
    FROM ULOrder o, ULCustomer c, ULProduct p
    WHERE o.cust_id = c.cust_id
    AND o.prod_id = p.prod_id
    ORDER BY der_id", "order_query_stmt" );
```

The second parameter is a persistent name that provides cross-page JavaScript object persistence.

3. Execute the query.

```
OrderResultSet = OrderStmt.executeQuery( "order_query" );
```


☞ For more information on how to use queries, see the CustDB sample code in *samples-dir\UltraLiteForMBusinessAnywhere\CustDB\custdb.js*.

Navigation with SQL

UltraLite for M-Business Anywhere provides you with a number of methods to navigate a result set to perform a wide range of navigation tasks.

The following methods of the ResultSet object allow you to navigate your result set:

- ◆ **moveAfterLast** moves to a position after the last row.
- ◆ **moveBeforeFirst** moves to a position before the first row.
- ◆ **moveFirst** moves to the first row.
- ◆ **moveLast** moves to the last row.
- ◆ **moveNext** moves to the next row.
- ◆ **movePrevious** moves to the previous row.
- ◆ **moveRelative** moves a certain number of rows relative to the current row. Positive index values move forward in the result set, negative index values move backward in the result set, and zero does not move the cursor. Zero is useful if you want to repopulate a row buffer.

Example

The following code fragment demonstrates how to use the moveFirst method to navigate within a result set.

```
PrepStmt = conn.prepareStatement(
    "SELECT ID, Name FROM customer", null );
MyResultSet = PrepStmt.executeQuery( null );
MyResultSet.moveFirst();
```

The same technique is used for all of the move methods.

☞ For more information about these navigational methods, see [“Class ResultSet” on page 92](#).

The ResultSetSchema object

The ResultSet.schema property allows you to retrieve information about the columns in the query.

The following example demonstrates how to use ResultSetSchema to capture schema information.

```
var i;
var MySchema = rs.schema ;
for ( i = 1; i <= MySchema.columnCount; i++) {
    colname = MySchema.getColumnName(i);
    coltype = MySchema.getColumnSQLType(colname).toString();
    alert ( colname + " " + coltype );
}
```

Working with data using the table API

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

 For information about SQL, see [“Working with data using SQL” on page 16](#).

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.

Navigation with the Table API

UltraLite for M-Business Anywhere provides you with a number of methods to navigate a table to perform a wide range of navigation tasks.

The following methods of the ULTable object allow you to navigate your result set:

- ◆ **moveAfterLast** moves to a position after the last row.
- ◆ **moveBeforeFirst** moves to a position before the first row.
- ◆ **moveFirst** moves to the first row.
- ◆ **moveLast** moves to the last row.
- ◆ **moveNext** moves to the next row.
- ◆ **movePrevious** moves to the previous row.
- ◆ **moveRelative** moves a certain number of rows relative to the current row. Positive index values move forward in the table, negative index values move backward in the table, and zero does not move the cursor. Zero is useful if you want to repopulate a row buffer.

Example

The following code opens the customer table and scrolls through its rows. It then displays an alert with the last name of each customer.

```
var tCustomer;
tCustomer = conn.getTable( "customer", null );
tCustomer.open();
tCustomer.moveBeforeFirst();
While (tCustomer.moveToNext()) {
    alert( tCustomer.getString(3) );
}
```

Specifying an index

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

Example

The following code moves to the first row of the customer table as ordered by the ix_name index.

```
tCustomer = conn.getTable("customer", null );
tCustomer.openWithIndex("ix_name");
tCustomer.moveFirst();
```

Accessing the values of the current row

At any time, a ULTable object is positioned at one of the following places.

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

If the ULTable object is positioned on a row, you can use one of the ULTable get methods to get the value of each column for the current row.

Example

The following code fragment retrieves the value of three columns from the tCustomer ULTable object, and displays them in text boxes.

```
var colID, colFirstName, colLastName;
colID = tCustomer.schema.getColumnID( "id" );
colFirstName = tCustomer.schema.getColumnID( "fname" );
colLastName = tCustomer.schema.getColumnID( "lname" );
alert( tCustomer.getInt( colID ) );
alert( tCustomer.getString( colFirstName ) );
alert( tCustomer.getString( colLastName ) );
```

You can also use methods of ULTable to set values.

```
tCustomer.setString( colLastName, "Kaminski" );
```

By assigning values to these properties you do not alter the value of 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. You cannot, however, get values from the column. For example, the following code fragment generates an error.

```
tCustomer.moveBeforeFirst();
id = tCustomer.getInt( colID );
```

Searching rows with find and lookup

UltraLite has several modes of operation for working with data. Two of these modes, the find and lookup modes, are used for searching. The ULTable object has methods corresponding to these modes for locating particular rows in a table.

Note

The columns searched using Find and Lookup methods must be in the index used to open the table.

- ◆ **Find methods** move to the first row that exactly matches a specified search value, under the sort order specified when the ULTable object was opened.

For more information about find methods, see [“Class ULTable” on page 150](#).

- ◆ **Lookup methods** move to the first row that matches or is greater than a specified search value, under the sort order specified when the ULTable object was opened.

For more information about lookup methods, see [“Class ULTable” on page 150](#).

- ◆ **To search for a row**

1. Enter find or lookup mode.

Call the FindBegin or LookupBegin method. For example, the following code fragment calls ULTable.findBegin.

```
tCustomer.findBegin();
```

2. Set the search values.

You do this by setting values in the current row. Setting these values affects the buffer, not the database. For example, the following code fragment sets the last name column in the buffer to Kaminski.

```
tCustomer.setString(3, "Kaminski" );
```

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

3. Search for the row.

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

```
tCustomer.findFirst();
```

Inserting, updating, and deleting rows

UltraLite exposes the rows in a table to your application one at a time. The ULTable object has a current position, which may be on a row, before the first row, or after the last row of the table.

When your application changes location, UltraLite makes a copy of the row in a buffer. Any operations to get or set values affect only the copy of data in this buffer. They do not affect the data in the database.

Example

The following statement changes the value of the first column in the buffer to 3.

```
tCustomer.setInt( 1 , 3 );
```

Using UltraLite modes

The UltraLite mode determines the purpose for which the values in the buffer will be used. UltraLite has the following four modes of operation, in addition to a default mode.

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

◆ To update a row

1. Move to the row you want to update.

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

2. Enter Update mode.

For example, the following instruction enters Update mode on the table `tCustomer`.

```
tCustomer.updateBegin();
```

3. Set the new values for the row to be updated.

For example, the following instruction sets the new value to Elizabeth.

```
tCustomer.setString( 2, "Elizabeth" );
```

4. Execute the Update.

```
tCustomer.update();
```

After the update operation, the current row is the row that was just updated. If you changed the value of a column in the index specified when the `ULTable` object was opened, the current position is undefined.

By default, UltraLite operates in AutoCommit mode, so that the update is immediately applied to the row in permanent storage. If you have disabled AutoCommit mode, the update is not applied until you execute a commit operation. For more information, see [“Transaction processing in UltraLite” on page 25](#).

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 similar to those for updating rows, except that there is no need to locate any particular row in the table before carrying out the insert operation. Rows are automatically sorted by the index specified when opening the table.

◆ To insert a row

1. Enter Insert mode.

For example, the following instruction enters Insert mode on the table CustomerTable.

```
tCustomer.insertBegin();
```

2. Set the values for the new row.

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, NULL is used. If the column does not allow NULL, the following defaults are used:

- ◆ For numeric columns, zero.
- ◆ For character columns, an empty string.

To set a value to NULL explicitly, use the setNull method.

```
colID = tCustomer.schema.getColumnID( "id" );  
colFirstName = tCustomer.schema.getColumnID( "fname" );  
colLastName = tCustomer.schema.getColumnID( "lname" );  
tCustomer.setInt( colID, 42 );  
tCustomer.setString( colFirstName, "Mitch" );  
tCustomer.setString( colLastName, "McLeod" );
```

3. Execute the insertion.

The inserted row is permanently saved to the database when a Commit is carried out. In AutoCommit mode, a Commit is carried out as part of the Insert method.

```
tCustomer.insert();
```

Deleting rows

There is no delete mode corresponding to the insert or update modes.

The following procedure deletes a row.

◆ To delete a row

1. Move to the row you want to delete.
2. Execute the delete:

```
tCustomer.deleteRow();
```

Working with BLOB data

You can fetch BLOB data for columns declared `BINARY` or `LONG BINARY` using the `GetByteChunk` method.

☞ For more information, see [“Method `getStringChunk`” on page 159](#).

Transaction processing in UltraLite

UltraLite provides transaction processing to ensure the integrity of the data in your database. A transaction is a logical unit of work. Either the entire transaction is executed, or none of the statements in the transaction are executed.

By default, UltraLite operates in `AutoCommit` mode. In `AutoCommit` mode, each insert, update, or delete is executed as a separate transaction. Once the operation is completed, the change is made to the database.

If you set the `Connection.AutoCommit` property to `false`, you can use multi-statement transactions. For example, if your application transfers money between two accounts, the deduction from the source account and the addition to the destination account constitute a single transaction.

If `AutoCommit` is `false`, you must execute a `Connection.commit` statement to complete a transaction and make changes to your database permanent, or you may execute a `Connection.rollback` statement to cancel all the operations of a transaction. Turning `AutoCommit` off improves performance.

Note

Synchronization causes a `Commit` even if you have `AutoCommit` set to `False`.

Accessing schema information

Each Connection, ULTable, and ResultSet object contains a schema property. These schema objects provide information about the tables, columns, indexes, and publications in a database.

- ◆ **DatabaseSchema** 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, access the Connection.databaseSchema property.

- ◆ **TableSchema** The number and names of columns in the table, as well as the Indexes collections for the table.

To obtain a TableSchema object, access the ULTable.schema property.

- ◆ **IndexSchema** Information about the column, or columns, in the index. As an index has no data directly associated with it, there is no separate Index object, only a IndexSchema object.

The IndexSchema objects are accessed using the TableSchema.getIndex method.

- ◆ **PublicationSchema** The numbers and names of tables and columns contained in a publication. Publications are also comprised of schema only, so there is a PublicationSchema object but no Publication object.

The PublicationSchema objects are accessible using the DatabaseSchema.getPublicationSchema method.

- ◆ **ResultSetSchema** The number and names of the columns in a result set.

The ResultSetSchema objects are accessible using the PreparedStatement.getResultSetSchema method or the ResultSet.schema property.

Handling errors

In normal operation, UltraLite for M-Business Anywhere can throw errors that are intended to be caught and handled in the script environment. Errors are expressed as SQLCODE values, negative numbers indicating the particular kind of error.

☞ For a list of error codes thrown by UltraLite for M-Business Anywhere, see [“Class SQLException” on page 112](#).

UltraLite for M-Business Anywhere throws errors only from the DatabaseManager and Connection objects. The following methods of DatabaseManager can throw errors.

- ◆ createDatabase
- ◆ dropDatabase
- ◆ openConnection

All other errors and exceptions within UltraLite for M-Business Anywhere are routed through the Connection object.

☞ For more information about accessing error numbers from DatabaseManager and Connection objects, see [“Class Connection” on page 58](#) and [“Class DatabaseManager” on page 70](#).

Authenticating users

New users have to be added from an existing connection. As all UltraLite databases are created with a default user ID and password of DBA and sql, respectively, you must first connect as this initial user.

You cannot change a user ID: you add a user and delete an existing user. A maximum of four user IDs are permitted for each UltraLite database.

☞ For more information about granting or revoking connection authority, see [“Method grantConnectTo” on page 62](#) and [“Method revokeConnectFrom” on page 63](#).

◆ To add a user or change the password for an existing user

1. Connect to the database as an existing user.
2. Grant the user connection authority with the desired password.

```
conn.grantConnectTo("Robert", "newPassword");
```

◆ To delete an existing user

1. Connect to the database as an existing user.
2. Revoke the user's connection authority as follows.

```
conn.revokeConnectFrom("Robert");
```

Synchronizing data

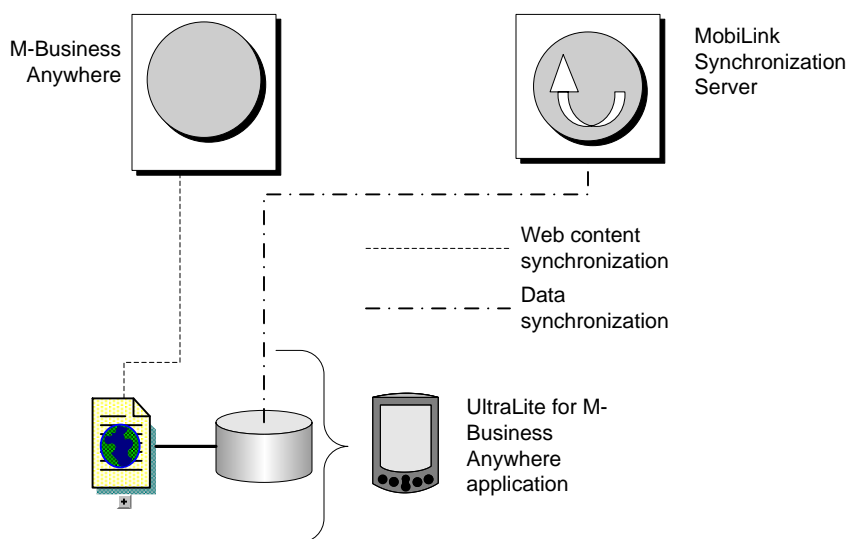
UltraLite for M-Business Anywhere applications typically involve two kinds of synchronization:

- ◆ **Web content synchronization** Web content, including HTML pages that define the application itself, is synchronized through M-Business Anywhere.
- ◆ **Data synchronization** The UltraLite database is synchronized with a MobiLink server.

Although these two kinds of synchronization are distinct, you can initiate them together in a technique called **one-button synchronization**. One-button synchronization is the recommended model for most applications, but as there may be cases where it is necessary to keep synchronization of data and web content entirely separate, that technique is discussed below.

One-button synchronization

One-button synchronization is a technique for initiating web content synchronization (using M-Business Anywhere) and UltraLite data synchronization (using MobiLink) in a single operation. It is available on Windows CE and Windows XP only. The architecture of one-button synchronization is as follows:



The sequence of events in one button synchronization is as follows:

1. The user synchronizes their web application, perhaps by placing it in the cradle.
2. The M-Business Client synchronizes the web content.
3. The MBConnect component of M-Business Client calls the *ulconnect.exe* application.
4. *ulconnect.exe* initiates synchronization of the UltraLite database.
5. Data is synchronized with MobiLink.

To implement one-button synchronization you must carry out the following steps:

1. In your application, set the synchronization parameters for MobiLink synchronization.

If you are synchronizing through M-Business Anywhere you can use the `SyncParms.setMBAserver` method to set the host and port synchronization parameters. For more information, see [“Method setMBAserver” on page 128](#).

Otherwise, use the standard methods to set synchronization parameters. For more information, see [“Class SyncParms” on page 122](#).

2. Save the synchronization parameters so that they can be read by `ulconnect.exe`.

Call the `Connection.saveSyncParms` method to save the synchronization parameters. For more information, see [“Method saveSyncParms” on page 64](#).

Synchronizing data

For most users it is useful to use one-button synchronization, which initiates both data synchronization and web content synchronization. For more information, see [“One-button synchronization” on page 29](#).

This section is for those users who want to synchronize data separately from web content synchronization.

Synchronization requires the MobiLink server and appropriate licensing. You can find a working example of synchronization in the CustDB sample application.

UltraLite for M-Business Anywhere supports TCP/IP, HTTP, HTTPS, and HotSync synchronization. Synchronization is initiated by the UltraLite application. In all cases, you use methods and properties of the `Connection` object to control synchronization.

Separately licensed component required


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

See [“Separately licensed components” \[SQL Anywhere 10 - Introduction\]](#).

◆ To synchronize over TCP/IP or HTTP

1. Prepare the synchronization information.

Assign values to the required properties of the `Connection.syncParms` object.

 For information about the properties and the values that you should set, see [“UltraLite Clients” \[MobiLink - Client Administration\]](#).

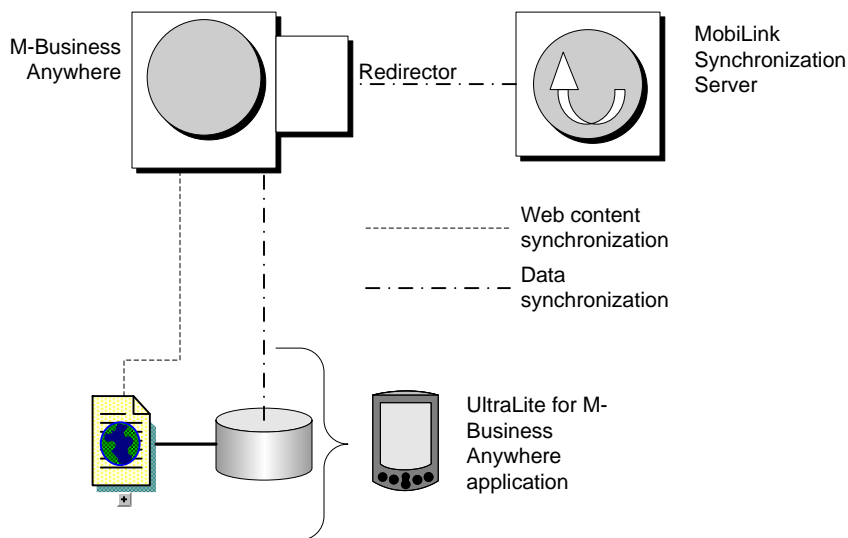
2. Synchronize.

Call the `Connection.synchronize` method.

Synchronizing data via M-Business Anywhere

Whether you use one-button synchronization or separate data synchronization, you can use a MobiLink Redirector to configure your M-Business Anywhere server to route data to and from a MobiLink server. For synchronization from outside the firewall, this reduces the number of ports that need to be externally accessible.

The following diagram illustrates the architecture for the case of one-button synchronization.



◆ To synchronize data via M-Business Anywhere

1. At the server side, set up a MobiLink Redirector to route data between M-Business Anywhere and your MobiLink server.

☞ For information on the MobiLink Redirector for M-Business Anywhere, see [“M-Business Anywhere Redirector” \[MobiLink - Server Administration\]](#).

2. In your client, set synchronization parameters so that UltraLite synchronization is directed to the host and port number of M-Business Anywhere. You can use the `SyncParms.setMBAserver` method to carry out this task.

☞ For more information, see [“Method setMBAserver” on page 128](#).

3. From a client application, initiate synchronization using either one-button synchronization or separate data synchronization.

☞ For more information, see [“One-button synchronization” on page 29](#), and [“Synchronizing data” on page 30](#).

Deploying UltraLite for M-Business Anywhere applications

When you have completed your application or when you want to test your application, you need to deploy it to a device. This section outlines the steps needed to deploy an UltraLite application to a device.

Deploying applications to Windows CE and Windows XP

You must carry out the following steps to deploy an UltraLite application to a Windows CE device:

- ◆ Deploy your application and UltraLite component.

For instructions, see [“UltraLite for M-Business Anywhere Quick Start” on page 6](#).

- ◆ Deploy an initial copy of the UltraLite database.

For instructions, see [“UltraLite for M-Business Anywhere Quick Start” on page 6](#).

In many situations it is sufficient to deploy an UltraLite database. You can use synchronization to load an initial copy of the data.

You must place the database so that it can be located by the application. The Database On CE connection parameter defines the location for Windows CE. The Database on Desktop connection parameter defines the location for Windows XP.

See [“CE_FILE connection parameter” \[UltraLite - Database Management and Reference\]](#), and [“DBF connection parameter” \[UltraLite - Database Management and Reference\]](#).

Deploying applications that use one-button synchronization

One-button synchronization requires a set of files, including: *ulconnect.exe*, *ulconnect.udb*, *ulpod10.dll*, and *ulrt10.dll*. For Windows CE, these files are located in file *ulpod.cab* in the directory *install-dir\ultralite\UltraLiteForMBusinessAnywhere\ce\arm*. When you deploy the cab file to a Windows CE device, it installs its contents in the proper locations automatically. For Windows XP, the required files must be deployed manually from the directory *install-dir\ultralite\UltraLiteForMBusinessAnywhere\win32\386*.

Deploying applications to Palm OS

You must carry out the following steps to deploy an UltraLite application to a Palm OS device:

- ◆ Deploy your application and UltraLite component.

For instructions, see [“UltraLite for M-Business Anywhere Quick Start” on page 6](#).

- ◆ Deploy an initial copy of the UltraLite database.

For instructions, see [“UltraLite for M-Business Anywhere Quick Start” on page 6](#).

In many situations it is sufficient to deploy an appropriately initialized UltraLite database file only. You can then use synchronization to load an initial copy of the data.

You can create *.pdb* files for deployment to Palm OS from any of the UltraLite utilities, including *ulxml* and *ulinit*.

You must supply a database using the correct creator ID, so that it can be located by your application. The Database On Palm connection parameter uses the creator ID to find the database.

See “PALM_FILE connection parameter” [*UltraLite - Database Management and Reference*].

- ◆ Deploy the MobiLink synchronization conduit for HotSync.

This step is required only if the application is synchronizing using HotSync.

For instructions, see “Using HotSync on Palm OS” [*MobiLink - Client Administration*].

CHAPTER 3

Tutorial: A Sample Application for M-Business Anywhere

Contents

Introduction	36
Lesson 1: Create project architecture	37
Lesson 2: Create the application files	39
Lesson 3: Set up the M-Business Anywhere Server and Client	41
Lesson 4: Add startup code to your application	43
Lesson 5: Add inserts to your application	46
Lesson 6: Add navigation to your application	49
Lesson 7: Add updates and deletes to your application	50
Lesson 8: Add synchronization to your application	52

About this chapter

This tutorial guides you through the process of building a sample UltraLite application for M-Business Anywhere.

Introduction

This tutorial describes how to build a cross-platform UltraLite application. At the end of the tutorial you will have an application and small database that synchronizes with a central consolidated database.

Timing

The tutorial takes about 30 minutes if you copy and paste the code. If you enter the code yourself, it takes significantly longer.

Prerequisites

This tutorial assumes that you have M-Business Anywhere installed on your computer and that your machine has a web server that you can use to deliver files.

You must also have access to an M-Business Client to test and run the application.

The tutorial assumes a basic familiarity with JavaScript programming language and M-Business Anywhere application development.

The tutorial also assumes that you know how to create an UltraLite database using either UltraLite in Sybase Central, or the ulcreate UltraLite utility. For more information, see [“Creating an UltraLite database from Sybase Central”](#) [*UltraLite - Database Management and Reference*] and [“Creating and Configuring UltraLite Databases”](#) [*UltraLite - Database Management and Reference*].

Lesson 1: Create project architecture

The first lesson describes how to set up the project architecture and creating an UltraLite database for the tutorial.

◆ To create project architecture and empty UltraLite database

1. Create a directory for this tutorial.

This tutorial assumes the directory is `c:\Tutorial\mbus`. If you create a directory with a different name, use that directory throughout the tutorial.

Create the following subdirectories for platform-specific files:

- ◆ `c:\Tutorial\mbus\PALM_OS`
- ◆ `c:\Tutorial\mbus\WIN32_OS`
- ◆ `c:\Tutorial\mbus\WINCE_OS`
- ◆ `c:\Tutorial\mbus\WINCE_OS\arm`

2. Configure your web server:

- a. Map a virtual directory named `tutorial` on your web server to `c:\Tutorial\mbus`. The URL to access this directory will be `http://localhost/tutorial`.

For Microsoft IIS, you can make these changes from the management tool.

For Apache, make a symbolic link named `tutorial` from your document root to the `c:\Tutorial\mbus` directory, or copy the tutorial files into your Apache document root.

- b. Ensure that your web server delivers the following files with MIME type `application/octet-stream`:

- ◆ `.cab`
- ◆ `.dll`
- ◆ `.prc`
- ◆ `.pdb`
- ◆ `.udb`

For Microsoft IIS, you can make these changes from the management tool. Go to the virtual directory properties and make the changes under HTTP Headers and File Types.

For Apache, edit the file `mime.types` in your `conf` directory.

3. Create a database using UltraLite in Sybase Central.

☞ For more information about creating a database, see “[Creating an UltraLite database from Sybase Central](#)” [*UltraLite - Database Management and Reference*].

- ◆ **Table name** Customer

- ◆ **Columns in Customer**

Column Name	Data Type (Size)	Column allows NULL values?	Default value
ID	integer	No	autoincrement
FName	char(15)	No	None
LName	char(20)	No	None
City	char(20)	Yes	None
Phone	char(12)	Yes	555-1234

It is usually better to use global autoincrement or UUID values for primary keys in a synchronizing environment. The autoincrement default is used here to keep the tutorial shorter.

◆ **Primary key** Ascending ID

4. Save the database.

If you are developing an application for Windows or Windows CE, choose File ► Save and choose *tutcustomer.udb* in the *WINCE_OS* or the *WIN32_OS* subdirectory of your tutorial directory as the file name.

If you are developing an application for Palm OS:

- a. From the File menu, choose Export Schema for Palm.
- b. Enter Syb3 as the creator ID.
- c. Save the file as *tutcustomer.pdb* in the *PALM_OS* subdirectory of your tutorial directory.

A note on Palm creator IDs

The creator ID is assigned to you by Palm. You can use Syb3 as your creator ID when you make sample applications. However, when you create a commercial application, you should use your own creator ID.

If you are developing a cross-platform application, save the database file in all the above locations.

Lesson 2: Create the application files

The following procedure uses the form to create a user interface. This example uses text boxes for input and output.

◆ Create the application files

1. Create the file `c:\Tutorial\mbus\main.html`.

This file will be the main file of the application. Later in the tutorial, you will add content to the file. For now, you just set it up to include a platform-specific file `ul_deps.html`. Add the following content to the file:

```
<html>
<body>
<a href="AG_DEVICEOS/ul_deps.html"></a>
</body>
</html>
```

2. Create the platform-specific files.

Each of these files references the appropriate UltraLite runtime library and database file. Create a file `ul_deps.html` in each of the operating system subdirectories of your tutorial directory, as follows:

```
<!-- PALM_OS\ul_deps.html -->
<html>
  <a href="ulpod10.prc"></a>
  <a href="tutCustomer.pdb"></a>
</html>

<!-- WINCE_OS\ul_deps.html -->
<html>
  <a href="AG_DEVICEPROCESSOR/ulpod10.dll"></a>
  <a href="tutCustomer.udb"></a>
</html>

<!-- WIN32_OS\ul_deps.html -->
<html>
  <a href="ulpod10.dll"></a>
  <a href="tutCustomer.udb"></a>
</html>
```

3. Copy the UltraLite runtime files to the tutorial directory.

The `ul_deps.html` files require that the UltraLite runtime library and database be in the proper location relative to the tutorial directory. The database file is already in place from earlier in the tutorial. You must now copy the UltraLite runtime library into place.

- ◆ For the Palm OS, copy `ulpod10.prc` from `install-dir\ultralite\UltraLiteForMBusinessAnywhere\palm\68k` to `c:\Tutorial\mbus\PALM_OS\`.
- ◆ For Windows CE, copy `ulpod10.dll` from `install-dir\ultralite\UltraLiteForMBusinessAnywhere\ce\arm` to `c:\Tutorial\mbus\WINCE_OS\arm\`.
- ◆ For Windows XP, copy `ulpod10.dll` from `install-dir\ultralite\UltraLiteForMBusinessAnywhere\win32\386` to `c:\Tutorial\mbus\WIN32_OS\`.

All application files are now in place.

Lesson 3: Set up the M-Business Anywhere Server and Client

In this lesson you create an M-Business Anywhere user, group, and channel for your application. This information is for M-Business Anywhere 6.0.

◆ Configure M-Business Anywhere

1. Open the M-Business Anywhere administration console and login as the admin user.

The default user ID is **Admin**, with an empty password.

2. Create a new user:

Later in this tutorial, you will use the user name and password you create in this step to synchronize from an M-Business client. If you already have an M-Business client set up for this server, you may want to use a user name that already exists.

- a. Click the Users menu option in the left navigation panel and then click the Create User link. The Create New User page appears.
- b. Enter a User Name and enter the same password in the Password and Confirm Password fields. The other fields are optional. Click Create.

3. Create a group and a channel:

- a. Click the Groups heading and click New Group.
- b. Name the new group **UltraLite Samples** and click Create and Edit.
- c. Under the Web tab, click New Group Channel.
- d. Use the following settings for the channel. Make sure to substitute the correct URL for your web server:

◆ **Title** UltraLite Tutorial

◆ **Location** *http://localhost:8091/tutorial/main.html*.

The location is the URL of the tutorial *main.html* page, as served by your web server.

◆ **Channel Size Limit** 1000 KB

◆ **Link Depth** 3

◆ **Allow Binary Distribution** Yes (checked).

◆ **Hidden** No (unchecked)

4. Add the user to the group:

- a. Click the Users heading and find the user you created in step 2.
- b. Click the User Name to show the user's properties.

- c. Click Add/Remove Groups.
- d. Check the **_UltraLite Samples_** group and click Update to add the user to this group.

The user, group, and channel are now set up on M-Business Anywhere. The next step is to synchronize the content of this channel to an M-Business client. You can do this from whichever platform you want to use.

The next procedure assumes that you have an M-Business client installed. It is recommended that you click Tools ► Options and set the client options to Show JavaScript Errors. This setting allows easier debugging of any errors in your application.

◆ **Synchronize the channel for your device**

- Synchronize your M-Business client with the UltraLite channel on the M-Business Anywhere Server.

At this stage there is no content for your application, so the page appears blank.

Lesson 4: Add startup code to your application

In this lesson you add startup code to your application that connects to an UltraLite database. This requires adding HTML to the main page, and adding JavaScript logic to control the application.

◆ Add content to your application

1. Add content to *main.html*.

Add the following form to your application's main page, *c:\Tutorial\mbus\main.html*, immediately after the `<a>` tag:

```
<form name="custForm">
<input type="text" name="fname" size="15"><br>
<input type="text" name="lname" size="20"><br>
<input type="text" name="city" size="20"><br>
<input type="text" name="phone" size="12"><br>
<input type="text" name="custid" size="10"><br>
<br><br>

<table>
<tr>
<td><input type="button"
value="Insert" onclick="ClickInsert();"
</td>
<td><input type="button"
value="Update" onclick="ClickUpdate();"
</td>
<td>
<input type="button"
value="Delete" onclick="ClickDelete();"
</td>
</tr>

<tr>
<td>
<input type="button"
value="Next" onclick="ClickNext();"
</td>
<td>
<input type="button"
value="Prev" onclick="ClickPrev();"
</td>
<td></td>
</tr>

<tr>
<td colspan=3>
<input type="button"
value="Synchronize" onclick="ClickSync();"
</td>
</tr>
</table>
</form>
```

2. Create a JavaScript file *c:\Tutorial\mbus\tutorial.js* that provides application logic.
3. Add content to the JavaScript file:

Add the following code to the top of the file to declare the required UltraLite objects:

```
var DatabaseMgr;  
var Connection;  
var CustomerTable;
```

Add connection code:

```
function Connect()  
{  
  DatabaseMgr = CreateObject  
( "iAnywhere.Data.UltraLite.DatabaseManager.Tutorial" );  
  if( DatabaseMgr == null ) {  
    alert( "Error, make sure POD is on device" );  
    return;  
  }  
  
  var connParms = DatabaseMgr.createConnectionParms();  
  var dir = DatabaseMgr.directory;  
  
  connParms.schemaOnPalm = "tutCustomer";  
  connParms.databaseOnPalm = "Syb3";  
  
  connParms.databaseOnCE = dir + "\\tutCustomer.udb";  
  connParms.databaseOnDesktop = dir + "\\tutCustomer.udb";  
  
  connParms.userID = "DBA";  
  connParms.password = "sql";  
  
  try {  
    // try to connect to an existing database  
    Connection = DatabaseMgr.openConnection( connParms.ToString() );  
    alert("Connected to an existing database");  
  } catch( ex ) {  
    if( DatabaseMgr.sqlCode !=  
DatabaseMgr.SQLError.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {  
      alert( ex.getMessage() );  
      return;  
    } else {  
      try {  
        // the database does not exist, create one  
        Connection = DatabaseMgr.createDatabase( connParms.ToString() );  
        alert("Created a new database");  
      } catch( ex2 ) {  
        alert( ex2.getMessage() );  
        return;  
      }  
    }  
  }  
}
```

4. Use the onload event handler to connect to the database when the application is started:
 - a. Import *tutorial.js* into *main.html* by adding the following line immediately before the <body> tag:

```
<script src="tutorial.js"></script>
```
 - b. Edit *main.html* and change the <body> tag to the following:

```
<body onload="Connect();">
```
5. Test your application.

Synchronize your M-Business Client and start the application. A message box appears when your application creates the UltraLite database. Once this is working properly, you can continue to the next lesson.

Lesson 5: Add inserts to your application

This lesson shows how to fill out your application with data manipulation and navigation logic.

◆ Open the table

1. Write code to initialize the CustomerTable that represents the Customer table in your database.

Add the following code to the end of the Connect function in *tutorial.js*:

```
try {
    CustomerTable = Connection.getTable( "customer", null );
    CustomerTable.open();
} catch( ex3 ) {
    alert("Error: " + ex3.getMessage() );
}
```

2. Add variables to move data between the database and the web form.

Add the following declarations to the top of *tutorial.js*, before the Connect function.

```
var Cust_FName = "";
var Cust_LName = "";
var Cust_City = "";
var Cust_Phone = "";
var Cust_Id = "";
```

3. Create procedures to fetch and display customer data.

Add the following function to *tutorial.js*, immediately after the Connect function. It fetches the current row of the customer and also ensures that NULL columns display as empty strings:

```
function FetchCurrentRow()
{
    var id;
    if( CustomerTable.getRowCount() == 0 ) {
        Cust_FName = "";
        Cust_LName = "";
        Cust_City = "";
        Cust_Phone = "";
        Cust_Id = "";
    } else {
        id = CustomerTable.schema.getColumnID( "FName" );
        Cust_FName = CustomerTable.getString(id);
        id = CustomerTable.schema.getColumnID( "LName" );
        Cust_LName = CustomerTable.getString(id);
        id = CustomerTable.schema.getColumnID( "city" );
        if( CustomerTable.isNull(id) ) {
            Cust_City = "";
        } else {
            Cust_City = CustomerTable.getString(id);
        }
        id = CustomerTable.schema.getColumnID( "phone" );
        if( CustomerTable.isNull(id) ) {
            Cust_Phone = "";
        } else {
            Cust_Phone = CustomerTable.getString(id);
        }
    }

    id = CustomerTable.schema.getColumnID( "id" );
    Cust_Id = CustomerTable.getString(id);
}
```

```

    }
}

```

Add the following JavaScript to *main.html*, immediately before the closing `</body>` tag. `DisplayCurrentRow` takes the values from the database and displays them in the web form. `FetchForm` takes the current values in the web form and makes them available to the database code.

```

<script>
function DisplayCurrentRow()
{
    FetchCurrentRow();
    document.custForm.fname.value = Cust_FName;
    document.custForm.lname.value = Cust_LName;
    document.custForm.city.value = Cust_City;
    document.custForm.phone.value = Cust_Phone;
    document.custForm.custid.value = Cust_Id;
}

function FetchForm()
{
    Cust_FName = document.custForm.fname.value;
    Cust_LName = document.custForm.lname.value;
    Cust_City = document.custForm.city.value;
    Cust_Phone = document.custForm.phone.value;
}
</script>

```

4. Call `DisplayCurrentRow` when the application is loaded.

Modify the body tag at the top of *main.html* as follows:

```

<body onload="Connect(); DisplayCurrentRow();">

```

Although there is no data in your database and no rows are displayed, this is a good place to synchronize M-Business Client to ensure that you have not introduced bugs.

◆ Add code to insert rows

- Write code to implement the Insert button.

In the following procedure, the call to `InsertBegin` puts the application into insert mode and sets all values in the current row to their defaults. For example, the ID column receives the next autoincrement value. The column values are set and the new row is inserted.

Add the following function to *tutorial.js*, immediately after `FetchCurrentRow`:

```

function Insert()
{
    var id;

    try {
        CustomerTable.insertBegin();
        id = CustomerTable.schema.getColumnID("FName");
        CustomerTable.setString(id, Cust_FName);
        id = CustomerTable.schema.getColumnID("LName");
        CustomerTable.setString(id, Cust_LName);
        id = CustomerTable.schema.getColumnID("city");
        if( Cust_City.length > 0 ) {
            CustomerTable.setString(id, Cust_City);
        }
    }
}

```

```
        id = CustomerTable.schema.getColumnID("phone");
        if( Cust_Phone.length > 0 ) {
            CustomerTable.setString(id, Cust_Phone);
        }
        CustomerTable.insert();
        CustomerTable.moveLast();
    } catch( ex ) {
        alert( "Insert error: " + ex.getMessage() );
    }
}
```

Add the following function to *main.html*, immediately after the *FetchForm* function:

```
function ClickInsert()
{
    FetchForm();
    Insert();
    DisplayCurrentRow();
}
```

You can now test your application.

◆ Test your application

1. Synchronize your M-Business Client.
2. Run the application.

After an initial message box, the form is displayed.

3. Insert two rows into the table:
 - a. Enter a first name of Jane in the first text box and a last name of Doe in the second text box. Click Insert.

A row is added to the table with these values. The application moves to the last row of the table and displays the row. The label displays the automatically incremented value of the ID column that UltraLite assigned to the row.

- b. Enter a first name of John in the first text box and a last name of Smith in the second. Click Insert.

The next step is to add navigation buttons

Lesson 6: Add navigation to your application

This lesson describes code for scrolling forward and backward through the rows of a result set.

◆ Add navigation code to your application

1. Add the MoveNext function to *tutorial.js*, immediately after the Insert function:

```
function MoveNext()
{
    if( ! CustomerTable.moveNext() ) {
        CustomerTable.moveLast();
    }
}
```

2. Add the MovePrev function to *tutorial.js*, immediately after the MoveNext function:

```
function MovePrev()
{
    if( ! CustomerTable.movePrevious() ) {
        CustomerTable.moveFirst();
    }
}
```

3. Add the following procedures to *main.html*:

```
function ClickNext()
{
    MoveNext();
    DisplayCurrentRow();
}

function ClickPrev()
{
    MovePrev();
    DisplayCurrentRow();
}
```

4. Synchronize your application and test the navigation buttons.

When the form is first displayed, the controls are empty as the current position is before the first row. After the form is displayed, click Next and Previous to move through the rows of the table.

Lesson 7: Add updates and deletes to your application

This lesson describes code for updating and deleting rows.

◆ Add update and delete functions to your application

1. Add the Update function to *tutorial.js*:

```
function Update()
{
  var id;
  try {
    CustomerTable.updateBegin();

    id = CustomerTable.schema.getColumnID("fname");
    CustomerTable.setString(id, Cust_FName);
    id = CustomerTable.schema.getColumnID("lname");
    CustomerTable.setString(id, Cust_LName);
    id = CustomerTable.schema.getColumnID("city");
    if( Cust_City.length > 0 ) {
      CustomerTable.setString(id, Cust_City);
    } else {
      CustomerTable.setNull(id);
    }
    id = CustomerTable.schema.getColumnID("phone");
    if( Cust_Phone.length > 0 ) {
      CustomerTable.setString(id, Cust_Phone);
    } else {
      CustomerTable.setNull(id);
    }
    CustomerTable.update();
  } catch( ex ) {
    alert( "Update error: " + ex.getMessage() );
  }
}
```

2. Add the Delete function to *tutorial.js*:

```
function Delete()
{
  if( CustomerTable.getRowCount() == 0 ) {
    return;
  }
  CustomerTable.deleteRow();
  CustomerTable.moveRelative(0);
}
```

3. Add the following functions to *main.html*:

```
function ClickUpdate()
{
  FetchForm();
  Update();
  DisplayCurrentRow();
}

function ClickDelete()
{
  Delete();
  DisplayCurrentRow();
}
```


4. Synchronize your M-Business Client and run the application.

Lesson 8: Add synchronization to your application

The following procedure implements synchronization.

◆ Add a synchronization function to your application

1. Add the Synchronize function to *tutorial.js*:

The synchronization parameters are stored in the SyncParms object. For example, the SyncParms.userName property specifies the user name for which MobiLink searches. The SyncParms.sendColumnNames property specifies that the column names are sent to MobiLink so it can generate upload and download scripts.

This function uses a TCP/IP synchronization stream and the default network communication options (stream parameters). These default options assume that you are synchronizing from either a Windows CE client connected to the computer running the MobiLink server via ActiveSync, or from a Windows XP/200x client running on the same computer as MobiLink. If this is not the case, change the synchronization stream type and set the network communication options to the appropriate values.

☞ For more information, see [“Method setStream” on page 131](#) and [“Method setStreamParms” on page 131](#)

```
function Synchronize()
{
    var SyncParms = Connection.syncParms;

    SyncParms.setUserName( "user-name" );
    SyncParms.setStream( SyncParms.STREAM_TYPE_TCPIP );
    SyncParms.setVersion( "ul_default" );
    SyncParms.setSendColumnNames( true );

    try {
        Connection.synchronize();
    } catch( ex ) {
        alert( "Sync error: " + ex.getMessage() );
    }
}
```

2. Add the following function to *main.html*:

```
function ClickSync()
{
    window.showBusy = true;
    Synchronize();
    window.showBusy = false;
    DisplayCurrentRow();
}
```

3. Synchronize your M-Business Client.

This synchronization downloads the latest version of the application. It does not synchronize your database.

The final step in this tutorial is to synchronize your UltraLite database. The SQL Anywhere sample database has a Customer table with columns matching those in the **Customer** table in your UltraLite database. The following procedure synchronizes your database with the SQL Anywhere sample database.

◆ **To synchronize your application**

1. From a command prompt, start the MobiLink server by running the following command.

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

The -zu+ option provides automatic addition of user scripts. For more information about this option, see [“MobiLink Server Options”](#) [*MobiLink - Server Administration*].

2. In M-Business Client, click Delete repeatedly to delete all the rows in your table.
Any rows in the table would be uploaded to the Customer table in the SQL Anywhere sample database.
3. Synchronize your application.
Click Synchronize. The MobiLink server window displays the synchronization progress.
4. When the synchronization is complete, click Next and Previous to move through the rows of the table.

This completes the tutorial.

CHAPTER 4

UltraLite for M-Business Anywhere API Reference

Contents

Data types in UltraLite for M-Business Anywhere	56
Class AuthStatusCode	57
Class Connection	58
Class ConnectionParms	66
Class CreationParms	68
Class DatabaseManager	70
Class DatabaseSchema	73
Class IndexSchema	78
Class PreparedStatement	81
Class PublicationSchema	91
Class ResultSet	92
Class ResultSetSchema	108
Class SQLError	112
Class SQLType	120
Class SyncParms	122
Class SyncResult	133
Class TableSchema	139
Class ULTable	150
Class UUID	175

About this chapter

This chapter describes the UltraLite for M-Business Anywhere API, a set of classes and methods that allow you to write code for applications that use UltraLite databases. Each topic contains information about a specific class, method, constant, or enum. The reference is organized by class, with associated methods following.

Data types in UltraLite for M-Business Anywhere

JavaScript has only one numeric data type and only one Date data type.

The prototypes in this API Reference include a variety of other data types in the method and property descriptions. These types are internal M-Business Anywhere data types. Distinct numeric data types such as UInt32 (unsigned 32-bit integer) are reported here to give an idea of the size and precision of data that may be supplied. Distinct data types related to date and time (Date, Time, Timestamp) are reported so that you can write code to extract the required information from the supplied data if necessary.

Class AuthStatusCode

Enumerates the status codes that may be reported during MobiLink user authentication.

This object can be obtained from DatabaseManager as follows:

```
var authStatus = dbMgr.AuthStatusCode;
```

Properties

The following constants are properties of AuthStatusCode

Constant	Value	Description
UNKNOWN	0	Authorization status is unknown, possibly because the connection has not yet performed a synchronization.
VALID	1	User ID and password were valid at time of synchronization.
VALID_BUT_EXPIRES_SOON	2	User ID and password were valid at time of synchronization but will expire soon.
EXPIRED	3	User ID or password has expired; authorization failed.
INVALID	4	Bad user ID or password; authorization failed.
IN_USE	5	User ID is already in use; authorization failed.

Method toString

Prototype

```
String toString();
```

Returns

The name of the code or **unknown** if not a recognized code.

Remarks

Generates the string name of the authorization status code constant.

Class Connection

Represents a connection to an UltraLite database.

Connections are instantiated using one of the following methods:

- ◆ DatabaseManager.openConnection
- ◆ DatabaseManager.createDatabase

You must open a connection before carrying out any other operation, and you must close the connection after you have finished all operations on the connection and before your application terminates.

You must close all tables opened on a connection before closing the connection.

When a JavaScript Error is thrown because of a failed UltraLite database operation, the SQL error code is set on the sqlCode field of the Connection object.

Properties

Prototype	Description
Boolean autoCommit	Controls whether a commit is performed after each statement (insert, update or delete). If autoCommit is false, a commit or rollback is performed only when the user invokes the commit() or rollback() method. By default, a database commit is performed after each successful statement. If the commit fails, you have the option to execute additional SQL statements and perform the commit again, or execute a rollback statement.
String openParms (read-only)	Gets the connection parameters string as a semicolon-separated list of name=value pairs. See “UltraLite Connection String Parameters Reference” [<i>UltraLite - Database Management and Reference</i>].
DatabaseSchema databaseSchema (read-only)	Gets the database schema. This property is valid only while its connection is open.
Boolean skipMBASync (read-write)	Controls whether the database should be synchronized during one-button synchronization (false) or whether it should be skipped (true). Default is false. See “One-button synchronization” on page 29.

Prototype	Description
Int32 sqlCode (read-only)	Gets the SQL Code of the last operation on this connection. The SQL Code is the standard SQL Anywhere code and is reset by any subsequent UltraLite database operation on this connection.
SyncParms syncParms (read-only)	Gets synchronization settings for this connection. See “ Synchronization parameters for UltraLite ” [MobiLink - Client Administration].
SyncResult syncResult (read-only)	Gets the results of the most recent synchronization for this connection. See “ Synchronization parameters for UltraLite ” [MobiLink - Client Administration].
INVALID_DATABASE_ID (read-only)	A constant indicating an invalid database.

Method `changeEncryptionKey`

Prototype

`changeEncryptionKey(String newKey)`

Parameters

◆ **newKey** The new encryption key for the database.

Remarks

Changes the database encryption key to the specified new key.

If the encryption key is lost, it is not possible to open the database.

Method `close`

Prototype

`close()`

Remarks

Closes this connection.

Once a connection is closed, it cannot be reopened. To reopen a connection, a new connection object must be created and opened.

It is an error to use any object (a table or schema for example) associated with a closed connection.

In JavaScript, the closed connection object is not set to NULL automatically after it is closed. It is recommended that you explicitly set the connection object to NULL after closing the connection.

Method `commit`

Prototype

`commit()`

Remarks

Commits outstanding changes to the database.

Method `countUploadRow`

Prototype

`UInt32 countUploadRow(UInt32 mask, UInt32 threshold)`

Parameters

- ◆ **mask** Set of publications to check. For more information, see `PublicationSchema` class.
- ◆ **threshold** Value that determines the maximum number of rows to count, and so limits the amount of time taken by the call. A value of 0 corresponds to the maximum limit. A value of 1 determines if any rows need to be synchronized. **threshold** must be in range `[0,0xffffffff]`.

Returns

The number of rows to be uploaded when the next synchronization takes place.

Method `getDatabaseID`

Prototype

`UInt32 getDatabaseID()`

Remarks

Gets the current Database ID value, as set by `setDatabaseID()`.

If the value has not been set, the constant `Connection.INVALID_DATABASE_ID` is returned.

Method `getGlobalAutoIncrementUsage`

Prototype

`UInt16 getGlobalAutoIncrementUsage()`

Returns

The percentage of available global increment values used so far.

Remarks

Returns the percentage of available global autoincrement values that have been used.

If the percentage approaches 100, your application should set a new value for the global database ID using the `setDatabaseID`.

Method getLastDownloadTime**Prototype**

Date `getLastDownloadTime(UInt32 mask)`

Parameters

◆ **mask** A set of publications to check.

Returns

The timestamp of the most recent download.

Remarks

The parameter mask must reference a single publication or be the special constant `PublicationSchema.SYNC_ALL_DB` for the time of the last download of the full database.

See also

◆ [“Class PublicationSchema” on page 91](#)

Method getLastIdentity**Prototype**

UInt64 `getLastIdentity()`

Returns

The most recent identity value used.

Remarks

This function is equivalent to the following SQL statement:

```
SELECT @@identity
```

The function is particularly useful in the context of global autoincrement columns. The returned value is an unsigned 64-bit integer, database data type `UNSIGNED BIGINT`. Since this statement only allows you to determine the most recently assigned default value, you should retrieve this value soon after executing the insert statement to avoid spurious results.

Occasionally, a single insert statement may include more than one column of type global autoincrement. In this case, the return value is one of the generated default values, but there is no reliable means to determine which one. For this reason, you should design your database and write your insert statements so as to avoid this situation.

Method `getNewUUID`

Prototype

UUID `getNewUUID()`

Returns

A new UUID value.

Method `getTable`

Prototype

Table `getTable(String name, String persistName)`

Parameters

- ◆ **name** Name of the table to fetch.
- ◆ **persistName** The name for cross-page JavaScript object persistence. Set to null if no persistence is required (for example, if the application has only a single HTML page).

Remarks

Creates and returns a reference to the requested table in the database.

Method `grantConnectTo`

Prototype

`grantConnectTo(String uid, String pwd)`

Parameters

- ◆ **uid** User ID to grant access to. The maximum length is 16 characters.
- ◆ **pwd** The password for the user ID.

Remarks

Grants access to an UltraLite database for a user ID with a specified password. If an existing user ID is specified, this function updates the password for the user. UltraLite supports a maximum of 4 users.

Method `isOpen`

Prototype

Boolean `isOpen()`;

Returns

true if the connection is open, false otherwise.

Remarks

Checks whether this connection is currently open.

Method `prepareStatement`

Prototype

PreparedStatement **prepareStatement**(String *sql*, String *persistName*)

Parameters

- ◆ **sql** A SQL statement that may contain one or more '?' IN parameter placeholder.
- ◆ **persistName** The name for cross-page JavaScript object persistence. Set to null if no persistence is required (for example, if the application has only a single HTML page).

Remarks

Pre-compiles and stores into a PreparedStatement object a SQL statement with or without IN parameters. This object can then be used to efficiently execute this statement multiple times.

Method `resetLastDownloadTime`

Prototype

resetLastDownloadTime(UInt32 *mask*)

Parameters

- ◆ **mask** Set of publications to reset.

Remarks

Resets the time of the most recent download.

Method `revokeConnectFrom`

Prototype

revokeConnectFrom(String *uid*)

Parameters

- ◆ **uid** User ID to be excluded from database access. The maximum length is 16 characters.

Remarks

Revokes access to an UltraLite database for a specified user ID.

Method `rollback`

Prototype

rollback()

Remarks

Rolls back outstanding changes to the database.

Method rollbackPartialDownload

Prototype

`rollbackPartialDownload()`

Remarks

Rolls back the changes from a failed synchronization.

When a communication error occurs during the download phase of synchronization, UltraLite can apply the downloaded changes, so that the synchronization can be resumed 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), RollbackPartialDownload rolls back the failed download transaction.

Method setDatabaseID

Prototype

`setDatabaseID(UInt32 value)`

Parameters

◆ **value** Database ID value. **value** must be in range **[0,0x0fffffff]**.

Remarks

Sets the database ID value to be used for global autoincrement columns.

Method saveSyncParms

Prototype

`saveSyncParms()`

Remarks

Saves the synchronization parameters for use by HotSync or for use during one-button synchronization.

Do not confuse the saveSyncParms method with the Connection.SyncParms property. The SyncParms property is used to define the synchronization parameters for this connection. The setSyncParms method just saves these parameters so that HotSync can use them.

See also

◆ [“One-button synchronization” on page 29](#)

Method startSynchronizationDelete

Prototype

`startSynchronizationDelete()`

Remarks

Marks for synchronization all subsequent deletes made by this connection. Once this function is called, all delete operations are again synchronized.

Method stopSynchronizationDelete**Prototype**

```
stopSynchronizationDelete( )
```

Remarks

Prevents delete operations from being synchronized. This is useful for deleting old information from an UltraLite database to save space, while not deleting this information on the consolidated database.

Method synchronize**Prototype**

```
synchronize( )
```

Remarks

Synchronizes the database using the current SyncParms object. A detailed result status is reported in this connection's SyncResult object. The synchronization is carried out using the synchronization properties defined in the Connection.SyncParms object for this connection.

Method synchronizeWithParm**Prototype**

```
synchronizeWithParm( SyncParms parms)
```

Parameters

◆ **parms** The SyncParms object to use for this synchronization.

Remarks

Synchronizes the database using the specified SyncParms object. This method makes it possible to share synchronization parameters among connections.

A detailed result status is reported in this connection's SyncResult object.

Class ConnectionParms

Specifies parameters for opening a connection to an UltraLite database.

Databases are created with a single authenticated user, DBA, whose initial password is sql. By default, connections are opened using the user ID DBA and password sql. To disable the default user, use `Connection.revokeConnectFrom`. To add a user or change a user's password, use `Connection.grantConnectTo`.

Currently, only one connection can be opened at any time. Only one database may be active at a given time. Attempts to open a connection to a different database while other connections are open result in an error.

Properties

The properties of the class are listed here.

Prototype	Description
String additionalParms (read-write)	Additional parameters specified as name=value pairs separated with semicolons.
String cacheSize (read-write)	The size of the cache. CacheSize values are specified in bytes. Use the suffix k or K for kilobytes and use the suffix m or M for megabytes. The default cache size is sixteen pages. Given a default page size of 4 KB, the default cache size is 64 KB. ☞ See “ CACHE_SIZE connection parameter ” [<i>UltraLite - Database Management and Reference</i>].
String connectionName (read-write)	A name for the connection. The connection name is used to share a single connection across multiple web pages. ☞ See “ CON connection parameter ” [<i>UltraLite - Database Management and Reference</i>], and “ Maintaining connections and application state across pages ” on page 11.
String creatorIdOnPalm	The UltraLite database creator ID on the Palm device.
String databaseOnCE (read-write)	The filename of the database on Windows CE. ☞ See “ CE_FILE connection parameter ” [<i>UltraLite - Database Management and Reference</i>].
String databaseOnDesktop (read-write)	The filename of the database deployed to Windows XP/200x. ☞ See “ DBF connection parameter ” [<i>UltraLite - Database Management and Reference</i>].

Prototype	Description
String databaseOnPalm (read-write)	The filename of the UltraLite database on Palm. ☞ See “PALM_FILE connection parameter” [<i>UltraLite - Database Management and Reference</i>].
String databaseOnSymbian (read-write)	The filename of the UltraLite database on Symbian. ☞ See “SYMBIAN_FILE connection parameter” [<i>UltraLite - Database Management and Reference</i>].
String encryptionKey (read-write)	A key for encrypting the database. OpenConnection must use the same key as specified during database creation. Suggestions for keys are: <ol style="list-style-type: none"> 1. Select an arbitrary, lengthy string 2. Select strings with a variety of numbers, letters and special characters, to decrease the chances of key penetration. ☞ See “DBKEY connection parameter” [<i>UltraLite - Database Management and Reference</i>].
String password (read-write)	The password for an authenticated user. Databases are initially created with one authenticated user (DBA) with password <i>sql</i> . Passwords are case-insensitive if the database is case-insensitive and case-sensitive if the database is case-sensitive. The default value is <i>sql</i> . ☞ See “PWD connection parameter” [<i>UltraLite - Database Management and Reference</i>].
String userID (read-write)	The authenticated user for the database. Databases are initially created with one authenticated user DBA. The UserID is case-insensitive. The default value is <i>DBA</i> . ☞ See “UID connection parameter” [<i>UltraLite - Database Management and Reference</i>].

Method toString

Prototype

String toString();

Returns

The name of the code or **unknown** if not a recognized code.

Remarks

Generates the string name of the authorization status code constant.

Class CreationParms

Defines parameters that may be specified when creating an UltraLite database.

Some UltraLite database options must be set at the time the database is created. The following parameters can be supplied when creating the database using the createDatabase method. For more information see [“Method createDatabase” on page 70](#).

Properties

The properties of the class are listed here. For more information see the corresponding descriptions in [“Choosing creation-time database properties” \[UltraLite - Database Management and Reference\]](#)

Prototype	Description
Boolean caseSensitive	Sets the case-sensitivity of string comparisons in the UltraLite database.
UInt32 checksumLevel	Sets the level of checksum validation in the database. Default is 0.
String dateFormat	Sets the default string format in which dates are retrieved from the database.
String dateOrder	Controls the interpretation of date ordering of months, days, and years.
UInt32 maxHashSize	Set the maximum number of bytes that are used to hash the UltraLite indexes. Default is 0.
UInt32 nearestCentury	Controls the interpretation of two-digit years in string-to-date conversions.
Boolean obfuscate	Controls whether or not to obfuscate data in the database. Obfuscation is a form of simple encryption.
UInt32 pageSize	Defines the database page size. Valid values are: 1024, 2048, 4096, 8192, 16384. Default is 4096.
UInt32 precision	Specifies the maximum number of digits in the result of any decimal arithmetic.
UInt32 scale	Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.
String timeFormat	Sets the format for times retrieved from the database.
String timestampFormat	Determines how the timestamp is formatted in UltraLite.

Prototype	Description
String timestampIncrement	Determines how the timestamp is truncated in UltraLite.
Boolean utf8Encoding	Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode.

Class DatabaseManager

Manages connections to an UltraLite database.

You must open a connection before carrying out any other operation, and you must close the connection after you have finished all operations on the connection, and before your application terminates. You must close all tables opened on a connection before closing the connection.

Properties

The properties of the class are listed here.

Property	Description
AuthStatusCode AuthStatusCode (read-only)	Gets the AuthStatusCode object associated with the most recent synchronization.
String directory (read-only)	The directory in which M-Business Anywhere is running. On Palm OS, this property is NULL.
UInt32 runtimeType (read-only)	The runtime type: either the UltraLite runtime (stand alone) library or the UltraLite database engine. The value is an enum, and is one of the following: <ul style="list-style-type: none"> ◆ DatabaseManager.UL_STANDALONE ◆ DatabaseManager.UL_ENGINE_CLIENT
Int32 sqlCode (read-only)	Gets the SQL Code value associated with the most recent operation.
SQLException SQLException (read-only)	Gets the SQLException object.
SQLType SQLType (read-only)	Gets the SQLType object.
PODSUInt32 UL_STANDALONE (read only)	A constant indicating that the runtime type is the UltraLite runtime library.
PODSUInt32 UL_ENGINE_CLIENT (read-only)	A constant indicating that the runtime type is the UltraLite database engine.

Method createDatabase

Prototype

```
Connection createDatabase(
String access_parms ,
PODSArray *coll_bytes,
```

```
String create_parms  
)
```

Parameters

- ◆ **access_parms** Parameters for connecting to the database. `access_parms` is used to specify connection parameters (including the database filename and location) and to open the connection. For more information see [“Connecting to an UltraLite Database” \[UltraLite - Database Management and Reference\]](#).
- ◆ **coll_bytes** A byte array defining a database collation to use for the database to be created. A number of source files are supplied with UltraLite as javascript source files (.js) in `install-dir\src\ulcollations\` with filenames of the form **Collation_XXXXX.js** where XXXXX represents the collation name. For example, `coll_1250LATIN2.js`.

The .js file must be included in the main html file before the database logic. The byte array variable is defined in the .js file.
- ◆ **create_parms** Parameters for creating the database. Parameter keywords are case-insensitive, and most values are case-sensitive. `create_parms` is used to specify certain parameters that may be specified only at database creation. For more information, see [“Choosing creation-time database properties” \[UltraLite - Database Management and Reference\]](#).

Returns

No return value.

Remarks

Creates a database and opens a connection to the database as specified by **access_parms**. If the database already exists, a `SQLiteDatabase_NOT_CREATED` exception is thrown.

Only one database may be active at a given time. Attempts to open a connection to a database result in an error if there are connections open to a different database.

Method dropDatabase

Prototype

```
dropDatabase(String parms)
```

Parameters

- ◆ **parms** Parameters for identifying a database.

Remarks

Deletes the specified database.

parms is a semicolon-separated list of keyword=value pairs ("**param1=value1;param2=value2**"). Parameter keywords are case-insensitive, and most values are case-sensitive.

You cannot drop a database that has open connections.

Method `getDatabaseOptions`

Prototype

Connection `openConnection`(String *parms*)

Method `openConnection`

Prototype

Connection `openConnection`(String *parms*)

Parameters

- ◆ **parms** A String holding the parameters for opening a connection as a set of keyword=value pairs. Parameter keywords are case-insensitive, and most values are case-sensitive.

Returns

An opened connection.

Remarks

Opens a connection to the database specified by **parms**. If the database does not exist, an error is thrown. You can check `Connection.sqlCode` within the error catching code to identify the cause of the error.

Only one database may be active at a given time. Attempts to open a connection to different database while other connections are open result in an error.

Method `reOpenConnection`

Prototype

Connection `reOpenConnection`(String *connectionName*)

Parameters

- ◆ **connectionName** The name of the connection to be reopened, as specified in the `ConnectionParms.connectionName` property.

Returns

An opened Connection object. The method is used to maintain connections across multiple web pages.

Class DatabaseSchema

Represents the schema of an UltraLite database. A **DatabaseSchema** object is attached to a connection and is only valid while that connection is open.

Constants

Constant	Description
SYNC_ALL_DB	Synchronize all tables in the database.
SYNC_ALL_PUBS	Synchronize all publications in the database.

The members of the class are listed here.

Method getCollationName

Prototype

String **getCollationName()**

Returns

A string identifying the character set and sort order used in this database.

Method getDatabaseProperty

Prototype

String **getDatabaseProperty**(String *name*)

Parameters

◆ **name** Name of the database property.

Returns

Returns the value of the specified database property. Recognized properties are:

- ◆ **"date_format"** The date format used for string conversions by the database.
- ◆ **"date_order"** The date order used for string conversions by the database.
- ◆ **"nearest_century"** The nearest century used for string conversions by the database.
- ◆ **"precision"** The floating-point precision used for string conversions by the database.
- ◆ **"scale"** The minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision during string conversions by the database.
- ◆ **"time_format"** The time format used for string conversions by the database.

- ◆ "timestamp_format" The timestamp format used for string conversions by the database.
- ◆ "timestamp_increment" The minimum difference between two unique timestamps, in nanoseconds (1,000,000th of a second).

Method getDateFormat

Prototype

String getDateFormat()

Returns

The date format used for string conversions.

Method getDateOrder

Prototype

String getDateOrder()

Returns

The date order used for string conversions.

Method getNearestCentury

Prototype

String getNearestCentury()

Returns

The nearest century used for string conversions.

Method getPrecision

Prototype

String getPrecision()

Returns

The floating-point precision used for string conversions.

Method getPublicationCount

Prototype

UInt16 getPublicationCount()

Returns

The number of publications in the database.

Remarks

Publication IDs range from 1 to **getPublicationCount()**, inclusively. Publication IDs are not publication masks.

Note: Publication IDs, masks, and count may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached IDs, masks, and counts after a schema upgrade.

Method getPublicationName**Prototype**

String **getPublicationName**(UInt16 *pubID*)

Parameters

◆ **pubID** ID of the publication. **pubID** must be in range [1,**getPublicationCount**()].

Returns

The name of the publication identified by the specified publication ID. Publication IDs are not publication masks.

Remarks

Note: Publication IDs, masks, and count may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached IDs, masks, and counts after a schema upgrade.

Method getPublicationSchema**Prototype**

PublicationSchema **getPublicationSchema**(String *name*)

Parameters

◆ **name** Name of the publication.

Returns

The publication schema corresponding to the named publication.

Method getSignature**Prototype**

String **getSignature**()

Returns

The signature of this database.

Method `getTableCount`

Prototype

UInt16 `getTableCount()`

Returns

The number of tables, or 0 if the connection is not open.

Remarks

Returns the number of tables in the database. Table IDs range from 1 to `getTableCount()`, inclusively.

Method `getTableCountInPublications`

Prototype

UInt16 `getTableCountInPublications(UInt32 mask)`

Parameters

◆ **mask** Set of publications to check.

Returns

The number of tables included in the specified publication mask. The count does not include tables whose names end in `_nosync`.

Method `getTableName`

Prototype

String `getTableName(UInt16 tableID)`

Parameters

◆ **tableID** ID of the table. **tableID** must be in range `[1,getTableCount()]`.

Returns

The name of the table identified by the specified table ID.

Remarks

Note: Table IDs may change during a schema upgrade. To correctly identify a table, access it by name or refresh the cached IDs after a schema upgrade.

Method `getTimeFormat`

Prototype

String `getTimeFormat()`

Returns

The time format used for string conversions.

Method `getTimestampFormat`**Prototype**

String `getTimestampFormat()`

Remarks

The timestamp format used for string conversions.

Method `isCaseSensitive`**Prototype**

Boolean `isCaseSensitive()`

Returns

True if the database is case sensitive, false otherwise.

Method `isOpen`**Prototype**

Boolean `isOpen()`

Returns

True if the database schema is open, false otherwise.

Remarks

Checks whether this database schema is currently open.

Class IndexSchema

Represents the schema of an UltraLite table index.

This object cannot be directly instantiated. Index schemas are created using the `TableSchema.getPrimaryKey`, `TableSchema.getIndex` and `TableSchema.getOptimalIndex` methods.

Method getColumnCount

Prototype

`UInt16 getColumnCount()`

Returns

The number of columns in this index. Column IDs in indexes range from 1 to `getColumnCount()`, inclusively.

Method getColumnName

Prototype

`String getColumnName(UInt16 colIDInIndex)`

Parameters

◆ **colIDInIndex** ID in this index of the column. `colIDInIndex` must be in range [1, `getColumnCount()`].

Returns

The name of the `colIDInIndex` column in this index.

Method getName

Prototype

`String getName()`

Returns

The name of this index.

Method getReferencedIndexName

Prototype

`String getReferencedIndexName()`

Returns

The name of the referenced primary index if this index is a foreign key.

Method `getReferencedTableName`

Prototype

String `getReferencedTableName()`

Returns

The name of the referenced primary table if index is a foreign key.

Method `isColumnDescending`

Prototype

Boolean `isColumnDescending(String name)`

Parameters

◆ **name** Name of the column.

Returns

True if column is used in descending order, false if column is used in ascending order.

Method `isForeignKey`

Prototype

Boolean `isForeignKey()`

Returns

True if index is the foreign key, false if index is not the foreign key.

Remarks

Columns in a foreign key may reference a non-null unique index of another table.

Method `isForeignKeyCheckOnCommit`

Prototype

Boolean `isForeignKeyCheckOnCommit()`

Returns

True if referential integrity is checked on commits, false if it is checked on inserts and updates.

Method `isForeignKeyNullable`

Prototype

Boolean `isForeignKeyNullable()`

Returns

True if this foreign key is nullable, false if this foreign key is not nullable.

Method `isPrimaryKey`

Prototype

Boolean `isPrimaryKey()`

Returns

True if index is the primary key, false if index is not the primary key.

Remarks

Columns in the primary key may not be null.

Method `isUniqueIndex`

Prototype

Boolean `isUniqueIndex()`

Returns

True if the index is unique, false otherwise.

Remarks

Columns in a unique index may be null.

Method `isUniqueKey`

Prototype

Boolean `isUniqueKey()`

Returns

True if index is unique key, false if index is not unique key.

Remarks

Columns in a unique key may not be null.

Class PreparedStatement

Represents a pre-compiled SQL statement with or without IN parameters. Created at runtime using `Connection.prepareStatement`.

This object can then be used to efficiently execute this statements multiple times.

When a prepared statement is closed, all `ResultSet` and `ResultSetSchema` objects associated with it are also closed. For resource management reasons, it is preferred that you explicitly close prepared statements when you are done with them.

Method AppendBytesParameter

Prototype

```
AppendBytesParameter(  
    UInt16 parameterID,  
    Array value,  
    UInt32 srcOffset,  
    UInt32 count  
)
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The value to append to the current new value for the parameter.
- ◆ **srcOffset** Start position in the source array.
- ◆ **count** The number of bytes to be copied.

Remarks

Appends the specified subset of the specified array of bytes to the new value for the specified `SQLType.LONGBINARY` column. The bytes at position `srcOffset` (starting from 0) through `srcOffset + count - 1` of the array **value** are appended to the value for the specified parameter. When inserting, **insertBegin** initializes the new value to the parameter's default value.

If any of the following is true, an Error with code `SQLException.SQLE_INVALID_PARAMETER` is thrown and the destination is not modified:

- ◆ The **value** argument is null.
- ◆ The **srcOffset** argument is negative.
- ◆ The **count** argument is negative.
- ◆ **srcOffset+count** is greater than **value.length**, the length of the source array.

Method AppendStringChunkParameter

Prototype

```
AppendStringChunkParameter(  
    UInt16 parameterID,  
    String value,  
)
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The value to append to the current new value for the parameter.

Remarks

Appends the String to the new value for the specified SQLType.LONGVARCHAR.

Example

The following statement appends one hundred instances of the string **XYZ** to the first parameter:

```
for ( i = 0; i < 100; i++ ){  
    stmt.AppendStringChunkParameter( 1, "XYZ" );  
}
```

Method close

Prototype

```
close()
```

Remarks

Close the prepared statement.

When a prepared statement is closed, all ResultSet and ResultSetSchema objects associated with it are also closed.

It is recommended that you set the preparedStatement object to null immediately after you close it.

Method executeQuery

Prototype

```
ResultSet executeQuery( String persistName )
```

Parameter

- ◆ **persistName** The name for cross-page JavaScript object persistence. Set to null if no persistence is required (for example, if the application has only a single HTML page).

Returns

The result set of the query, as a set of rows.

Remarks

Executes a SQL SELECT statement and returns the result set.

Method executeStatement**Prototype**

Int32 **executeStatement**()

Returns

The number of rows affected by the statement.

Remarks

Executes a statement that does not return a result set, such as a SQL INSERT, DELETE or UPDATE statement.

If Connection.autoCommit is true, the statement commits only if one or more rows is affected by the statement.

Method getPlan**Prototype**

String **getPlan**()

Returns

A string describing the access plan UltraLite will use to execute a query. This method is intended primarily for use during development.

See also

- ◆ [“Query access plans in UltraLite”](#) [*UltraLite - Database Management and Reference*].

Method getResultSetSchema**Prototype**

ResultSetSchema **getResultSetSchema**()

Returns

The schema describing the result set of this query statement.

Method hasResultSet

Prototype

Boolean **hasResultSet**()

Returns

True if a result set is generated when this statement is executed, false if no result set is generated.

Method isOpen

Prototype

Boolean **isOpen**()

Returns

True if the prepared statement is open, false otherwise.

Method setBooleanParameter

Prototype

setBooleanParameter(UInt16 *parameterID*, Boolean *value*)

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified parameter using a Boolean.

Example

The following statement sets a value for the first parameter:

```
stmt.setBooleanParameter(1, false);
```

Method setBytesParameter

Prototype

setBytesParameter(UInt16 *parameterID*, Array *value*)

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified parameter using an array of bytes. Suitable for columns of type `SQLType.BINARY` or `SQLType.LONGBINARY` only.

Example

The following statement sets a value for the first parameter:

```
var blob = new Array( 3 );
blob[ 0 ] = 78;
blob[ 1 ] = 0;
blob[ 2 ] = 68;
stmt.setBytesParameter( 1, blob );
```

Method setDateParameter

Prototype

```
setDateParameter( UInt16 parameterID, Date value )
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified `SQLType.DATE` type parameter using a date. Only the year, month, and day fields of the `Date` object are relevant.

Example

The following statement sets a value for the first parameter to 2004/09/27:

```
stmt.setDateParameter(
    1, new Date( 2004,9,27,0,0,0 )
);
```

Method setDoubleParameter

Prototype

```
setDoubleParameter( UInt16 parameterID, Double value )
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified parameter using a **double**.

Example

The following statement sets a value for the first parameter:

```
stmt.setDoubleParameter( 1, Number.MAX_VALUE );
```

Method setFloatParameter

Prototype

```
setFloatParameter( UInt16 parameterID, Float value )
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified `SQLType.REAL` parameter.

Example

The following statement sets a floating-point value for the first parameter:

```
stmt.setFloatParameter( 1,  
    ( 2 - Math.pow(2,-23)) * Math.pow(2,127)  
);
```

Method setIntParameter

Prototype

```
setUInt16Parameter( UInt16 parameterID, UInt16 value )
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified parameter using a `UInt16`.

Example

The following statement sets the value for the first parameter to **2147483647**:

```
stmt.setIntParameter( 1, 2147483647 );
```

Method setLongParameter

Prototype

```
setLongParameter( UInt16 parameterID, Int64 value )
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified parameter.

Example

The following statement sets the value for the first parameter to **9223372036854770000**:

```
stmt.setLongParameter( 1, 9223372036854770000 );
```

Method setNullParameter

Prototype

```
setNullParameter( UInt16 parameterID )
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.

Remarks

Sets the specified parameter to the SQL NULL value.

Method setShortParameter

Prototype

```
setUInt16Parameter( UInt16 parameterID, UInt16 value )
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified parameter.

Example

The following statement sets the value for the first parameter to **32767**:

```
stmt.setShortParameter( 1, 32767 );
```

Method `setStringParameter`

Prototype

```
setStringParameter( UInt16 parameterID, String value )
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified parameter.

Example

The following statement sets the value for the first parameter to **ABC**:

```
stmt.setStringParameter( 1, "ABC" );
```

Method `setTimeParameter`

Prototype

```
setTimeParameter( UInt16 parameterID, Date value )
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified `SQLType.TIME` type parameter using a date. Only the hour, minute, and second fields of the Date object are relevant.

Example

The following statement sets a value for the first parameter to 18:02:13:0000:

```
stmt.setTimeParameter(  
    1, new Date( 1966,4,1,18,2,13,0 )  
);
```

Method setTimestampParameter

Prototype

```
setTimestampParameter( UInt16 parameterID, Date value )
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified parameter using a **Timestamp**.

Example

The following statement sets a value for the first parameter to 1966/04/01 18:02:13:0000:

```
stmt.setTimestampParameter(  
    1, new Date( 1966,4,1,18,2,13,0 )  
);
```

Method setULongParameter

Prototype

```
setULongParameter( UInt16 parameterID, UInt64 value )
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter. Uses a Double to represent the value of an unsigned 64-bit integer.

Remarks

Sets the value for the specified parameter using a Double treated as an unsigned value. For more information, see class Unsigned64.

Example

The following statement sets the value for the first parameter:

```
stmt.setLongParameter( 1, 9223372036854770000 * 4096 );
```

Method setUUIDParameter

Prototype

```
setUUIDParameter(UInt16 parameterID, UUID value)
```

Parameters

- ◆ **parameterID** The ID number of the parameter. The first parameter in the result set has an ID value of one.
- ◆ **value** The new value for the parameter.

Remarks

Sets the value for the specified parameter using a **UUID**.

Class PublicationSchema

Represents the schema of an UltraLite publication.

This class cannot be directly instantiated. Publication schemas are created using the DatabaseSchema.getPublicationSchema method.

UltraLite methods requiring a publication mask actually require a set of publications to check. A set is formed by or'ing the publication masks of individual publications. For example:

```
pub1.getMask() | pub2.getMask()
```

Two special mask values are provided by DatabaseSchema object. SYNC_ALL_DB corresponds to the entire database. SYNC_ALL_PUBS corresponds to all publications.

Publication masks may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached masks after a schema upgrade.

Method getMask

Prototype

```
UInt32 getMask()
```

Returns

The publication mask of this publication.

Note: Publication IDs, masks, and count may change during a schema upgrade. To correctly identify a publication, access it by name or refresh the cached masks, and counts after a schema upgrade.

Method getName

Prototype

```
String getName()
```

Returns

The name of this publication.

Class ResultSet

Represents a result set in an UltraLite database. Created at runtime using `PreparedStatement.executeQuery`.

Properties

The properties of the class are listed here.

Property	Description
<code>ResultSetSchema schema</code> (read-only)	The schema of this result set. This property is only valid while its prepared statement is open.
<code>NULL_TIMESTAMP_VAL</code>	A constant indicating that a timestamp value is NULL.

Method appendBytes

Prototype

```
appendBytes(  
    UInt16 columnID,  
    Array value,  
    UInt32 srcOffset,  
    UInt32 count  
)
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.
- ◆ **srcOffset** The value to append to the current new value for the column.
- ◆ **count** The number of bytes to be copied.

Remarks

Appends the specified subset of the specified array of bytes to the new value for the specified `SQLType.LONGBINARY` column. The bytes at position `srcOffset` (starting from 0) through **srcOffset + count - 1** of the array **value** are appended to the value for the specified column. When inserting, `insertBegin` initializes the new value to the column's default value. The data in the row is not actually changed until you execute an **insert**, and that change is not permanent until it is committed.

If any of the following is true, an Error with code `SQLCode.SQLE_INVALID_PARAMETER` is thrown and the destination is not modified:

- ◆ The **value** argument is null.
- ◆ The **srcOffset** argument is negative.

- ◆ The **count** argument is negative.
- ◆ **srcOffset+count** is greater than **value.length**, the length of the source array.

For other errors, a **SQLException** with the appropriate error code is thrown.

Method appendStringChunk

Prototype

```
appendStringChunk(  
    UInt16 columnID,  
    String value  
)
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Appends the specified string to the new value for the specified `SQLType.LONGVARCHAR` column.

Example

The following statements append one hundred instances of the string **XYZ** to the value in the first column:

```
for ( i = 0; i < 100; i++ ){  
    t.AppendStringChunk( 1, "XYZ" );  
}
```

Method close

Prototype

```
close()
```

Remarks

Frees all resources associated with this object.

Method deleteRow

Prototype

```
deleteRow()
```

Remarks

Deletes the current row.

Each `deleteRow` must be preceded by a call to `updateBegin`.

Method `getBoolean`

Prototype

Boolean `getBoolean`(UInt16 *index*)

Parameters

◆ **index** The ID number of the column. The first column in the result set has an ID of one.

Returns

The value for the specified column as a Boolean.

Method `getBytes`

Prototype

Array `getBytes`(UInt16 *index*)

Parameters

◆ **index** The ID number of the column. The first column in the result set has an ID of one.

Returns

The value for the specified column as an array of bytes.

Remarks

Only valid for columns of type `SQLType.BINARY` or `SQLType.LONGBINARY`.

Method `getBytesSection`

Prototype

```
UInt32 getBytesSection(  
    UInt16 index,  
    UInt32 srcOffset,  
    Array dst,  
    UInt32 dstOffset,  
    UInt32 count  
)
```

Parameters

index The 1-based ordinal of the column containing the binary data.

offset The offset into the underlying array of bytes. The source offset must be greater than or equal to 0, otherwise a `SQLE_INVALID_PARAMETER` error will be raised. A buffer bigger than 64K is also permissible.

data An array of bytes.

data_len The length of the buffer, or array. The `data_len` must be greater than or equal to 0.

Returns

The number of bytes read.

Remarks

Copies a subset of the contents of the specified `SQLType.LONGBINARY` column, beginning at the specified offset, to the specified offset of the destination byte array.

The bytes at position `srcOffset` (starting from 0) through `srcOffset+count-1` of the source array are copied into positions `dstOffset` through `dstOffset+count-1`, respectively, of the destination array. If the end of the value is encountered before `count` bytes are copied, the remainder of the destination array is left unchanged.

If any of the following is true, an error is thrown, `SQLException` code `SQLCode.SQLE_INVALID_PARAMETER` is set, and the destination is not modified:

- ◆ The `dst` argument is null.
- ◆ The `srcOffset` argument is negative.
- ◆ The `dstOffset` argument is negative.
- ◆ The `count` argument is negative.
- ◆ `dstOffset + count` is greater than the length of the destination array, `dst.length`.

Errors set

SQL_CONVERSION_ERROR The error occurs if the column data type is not `BINARY` or `LONG BINARY`.

SQL_INVALID_PARAMETER The error occurs if the column data type is `BINARY` and the offset is not 0 or 1, or, the data length is less than 0.

The error also occurs if the column data type is `LONG BINARY` and the offset is less than 1.

Method getDate

Prototype

Date **getDate**(`UInt16 index`)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as a Date.

Method getDouble

Prototype

Double **getDouble**(`UInt16 index`)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as a Double.

Method getFloat

Prototype

Float **getFloat**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value for the specified column.

Method getInt

Prototype

UInt32 **getInt**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value for the specified column.

Method getLong

Prototype

Int64 **getLong**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value for the specified column.

Method getRowCount

Prototype

UInt32 **getRowCount**()

Returns

The number of rows in the result set.

Method getShort

Prototype

```
Int16 getShort( UInt16 index )
```

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as an Int16.

Method getString

Prototype

```
String getString( UInt16 index )
```

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as a String.

Method getStringChunk

Prototype

```
String getStringChunk(  
    UInt16 index,  
    UInt32 srcOffset,  
    UInt32 count  
)
```

Parameters

- ◆ **index** The 1-based ordinal in the result set to get
- ◆ **srcOffset** The 0-based start position in the string value.
- ◆ **count** The number of characters to be copied.

Returns

The string, with specified characters copied.

Remarks

Copies a subset of the value for the specified `SQLType.LONGVARCHAR` column, starting at the specified offset, to the String object.

Method getTime

Prototype

Date **getTime**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as a Date.

Method getTimestamp

Prototype

Date **getTimestamp**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as a Date.

Method getULong

Prototype

UInt64 **getULong**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as an unsigned 64-bit integer.

Method getUUID

Prototype

UUID **getUUID**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value of the column as a UUID. The column must be of type `SQLType.BINARY` with length 16.

Method isBOF

Prototype

Boolean **isBOF**()

Returns

true if the current row position is before first row.

false otherwise.

Method isEOF

Prototype

Boolean **isEOF**()

Returns

true if the current row position is after the last row.

false otherwise.

Method isNull

Prototype

Boolean **isNull**(Uint16 *index*)

Parameters

index The column index value.

Returns

true if the value is null.

false otherwise.

Method isOpen

Prototype

Boolean **isOpen**()

Returns

true if the ResultSet is open, **false** otherwise.

Method `moveAfterLast`

Prototype

`moveAfterLast()`

Remarks

Moves to a position after the last row of the `ULResultSet`.

Method `moveBeforeFirst`

Prototype

`moveBeforeFirst()`

Remarks

Moves to a position before the first row.

Method `moveFirst`

Prototype

Boolean `moveFirst()`

Remarks

Moves to the first row.

Returns

True if successful.

False if unsuccessful. The method fails, for example, if there are no rows.

Method `moveLast`

Prototype

Boolean `moveLast()`

Remarks

Moves to the last row.

Returns

True if successful.

False if unsuccessful. The method fails, for example, if there are no rows.

Method `moveNext`

Prototype

Boolean `moveNext()`

Remarks

Moves to the next row.

Returns

True if successful.

False if unsuccessful. The method fails, for example, if there are no rows.

Method `movePrevious`

Prototype

Boolean `movePrevious()`

Remarks

Moves to the previous row.

Returns

true if successful.

false if unsuccessful. The method fails, for example, if there are no rows.

Method `moveRelative`

Prototype

Boolean `moveRelative(Int32 index)`

Remarks

Moves a certain number of rows relative to the current row. Relative to the current position of the cursor in the result set, positive index values move forward in the result set, negative index values move backward in the result set and zero does not move the cursor.

Parameters

index The number of rows to move. The value can be positive, negative, or zero.

Returns

true if successful.

false if unsuccessful. The method fails, for example, if there are no rows.

Method setBoolean

Prototype

`setBoolean(short columnID, boolean value)`

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a **boolean**. The data in the row is not actually changed until you execute an **update**, and that change is not permanent until it is committed.

Method setBytes

Prototype

`setBytes(UInt16 columnID, Array value)`

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using an array of **bytes**. Suitable for columns of type **SQLType.BINARY** or **SQLType.LONGBINARY** only. The data in the row is not actually changed until you execute an **update**, and that change is not permanent until it is committed.

Method setDate

Prototype

`setDate(UInt16 columnID, Date value)`

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a **Date**. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setDateTime

Prototype

`setDateTime(UInt16 columnID, Date value)`

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a **Date**. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setDouble**Prototype**

```
setDouble( UInt16 columnID, Double value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a **double**. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setFloat**Prototype**

```
setFloat( UInt16 columnID, Float value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a **float**. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setInt**Prototype**

```
setInt( UInt16 columnID, Int32 value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using an Integer. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setLong

Prototype

```
setLong( UInt16 columnID, Int64 value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using an Int64. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setNull

Prototype

```
setNull( UInt16 columnID )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Remarks

Sets a column to the SQL NULL. The data is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setShort

Prototype

```
setShort( UInt16 columnID, Int16 value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a UInt16. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setString

Prototype

setString(UInt16 *columnID*, String *value*)

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a String. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setTime

Prototype

setTime(UInt16 *columnID*, Date *value*)

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a Date. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setTimestamp

Prototype

setTimestamp(UInt16 *columnID*, Date *value*)

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a Date. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setULong

Prototype

setULong(UInt16 *columnID*, UInt64 *value*)

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a 64-bit integer treated as an unsigned value. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed.

Method setUUID

Prototype

```
setUUID( UInt16 columnID, UUID value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a UUID. The data in the row is not actually changed until you execute an update, and that change is not permanent until it is committed. Only valid for columns of type `SQLType.BINARY` and length 16.

See also

- ◆ [“Using UUIDs” \[MobiLink - Server Administration\]](#)

Method update

Prototype

```
update()
```

Remarks

Updates the current row with the current column values (specified using the set methods).

Each update must be preceded by a call to `updateBegin`.

Method updateBegin

Prototype

```
updateBegin()
```


Remarks

Prepares to update the current row in this result set. Column values are modified by calling the appropriate *setType* method or methods.

The data is not actually changed until you execute the update, and that change is not permanent until it is committed.

Class ResultSetSchema

Represents the schema of an UltraLite result set.

Method getColumnCount

Prototype

```
UInt16 getColumnCount();
```

Returns

The number of columns in this cursor. Column IDs range from 1 to getColumnCount inclusively.

Remarks

Column IDs and count may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

Method getColumnID

Prototype

```
UInt16 getColumnID(String name)
```

Parameters

◆ **name** The name of the column.

Returns

The column ID of the named column. Column IDs range from 1 to getColumnCount(), inclusively.

Remarks

Column IDs and count may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

Method getColumnName

Prototype

```
String getColumnName(UInt16 columnID)
```

Parameters

◆ **columnID** ID of the column. **columnID** must be in the range **[1,getColumnCount()]**.

Returns

The name of column identified by the specified column ID.

Remarks

Column IDs and count may change during a schema upgrade. To correctly identify a column, access it by name or refresh the cached IDs and counts after a schema upgrade.

Method getColumnPrecision**Prototype**

```
Int32 getColumnPrecision(String name)
```

Parameters

◆ **name** The name of the column.

Returns

The precision of the named column. The column must be of type `SQLType.NUMERIC`.

Method getColumnPrecisionByColID**Prototype**

```
Int32 getColumnPrecisionByColID( UInt16 columnID )
```

Parameters

◆ **columnID** The ID number of the column. The first column in the result set has an ID value of one.

Returns

The precision of the column. The column must be of type `SQLType.NUMERIC`.

Method getColumnScale**Prototype**

```
Int32 getColumnScale(String name)
```

Parameters

◆ **name** The name of the column.

Returns

The scale of the column. The column must be of type `SQLType.NUMERIC`.

Method getColumnScaleByColID**Prototype**

```
Int32 getColumnScaleByColID( UInt16 columnID )
```

Parameters

◆ **columnID** The ID number of the column. The first column in the result set has an ID value of one.

Returns

The scale of the column. The column must be of type `SQLType.NUMERIC`.

Method `getColumnSize`

Prototype

```
UInt32 getColumnSize(String name)
```

Parameters

◆ **name** The name of the column.

Returns

The size of the named column. The column must be of type `SQLType.NUMERIC`.

Method `getColumnSizeByColID`

Prototype

```
UInt32 getColumnSizeByColID( UInt16 columnID )
```

Parameters

◆ **columnID** The ID number of the column. The first column in the result set has an ID value of one.

Returns

The size of the column. The column must be of type `SQLType.NUMERIC`.

Method `getColumnSQLType`

Prototype

```
UInt16 getColumnSQLType(String name)
```

Parameters

◆ **name** The name of the column.

Remarks

The code for the SQL data type of the named column.

Method `getColumnSQLTypeByColID`

Prototype

```
UInt16 getColumnSQLTypeByColID( UInt16 columnID )
```

Parameters

◆ **columnID** The ID number of the column. The first column in the result set has an ID value of one.

Returns

The SQLType of the column, in a SQLType enumerated integer.

Method isOpen**Prototype**

Boolean **isOpen()**;

Returns

true if the result set is open, **false** otherwise.

Class **SQLException**

Enumerates the SQL codes that may be reported by UltraLite for M-Business Anywhere. This class provides static constants and cannot be directly instantiated.

members	Description
SQLException_AGGREGATES_NOT_ALLOWED	See “Invalid use of an aggregate function” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_ALIAS_NOT_UNIQUE	See “Alias '%1' is not unique” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_ALIAS_NOT_YET_DEFINED	See “Definition for alias '%1' must appear before its first reference” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_AMBIGUOUS_INDEX_NAME	See “Index name '%1' is ambiguous” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_BAD_ENCRYPTION_KEY	See “Incorrect or missing encryption key” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_BAD_PARAM_INDEX	See “Input parameter index out of range” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CANNOT_ACCESS_FILESYSTEM	See “Unable to access the filesystem on the device” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CANNOT_CHANGE_USER_NAME	See “Cannot change synchronization user_name when status of the last upload is unknown” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CANNOT_CONVERT	See “Invalid data conversion” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CANNOT_EXECUTE_STMT	See “Statement cannot be executed” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CANNOT_MODIFY	See “Cannot modify column '%1' in table '%2'” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CLIENT_OUT_OF_MEMORY	See “Client out of memory” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_COLUMN_AMBIGUOUS	See “Column '%1' found in more than one table -- need a correlation name” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_COLUMN_CANNOT_BE_NULL	See “Column '%1' in table '%2' cannot be NULL” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_COLUMN_IN_INDEX	See “Cannot alter a column in an index” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_COLUMN_NOT_FOUND	See “Column '%1' not found” [<i>SQL Anywhere 10 - Error Messages</i>].

members	Description
SQLException_COLUMN_NOT_INDEXED	See “Column '%1' not part of any indexes in its containing table” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_COLUMN_NOT_STREAMABLE	See “The operation failed because column '%1's type does not support streaming” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_COMMUNICATIONS_ERROR	See “Communication error” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CONNECTION_ALREADY_EXISTS	See “This connection already exists” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CONNECTION_NOT_FOUND	See “Connection not found” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CONNECTION_RESTORED	See “UltraLite connection was restored” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CONSTRAINT_NOT_FOUND	See “Constraint '%1' not found” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CONVERSION_ERROR	See “Cannot convert %1 to a %2” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_COULD_NOT_FIND_FUNCTION	See “Could not find '%1' in dynamic library '%2'” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_COULD_NOT_LOAD_LIBRARY	See “Could not load dynamic library '%1'” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CURSOR_ALREADY_OPEN	See “Cursor already open” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CURSOR_NOT_OPEN	See “Cursor not open” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CURSOR_RESTORED	See “UltraLite cursor (or result set or table) was restored” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_CURSOROP_NOT_ALLOWED	See “Illegal cursor operation attempt” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_DATABASE_ERROR	See “Internal database error %1 -- transaction rolled back” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_DATABASE_NAME_REQUIRED	See “Database name required to start server” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_DATABASE_NOT_CREATED	See “Database creation failed: %1” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_DATATYPE_NOT_ALLOWED	See “Expression has unsupported data type” [<i>SQL Anywhere 10 - Error Messages</i>].

members	Description
SQLE_DBSPACE_FULL	See “A dbspace has reached its maximum file size” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_DESCRIBE_NONSELECT	See “Can only describe a SELECT statement” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_DIV_ZERO_ERROR	See “Division by zero” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_DOWNLOAD_CONFLICT	See “Download failed because of conflicts with existing rows” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_DOWNLOAD_RESTART_FAILED	See “Unable to retry download because upload is not finished” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_DROP_DATABASE_FAILED	See “An attempt to delete database '%1' failed” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_DUPLICATE_CURSOR_NAME	See “The cursor name '%1' already exists” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_DUPLICATE_FOREIGN_KEY	See “Foreign key '%1' for table '%2' duplicates an existing foreign key” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_DUPLICATE_OPTION	See “Option '%1' specified more than once” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_DYNAMIC_MEMORY_EXHAUSTED	See “Dynamic memory exhausted” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_ENCRYPTION_INITIALIZATION_FAILED	See “Could not initialize the encryption DLL: '%1'” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_ENGINE_ALREADY_RUNNING	See “Database server already running” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_ENGINE_NOT_MULTUSER	See “Database server not running in multi-user mode” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_ERROR	See “Run time SQL error -- %1” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_ERROR_CALLING_FUNCTION	See “Could not allocate resources to call external function” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_ERROR_IN_ASSIGNMENT	See “Error in assignment” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_EXPRESSION_ERROR	See “Invalid expression near '%1'” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_FEATURE_NOT_ENABLED	See “The method you attempted to invoke was not enabled for your application” [<i>SQL Anywhere 10 - Error Messages</i>].

members	Description
SQLException_BAD_DB	See “Unable to start specified database: '%1' is not a valid database file” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_IN_USE	See “Specified database file already in use” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_NOT_DB	See “Unable to start specified database: '%1' is not a database” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_VOLUME_NOT_FOUND	See “Specified file system volume not found for database '%1'” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_WRONG_VERSION	See “Unable to start specified database: '%1' was created by a different version of the software” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_FOREIGN_KEY_NAME_NOT_FOUND	See “Foreign key name '%1' not found” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_IDENTIFIER_TOO_LONG	See “Identifier '%1' too long” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_INCORRECT_VOLUME_ID	See “Incorrect volume ID for '%1'” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_INDEX_NAME_NOT_UNIQUE	See “Index name '%1' not unique” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_INDEX_NOT_FOUND	See “Cannot find index named '%1'” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_INDEX_NOT_UNIQUE	See “Index '%1' for table '%2' would not be unique” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_INTERRUPTED	See “Statement interrupted by user” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_INVALID_CONSTRAINT_REF	See “Invalid reference to or operation on constraint '%1'” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_INVALID_DESCRIPTOR_INDEX	See “Invalid descriptor index” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_INVALID_DESCRIPTOR_NAME	See “Invalid SQL descriptor name” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_INVALID_DISTINCT_AGGREGATE	See “Grouped query contains more than one distinct aggregate function” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_INVALID_FOREIGN_KEY	See “No primary key value for foreign key '%1' in table '%2'” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_INVALID_FOREIGN_KEY_DEF	See “Column '%1' in foreign key has a different definition than primary key” [<i>SQL Anywhere 10 - Error Messages</i>].

members	Description
SQL_E_INVALID_GROUP_SELECT	See “Function or column reference to '%1' must also appear in a GROUP BY” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_INVALID_LOGON	See “Invalid user ID or password” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_INVALID_OPTION_SETTING	See “Invalid setting for option '%1’” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_INVALID_OPTION_VALUE	See “'%1' is an invalid value for '%2’” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_INVALID_ORDER	See “Invalid ORDER BY specification” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_INVALID_PARAMETER	See “Invalid parameter” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_INVALID_PARSE_PARAMETER	See “Parse error: %1” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_INVALID_PUBLICATION_MASK	See “The specified publication mask is invalid” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_INVALID_SQL_IDENTIFIER	See “Invalid SQL identifier” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_INVALID_UNION	See “Select lists in UNION, INTERSECT, or EXCEPT do not match in length” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_KEYLESS_ENCRYPTION	See “Unable to perform requested operation since this database uses keyless encryption” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_LOCKED	See “User '%1' has the row in '%2' locked” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_MEMORY_ERROR	See “Memory error -- transaction rolled back” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_NAME_NOT_UNIQUE	See “Item '%1' already exists” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_NO_COLUMN_NAME	See “Derived table '%1' has no name for column %2” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_NO_CURRENT_ROW	See “No current row of cursor” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_NO_INDICATOR	See “No indicator variable provided for NULL result” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_NO_MATCHING_SELECT_ITEM	See “The select list for the derived table '%1' has no expression to match '%2’” [<i>SQL Anywhere 10 - Error Messages</i>].

members	Description
SQLException_NO_PRIMARY_KEY	See “Table '%1' has no primary key” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_NOERROR	SQLException_NOERROR(0) - This code indicates that there was no error or warning.
SQLException_NON_UPDATEABLE_COLUMN	See “Cannot update an expression” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_NON_UPDATEABLE_CURSOR	See “FOR UPDATE has been incorrectly specified for a READ ONLY cursor” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_NOT_IMPLEMENTED	See “Feature '%1' not implemented” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_NOT_SUPPORTED_IN_ULTRALITE	See “Feature not available with UltraLite” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_NOTFOUND	See “Row not found” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_ONLY_ONE_TABLE	See “INSERT/DELETE on cursor can modify only one table” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_OVERFLOW_ERROR	See “Value %1 out of range for destination” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_PAGE_SIZE_INVALID	See “Invalid database page size” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_PARTIAL_DOWNLOAD_NOT_FOUND	See “No partial download was found” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_PERMISSION_DENIED	See “Permission denied: %1” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_PRIMARY_KEY_NOT_UNIQUE	See “Primary key for table '%1' is not unique” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_PRIMARY_KEY_TWICE	See “Table cannot have two primary keys” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_PRIMARY_KEY_VALUE_REFERENCED	See “Primary key for row in table '%1' is referenced by foreign key '%2' in table '%3’” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_PUBLICATION_NOT_FOUND	See “Publication '%1' not found” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_PUBLICATION_PREDICATE_IGNORED	See “Publication predicates were not evaluated” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLException_RESOURCE_GVERNOR_EXCEEDED	See “Resource governor for '%1' exceeded” [<i>SQL Anywhere 10 - Error Messages</i>].

members	Description
SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY	See “Row was dropped from table %1 to maintain referential integrity” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_SCHEMA_UPGRADE_NOT_ALLOWED	See “A schema upgrade is not currently allowed” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_SERVER_SYNCHRONIZATION_ERROR	See “Synchronization failed due to an error on the server: %1” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_START_STOP_DATABASE_DENIED	See “Request to start/stop database denied” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_STATEMENT_ERROR	See “SQL statement error” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_STRING_RIGHT_TRUNCATION	See “Right truncation of string data” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_SUBQUERY_SELECT_LIST	See “Subquery allowed only one select list item” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_SYNC_INFO_INVALID	See “Information for synchronization is incomplete or invalid, check %1” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_SYNC_INFO_REQUIRED	See “Information for synchronization was not provided” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_SYNC_NOT_REENTRANT	See “Synchronization process was unable to re-enter synchronization” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_SYNC_STATUS_UNKNOWN	See “The status of the last synchronization upload is unknown” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_SYNTAX_ERROR	See “Syntax error near '%1' %2” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_TABLE_ALREADY_INCLUDED	See “Table '%1' is already included” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_TABLE_IN_USE	See “Table in use” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_TABLE_NOT_FOUND	See “Table '%1' not found” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_TOO_MANY_BLOB_REFS	See “Too many references to a BLOB” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_TOO_MANY_CONNECTIONS	See “Database server connection limit exceeded” [<i>SQL Anywhere 10 - Error Messages</i>].
SQLE_TOO_MANY_PUBLICATIONS	See “Too many publications specified in publication mask” [<i>SQL Anywhere 10 - Error Messages</i>].

members	Description
SQL_E_TOO_MANY_TEMP_TABLES	See “Too many temporary tables in connection” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_TOO_MANY_USERS	See “Too many users in database” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_ULTRALITE_DATABASE_NOT_FOUND	See “The database '%1' was not found” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_ULTRALITE_OBJECT_CLOSED	See “Invalid operation on a closed object” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_ULTRALITE_WRITE_ACCESS_DENIED	See “Write access was denied” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_UNABLE_TO_CONNECT	See “Database cannot be started -- %1” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_UNABLE_TO_START_DATABASE	See “Unable to start specified database: %1” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_UNABLE_TO_START_DATABASE_VER_NEWER	See “Unable to start specified database: Server must be upgraded to start database %1” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_UNCOMMITTED_TRANSACTIONS	See “You cannot synchronize or upgrade with uncommitted transactions” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_UNKNOWN_FUNC	See “Unknown function '%1’” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_UNKNOWN_OPTION	See “'%1' is an unknown option” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_UNKNOWN_USERID	See “User ID '%1' does not exist” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_UNRECOGNIZED_OPTION	See “The option '%1' is not recognized” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_UPLOAD_FAILED_AT_SERVER	See “Synchronization server failed to commit the upload” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_VALUE_IS_NULL	See “Cannot return NULL result as requested data type” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_VARIABLE_INVALID	See “Invalid host variable” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_WRONG_NUM_OF_INSERT_COLS	See “Wrong number of values for INSERT” [<i>SQL Anywhere 10 - Error Messages</i>].
SQL_E_WRONG_PARAMETER_COUNT	See “Wrong number of parameters to function '%1’” [<i>SQL Anywhere 10 - Error Messages</i>].

Class SQLType

This enumeration lists as constants the available UltraLite SQL database types used as table column types.

Constant	UltraLite Database Type
BAD_INDEX	
S_LONG	INT
U_LONG	UNSIGNED INT
S_SHORT	SMALLINT
U_SHORT	UNSIGNED SMALLINT
S_BIG	BIGINT
U_BIG	UNSIGNED BIGINT
TINY	TINY INT
BIT	BIT
TIMESTAMP	TIMESTAMP
DATE	DATE
TIME	TIMESTAMP
DOUBLE	DOUBLE
REAL	REAL
NUMERIC	NUMERIC
BINARY	BINARY
CHAR	CHAR or VARCHAR
LONGVARCHAR	LONG VARCHAR
LONGBINARY	LONG BINARY
MAX_INDEX	

Method toString

Prototype

String **toString**(UInt16 *code*)

Parameters

- ◆ **code** The SQL column type constant.

Returns

The string name of the specified SQL column type constant or `BAD_SQL_TYPE` if not a recognized type.

Class SyncParms

Represents synchronization parameters that define how to synchronize an UltraLite database. Each connection has its own SyncParms instance.

Constants

Constant	Value	Description
STREAM_TYPE_TCPIP	0	TCP/IP stream
STREAM_TYPE_HTTP	1	HTTP stream
STREAM_TYPE_HTTPS	2	HTTPS synchronization
STREAM_TYPE_TLS	3	TLS synchronization
STREAM_TYPE_HOTSYNC	4	For HotSync synchronization

Method getAuthenticationParms

Prototype

Array `getAuthenticationParms()`

Returns

Parameters provided to a custom user authentication script or null if no parameters are specified.

Method getCheckpointStore

Prototype

Boolean `getCheckpointStore()`

Returns

True if the client performs extra checkpoints, false if the client only performs required checkpoints.

Method getDisableConcurrency

Prototype

Boolean `getDisableConcurrency()`

Returns

True if concurrent synchronization is disabled, false if concurrent synchronization is enabled.

Method `getDownloadOnly`

Prototype

Boolean `getDownloadOnly()`

Returns

True if uploads are disabled, false if uploads are enabled.

Method `getKeepPartialDownload`

Prototype

Boolean `getKeepPartialDownload()`

Returns

True if partial downloads are to be kept, false if partial downloads should be rolled back.

Method `getNewPassword`

Prototype

String `getNewPassword()`

Returns

The new password that is associated with the MobiLink user after the next synchronization.

Method `getPartialDownloadRetained`

Prototype

Boolean `getPartialDownloadRetained()`

Returns

True if a download failed because of a communications error and the partial download was retained, false if the download was not interrupted, or if the partial download was rolled back.

Method `getPassword`

Prototype

String `getPassword();`

Returns

The MobiLink password for the user specified with `setUserName`.

Method `getPingOnly`

Prototype

Boolean `getPingOnly()`

Returns

True if client only pings the server, false if client performs a synchronization.

Method `getPublicationMask`

Prototype

```
UInt32 getPublicationMask();
```

Returns

The publications to be synchronized. For more information, see `PublicationSchema` class.

Method `getResumePartialDownload`

Prototype

```
Boolean getResumePartialDownload()
```

Returns

True if the previous partial download is to be resumed, false if the previous partial download is to be rolled back.

Method `getSendColumnNames`

Prototype

```
Boolean getSendColumnNames()
```

Returns

True if client sends column names to the MobiLink server during synchronization, false if client does not send column names.

Method `getSendDownloadAck`

Prototype

```
Boolean getSendDownloadAck()
```

Returns

True if client provides a download acknowledgement to the MobiLink server, false if the client does not provide a download acknowledgement.

Method `getStream`

Prototype

```
UInt16 getStream();
```

Returns

The type of MobiLink synchronization stream to use for synchronization.

Method `getStreamParms`

Prototype

String `getStreamParms()`;

Remarks

A string containing all the network protocol options used for synchronization streams.

Method `getUploadOnly`

Prototype

Boolean `getUploadOnly()`

Remarks

True if downloads are disabled, false if downloads are enabled.

Method `getUserName`

Prototype

String `getUserName()`

Returns

The MobiLink user name.

Method `getVersion`

Prototype

String `getVersion()`

Remarks

The version string that indicates which synchronization scripts are to be used.

Method `setAuthenticationParms`

Prototype

`setAuthenticationParms(Array value)`

Parameters

- ◆ **value** An array of strings, each containing an authentication parameter (null array entries result in a synchronization error).

Remarks

Specifies parameters for a custom user authentication script (MobiLink `authenticate_parameters` connection event).

Only the first 255 strings are used and each string should be no longer than 128 characters (longer strings are truncated when sent to MobiLink).

Method `setCheckpointStore`

Prototype

`setCheckpoint16Store(Boolean value)`

Parameters

- ◆ **value** Set to true to perform extra checkpoints, or set to false to only perform the required checkpoints.

Remarks

Specifies whether the client should perform extra store checkpoints to control the growth of the database store during synchronization.

The checkpoint operation adds I/O operations for the application, and so slows synchronization. This option is most useful for large downloads with many updates. Devices with slow flash memory may not want to incur the performance penalty.

Method `setDisableConcurrency`

Prototype

`setDisableConcurrency(Boolean value);`

Parameters

- ◆ **value** Set to true to disable concurrent synchronization, or set to false to enable concurrent synchronization.

Remarks

Specifies whether to disable or enable concurrent access to UltraLite while performing a synchronization.

By default, other threads may perform UltraLite operations while a thread is synchronizing. When concurrent synchronization is disabled, other threads block on UltraLite calls until the synchronization has completed.

Method `setDownloadOnly`

Prototype

`setDownloadOnly(Boolean value)`

Parameters

- ◆ **value** Set to true to disable uploads, or set to false to enable uploads.

Remarks

Specifies whether to disable or enable uploads when synchronizing.

Method setKeepPartialDownload

Prototype

```
setKeepPartialDownload( Boolean value )
```

Parameters

- ◆ **value** Set to true to enable the retention of partial downloads when synchronizing, or set to false to discard partial downloads.

Remarks

Specifies whether to disable or enable partial downloads when synchronizing. UltraLite has the ability to restart downloads that fail because of communication errors. UltraLite processes the download as it is received. If a download is interrupted, then the partial download transaction will remain in the database and can be resumed during the next synchronization.

To indicate that UltraLite should save partial downloads, specify `Connection.syncParms.setKeepPartialDownload(true)`; otherwise the download will be rolled back if an error occurs.

If a partial download was kept, then the output field `connection.SyncResult.getPartialDownloadRetained` will be set to true when the `connection.synchronize` exits. If `getPartialDownloadRetained` is set, then you can resume a download. To do this, call `connection.synchronize` with `connection.syncParms.setResumePartialDownload(true)`. You'll likely still want `KeepPartialDownload` set to true as well in case another communications error occurs. No upload is done if a download is skipped.

The download you receive during a resumed download will be as old as when the download originally began. If you need the most up to date data, then you can do another download immediately after the special resumed download completes.

When resuming a download, many of the SyncParms fields are not relevant. For example, the `PublicationMask` field is not used. You will receive the publications that you requested on the initial download. The only fields that need to be set are `setResumePartialDownload(boolean)` and `setUserName(String)`. The fields `setKeepPartialDownload(boolean)`, `setDownloadOnly(boolean)`, and `setDisableConcurrency(boolean)` may be set if desired and will function as normal.

If you have a partial download and it is no longer needed, then you can call `Connection.rollbackPartialDownload` to roll back the failed download transaction. Also if you attempt to synchronize again and do not specify `ResumePartialDownload`, then the partial download will roll back before the next synchronization begins.

For more information, refer to [“How synchronization failure is handled” \[MobiLink - Getting Started\]](#).

Method setMBA Server

Prototype

```
setMBA Server( String host, String port, String url_suffix )
```


Parameters

- ◆ **host** The host or IP value of the M-Business Anywhere server. If host is null, UltraLite sets it to the current M-Business Anywhere host.
- ◆ **port** The port on which the M-Business Anywhere server is listening. If port is null, UltraLite sets it to the current M-Business Anywhere port value.
- ◆ **url_suffix** This corresponds to the url_suffix parameter set in the *sync.conf* file of M-Business Anywhere.

Remarks

Provides a quick way to set the synchronization parameters for the MobiLink host and port to those of the M-Business Anywhere server used by the M-Business client. Use the MobiLink redirector for M-Business Anywhere to route data to and from the MobiLink server.

If you are using one-button synchronization, you must save the synchronization parameters using `Connection.saveSyncParms`.

 For information about configuring M-Business Server to route HTTP database traffic through the M-Business Anywhere Redirector, see “[M-Business Anywhere Redirector](#)” [[MobiLink - Server Administration](#)].

Method setMBA ServerWithMoreParms

Prototype

```
setMBA ServerWithMoreParms( String host, String port, String url_suffix, String additional)
```

Parameters


- ◆ **host** The host or IP value of the M-Business Anywhere server. If host is null, UltraLite sets it to the current M-Business Anywhere host.
- ◆ **port** The port on which the M-Business Anywhere server is listening. If port is null, UltraLite sets it to the current M-Business Anywhere port value.
- ◆ **url_suffix** This corresponds to the url_suffix parameter set in the *sync.conf* file of M-Business Anywhere.
- ◆ **additional** This parameter may contain additional stream parameters that are not covered by the preceding parameters (for example, proxy host, proxy port or security-related parameters). If you need to specify host, port or url_suffix information, you may use the `setMBA Server` method described in the previous section.

Remarks

Provides a quick way to set the synchronization parameters for the MobiLink host and port to those of the M-Business Anywhere server used by the M-Business client. Use the MobiLink redirector for M-Business Anywhere to route data to and from the MobiLink server.

This method expands on the capabilities provided by `setMBA Server` by permitting the user to specify other parameters in an additional parameter.

If you are using one-button synchronization, you must save the synchronization parameters using `Connection.saveSyncParms`.

 For information about configuring M-Business Server to route HTTP database traffic through the M-Business Anywhere Redirector, see “[M-Business Anywhere Redirector](#)” [[MobiLink - Server Administration](#)].

Method setNewPassword**Prototype**

```
setNewPassword( String value )
```

Parameters

◆ **value** New password for MobiLink user.

Remarks

Sets a new MobiLink password for the user specified with `setUserName`. The new password takes effect after the next synchronization.

Method setPassword**Prototype**

```
setPassword( String value )
```

Parameters

◆ **value** The password for MobiLink user.

Remarks

Sets the MobiLink password for the user specified with `setUserName`. This user name and password are separate from any database user ID and password, and serves to identify and authenticate the application to the MobiLink server.

Method setPingOnly**Prototype**

```
setPingOnly( Boolean value );
```

Parameters

- ◆ **value** Set to true to only ping the MobiLink server, or to false to perform a synchronization.

Remarks

Specifies whether the client should only ping the MobiLink server instead of performing a real synchronization.

Method `setPublicationMask`

Prototype

`setPublicationMask(UInt16 mask)`

Parameters

- ◆ **mask** Set of publications to synchronize.

Remarks

Specifies the publications to be synchronized. For more information, see **PublicationSchema** class.

Method `setSendColumnNames`

Prototype

`setSendColumnNames(Boolean value)`

Parameters

- ◆ **value** Set to true to send column names, or set to false not to send column names.

Remarks

Specifies whether the client should send column names to the MobiLink server during synchronization.

This parameter is used for direct row handling.

Method `setSendDownloadAck`

Prototype

`setSendDownloadAck(Boolean value)`

Parameters

- ◆ **value** Set to true to send a download acknowledgement (positive or negative) or set to false to tell the server that no download acknowledgement is sent.

Remarks

Specifies whether the client should send a download acknowledgement to the MobiLink server during synchronization. Such an acknowledgement is sent after the download has been fully applied and committed at the remote (positive acknowledgement) or the download fails (negative acknowledgement).

If the client does send a download acknowledgement, the MobiLink server database worker thread must wait for the client to apply and commit the download. If the client does not send a download acknowledgement, the MobiLink server is freed up sooner for its next synchronization.

Method `setStream`

Prototype

```
setStream( UInt16 value )
```

Parameters

- ◆ **value** Type of MobiLink synchronization stream to use for synchronization. For a list of valid choices, see “Constants” on page 122.

Remarks

Sets the MobiLink synchronization stream to use for synchronization. Most synchronization streams require parameters to identify the MobiLink server address and control other behavior. These parameters are supplied with the `setStreamParms()` method.

The default stream type is `STREAM_TYPE_TCPIP`.

Method `setStreamParms`

Prototype

```
setStreamParms( String value )
```

Parameters

- ◆ **value** String containing all the network protocol options used for synchronization streams. Options are specified as a semicolon-separated list of name=value pairs ("**param1=value1;param2=value2**").

Remarks

Sets the parameters to configure the synchronization stream. For information on configuring specific stream types, refer to the **Synchronization Stream Parameters Reference** section of the **UltraLite Database User's Guide** online book.

Method `setUploadOnly`

Prototype

```
setUploadOnly( Boolean value )
```

Parameters

- ◆ **value** Set to true to disable downloads, or set to false to enable downloads.

Remarks

Specifies whether to disable or enable downloads when synchronizing.

Method setUsername

Prototype

setUserName(String *value*)

Parameters

- ◆ **value** MobiLink user name.

Remarks

Sets the user name that uniquely identifies the MobiLink client to the MobiLink server. MobiLink uses this value to determine the download content, to record the synchronization state, and to recover from interruptions during synchronization. This user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink server.

Method setVersion

Prototype

setVersion(String *value*)

Parameters

- ◆ **value** Script version string.

Remarks

Specifies which synchronization script version to use. Each synchronization script in the consolidated database is marked with a version string. For example, there may be two different `download_cursor` scripts, and each one is identified by different version strings. The version string allows an UltraLite application to choose from a set of synchronization scripts.

Class SyncResult

Represents the status of the last synchronization. Each connection has its own SyncResult instance.

This class cannot be directly instantiated.

Method getAuthStatus

Prototype

UInt16 **getAuthStatus()**

Remarks

Returns the authorization status code for the last synchronization attempt.

Method getIgnoredRows

Prototype

Boolean **getIgnoredRows()**

Parameters

◆ **return** True if any uploaded rows were ignored, false if no uploaded rows were not ignored.

Returns

True if any uploaded rows were ignored during the most recent synchronization, false if no uploaded rows were ignored.

Method getPartialDownloadRetained

Prototype

Boolean **getPartialDownloadRetained()**

Returns

True if a download was interrupted and the partial download was retained, false if the download was not interrupted or the partial download was rolled back.

Method getStreamErrorCode

Prototype

UInt16 **getStreamErrorCode()**

Parameters

◆ **return** Error code reported by the synchronization stream.

Returns

An integer representing the error reported by the stream itself. The following table gives a brief description of the error codes. For more complete descriptions, see [“MobiLink Communication Error Messages” \[SQL Anywhere 10 - Error Messages\]](#)

Value	Description
0	None
1	Parameter
2	Parameter not uint32
3	Parameter not uint32 range
4	Parameter not boolean
5	Parameter not hex
6	Memory allocation
7	Parse
8	Read
9	Write
10	End write
11	End read
12	Not implemented
13	Would block
14	Generate random
15	Init random
16	Seed random
17	Create random object
18	Shutting down
19	Dequeuing connection
20	Secure certificate root
21	Secure certificate company name
22	Secure certificate chain length
23	Secure certificate ref

Value	Description
24	Secure certificate not trusted
25	Secure duplicate context
26	Secure set io
27	Secure set io semantics
28	Secure certificate chain func
29	Secure certificate chain ref
30	Secure enable non blocking
31	Secure set cipher suites
32	Secure set chain number
33	Secure certificate file not found
34	Secure read certificate
35	Secure read private key
36	Secure set private key
37	Secure certificate expiry date
38	Secure export certificate
39	Secure add certificate
40	Secure trusted certificate file not found
41	Secure trusted certificate read
42	Secure certificate count
43	Secure create certificate
44	Secure import certificate
45	Secure set random ref
46	Secure set random func
47	Secure set protocol side
48	Secure add trusted certificate
49	Secure create private key object
50	Secure certificate expired

Value	Description
51	Secure certificate company unit
52	Secure certificate common name
53	Secure handshake
54	HTTP version
55	Secure set read func
56	Secure set write func
57	Socket host name not found
58	Socket get host by addr
59	Socket localhost name not found
60	Socket create TCP/IP
61	Socket create UDP
62	Socket bind
63	Socket cleanup
64	Socket close
65	Socket connect
66	Socket get name
67	Socket get option
68	Socket set option
69	Socket listen
70	Socket shutdown
71	Socket select
72	Socket startup
73	Socket port out of range
74	Load network library
75	ActiveSync no port
89	HTTP expected post

Method `getStreamErrorContext`

Prototype

UInt16 `getStreamErrorContext()`

Remarks

The basic network operation being performed when the stream error occurred. The known contexts are as follows:

Value	Context
0	Unknown
1	Register
2	Unregister
3	Create
4	Destroy
5	Open
6	Close
7	Read
8	Write
9	WriteFlush
10	EndWrite
11	EndRead
12	Yield
13	Softshutdown

Method `getStreamErrorID`

Prototype

UInt32 `getStreamErrorID()`

Returns

The network layer reporting the error. The value is the ID of network layer.

The known IDs are as follows:

Value	Description
0	TCP/IP stream
7	HTTP stream
8	HTTPS synchronization
3	For HotSync synchronization

Method `getStreamErrorSystem`

Prototype

UInt16 `getStreamErrorSystem()`

Parameters

◆ **return** A system-specific error code.

Remarks

Returns the stream error system-specific code.

Method `getTimestamp`

Prototype

Date `getTimestamp()`

Returns

The timestamp of the most recent synchronization.

Method `getUploadOK`

Prototype

Boolean `getUploadOK()`

Remarks

True if last upload synchronization was successful, false if last upload synchronization was unsuccessful.

Class TableSchema

Represents the schema of an UltraLite table.

Method getColumnCount

Prototype

UInt16 getColumnCount()

Returns

The 1-based number of columns in this table. Column IDs range from 1 to getColumnCount().

Method getColumnDefaultValue

Prototype

String getColumnDefaultValue(String *name*)

Parameters

◆ **name** Name of the column.

Remarks

The default value of the named column or null if the default value is **null**.

Method getColumnDefaultValueByColID

Prototype

String getColumnDefaultValueByColID(UInt16 *columnID*)

Parameters

◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

The default value of the column or null if the default value is **null**.

Method getColumnID

Prototype

UInt16 getColumnID(String *name*)

Parameters

◆ **name** Name of the column.

Returns

The 1-based ID of the specified column.

Method `getColumnName`

Prototype

String `getColumnName`(UInt16 *colID*)

Parameters

◆ **colID** The 1-based column ID of the column.

Returns

The name of the specified column.

Method `getColumnPartitionSize`

Prototype

UInt64 `getColumnPartitionSize`(String *name*)

Parameters

◆ **name** Name of the column.

Returns

The column's global autoincrement partition size as an unsigned 64-bit number represented by a Double. All global autoincrement columns in a given table share the same global autoincrement partition.

Method `getColumnPartitionSizeByColID`

Prototype

UInt64 `getColumnPartitionSizeByColID`(UInt16 *columnID*)

Parameters

◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

The column's global autoincrement partition size as an unsigned 64-bit number represented by a Double. All global autoincrement columns in a given table share the same global autoincrement partition.

Method `getColumnPrecision`

Prototype

Int32 `getColumnPrecision`(String *name*)

Parameters

◆ **name** Name of the column.

Returns

The precision of the column. The column must be of type `SQLType.NUMERIC`.

Method getColumnPrecisionByColID

Prototype

Int32 getColumnPrecisionByColID(UInt16 *columnID*)

Parameters

◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

The precision of the column. The column must be of type SQLite.NUMERIC

Method getColumnScale

Prototype

Int32 getColumnScale(String *name*)

Parameters

◆ **name** Name of the column.

Returns

The scale of the column. The column must be of type SQLite.NUMERIC.

Method getColumnScaleByColID

Prototype

Int32 getColumnScaleByColID(UInt16 *columnID*)

Parameters

◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

The scale of the column. The column must be of type SQLite.NUMERIC.

Method getColumnSize

Prototype

UInt32 getColumnSize(String *name*)

Parameters

◆ **name** Name of the column.

Returns

The size of the column. The column must be of type SQLite.BINARY or SQLite.CHAR.

Method getColumnSizeByColID

Prototype

UInt32 **getColumnSizeByColID**(UInt16 *columnID*)

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

The size of the column. The column must be of type `SQLType.BINARY` or `SQLType.CHAR`.

Method getColumnSQLType

Prototype

Int16 **getColumnSQLType**(String *name*)

Parameters

- ◆ **name** Name of the column.

Returns

The `SQLType` of the column, in a `SQLType` enumerated integer.

Method getColumnSQLTypeByColID

Prototype

Int16 **getColumnSQLTypeByColID**(UInt16 *columnID*)

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

The `SQLType` of the column, in a `SQLType` enumerated integer.

Method getIndex

Prototype

IndexSchema **getIndex**(String *name*)

Parameters

- ◆ **name** Name of the index.

Returns

The index schema of the named index.

Method `getIndexCount`

Prototype

UInt16 `getIndexCount()`

Returns

The number of indexes on this table. Index IDs range from 1 to `getIndexCount()`, inclusively.

Remarks

Note: Index IDs and count may change during a schema upgrade. To correctly identify an index, access it by name or refresh the cached IDs and counts after a schema upgrade.

Method `getIndexName`

Prototype

String `getIndexName(UInt16 indexID)`

Parameters

◆ **indexID** ID of the index. **indexID** must be in the range `[1, getIndexCount()]`.

Returns

The name of the index identified by the specified index ID.

Remarks

Note: Index IDs and count may change during a schema upgrade. To correctly identify an index, access it by name or refresh the cached IDs and counts after a schema upgrade.

Method `getName`

Prototype

String `getName()`

Returns

The name of this table.

Method `getOptimalIndex`

Prototype

IndexSchema `getOptimalIndex(String name)`

Parameters

◆ **name** Name of the column.

Returns

The optimal index for searching a table using the named column. The named column is the first column in the index but the index may have more than one column.

Method `getPrimaryKey`

Prototype

IndexSchema `getPrimaryKey()`

Returns

The index schema of the primary key for this table.

Method `getUploadUnchangedRows`

Prototype

Boolean `getUploadUnchangedRows()`

Returns

True if the table is marked to upload all rows, false if the table is not marked to upload all rows.

Remarks

Tables for which this method returns true always upload unchanged rows, as well as changed rows, when the table is synchronized. These tables are sometimes referred to as "all sync" tables.

Method `isColumnAutoIncrement`

Prototype

Boolean `isColumnAutoIncrement(String name)`

Parameters

◆ **name** Name of the column.

Returns

True if the column is autoincrementing, false otherwise.

Method `isColumnAutoIncrementByColID`

Prototype

Boolean `isColumnAutoIncrementByColID(UInt16 columnID)`

Parameters

◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

True if the column is autoincrementing, false otherwise.

Method isColumnCurrentDate**Prototype**

Boolean **isColumnCurrentDate**(String *name*)

Parameters

◆ **name** Name of the column.

Returns

True if the column defaults to the current date, false otherwise.

Remarks

The column must be of type SQLType.DATE.

Method isColumnCurrentDateByColID**Prototype**

Boolean **isColumnCurrentDateByColID**(UInt16 *columnID*)

Parameters

◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

True if the column defaults to the current date, false otherwise.

Remarks

The column must be of type SQLType.DATE.

Method isColumnCurrentTime**Prototype**

Boolean **isColumnCurrentTime**(String *name*)

Parameters

◆ **name** Name of the column.

Returns

True if the column defaults to the current time, false otherwise.

Remarks

The column must be of type `SQLType.TIME`.

Method `isColumnCurrentTimeByColID`

Prototype

Boolean `isColumnCurrentTimeByColID(UInt16 columnID)`

Parameters

◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

True if the column defaults to the current time, false otherwise.

Remarks

The column must be of type `SQLType.TIME`.

Method `isColumnCurrentTimestamp`

Prototype

Boolean `isColumnCurrentTimestamp(String name)`

Parameters

◆ **name** Name of the column.

Returns

True if the column defaults to the current timestamp, false otherwise.

Remarks

The column must be of type `SQLType.TIMESTAMP`.

Method `isColumnCurrentTimestampByColID`

Prototype

Boolean `isColumnCurrentTimestampByColID(UInt16 columnID)`

Parameters

◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

True if the column defaults to the current timestamp, false otherwise.

Remarks

The column must be of type `SQLType.TIME`.

Method `isColumnGlobalAutoIncrement`**Prototype**

Boolean `isColumnGlobalAutoIncrement(String name)`

Parameters

- ◆ **name** Name of the column.
- ◆ **return** True if the column is global autoincrementing, false if not global autoincrementing.

Returns

True if the column defaults to global autoincrement, false otherwise.

Method `isColumnGlobalAutoIncrementByColID`**Prototype**

Boolean `isColumnGlobalAutoIncrementByColID(UInt16 columnID)`

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

True if the column defaults to global autoincrement, false otherwise.

Method `isColumnNewUUID`**Prototype**

Boolean `isColumnNewUUID(String name)`

Parameters

- ◆ **name** Name of the column.

Returns

True if the column defaults to a new UUID, false otherwise.

Method `isColumnNewUUIDByColID`**Prototype**

Boolean `isColumnNewUUIDByColID(UInt16 columnID)`

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

True if the column defaults to a new UUID, false otherwise.

Method `isColumnNullable`

Prototype

Boolean `isColumnNullable(String name)`

Parameters

◆ **name** Name of the column.

Returns

True if the column is nullable, false otherwise.

Method `isColumnNullableByColID`

Prototype

Boolean `isColumnNullableByColID(UInt16 columnID)`

Parameters

◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Returns

True if the column defaults to a new UUID, false otherwise.

Method `isInPublication`

Prototype

Boolean `isInPublication(String pubName)`

Parameters

◆ **pubName** Name of the publication.

Returns

True if table in publication, false if table not in publication.

Method `isNeverSynchronized`

Prototype

Boolean `isNeverSynchronized()`

Returns

True if the table is marked as never synchronized, false if the table is not marked as never synchronized.

Remarks

Tables for which this method returns true are never synchronized, even if they are included in a publication. These tables are sometimes referred to as "no sync" tables.

Class ULTable

Represents an UltraLite table.

Properties

The properties of the class are listed here.

Property	Description
TableSchema schema (read-only)	The schema of this result set. This property is only valid while its prepared statement is open.
NULL_TIMESTAMP_VAL	A constant indicating that a timestamp value is NULL.

Method AppendBytes

Prototype

```
AppendBytes(  
    UInt16 columnID,  
    Array value,  
    UInt32 srcOffset,  
    UInt32 count  
)
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.
- ◆ **srcOffset** The value to append to the current new value for the column.
- ◆ **count** The number of bytes to be copied.

Remarks

Appends the specified subset of the specified array of bytes to the new value for the specified `SQLType.LONGBINARY` column. The bytes at position `srcOffset` (starting from 0) through **srcOffset + count - 1** of the array **value** are appended to the value for the specified column. When inserting, `insertBegin` initializes the new value to the column's default value. The data in the row is not actually changed until you execute an **insert**, and that change is not permanent until it is committed.

If any of the following is true, an Error with code `SQLCode.SQLE_INVALID_PARAMETER` is thrown and the destination is not modified:

- ◆ The **value** argument is null.
- ◆ The **srcOffset** argument is negative.

- ◆ The **count** argument is negative.
- ◆ **srcOffset+count** is greater than **value.length**, the length of the source array.

For other errors, a **SQLException** with the appropriate error code is thrown.

Method AppendStringChunk

Prototype

```
AppendStringChunk(  
    UInt16 columnID,  
    String value  
)
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Appends the specified string to the new value for the specified `SQLType.LONGVARCHAR` column.

Example

The following statements append one hundred instances of the string **XYZ** to the value in the first column:

```
for ( i = 0; i < 100; i++ ){  
    t.AppendStringChunk( 1, "XYZ" );  
}
```

Method deleteRow

Prototype

```
deleteRow()
```

Remarks

Deletes the current row.

Method deleteAllRows

Prototype

```
deleteAllRows()
```

Remarks

Deletes all rows in the table.

In some applications, it can be useful to delete all rows from a table before downloading a new set of data into the table. Rows can be deleted from the UltraLite database without being deleted from the consolidated database using the `Connection.startSynchronizationDelete` method.

Method `findBegin`

Prototype

`findBegin()`

Remarks

Prepares to perform a new find on this table. The value(s) for which to search are specified by calling the appropriate `setType` method(s) on the columns in the index with which this table was opened.

Method `findFirst`

Prototype

Boolean `findFirst()`

Returns

`true` if successful, `false` otherwise

Remarks

Move forward through the table from the beginning, looking for a row that exactly matches a value or full set of values in the current index.

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure the cursor position is after the last row (isEOF).

Each search must be preceded by a call to the `findBegin` method.

See also

- ◆ [“Method `findBegin`” on page 152](#)
- ◆ [“Method `isEOF`” on page 99](#)

Method `findFirstForColumns`

Prototype

Boolean `findFirstForColumns(
 UInt16 numColumns
)`

Parameters

- ◆ **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a **numColumns** value of 1.

Returns

true if successful, **false** otherwise

Remarks

Move forward through the table from the beginning, looking for a row that exactly matches a value or partial set of values in the current index.

To specify the value for which to search for, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure the cursor position is after the last row (isEOF).

Each search must be preceded by a call to the findBegin method.

See also

- ◆ [“Method findBegin” on page 152](#)
- ◆ [“Method isEOF” on page 99](#)

Method findLast

Prototype

Boolean **findLast**()

Returns

true if successful, **false** otherwise.

Remarks

Move backward through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure the cursor position is before the first row (isBOF).

Each search must be preceded by a call to the findBegin method.

See also

- ◆ [“Method findBegin” on page 152](#)
- ◆ [“Method isBOF” on page 99](#)

Method findLastForColumns

Prototype

Boolean **findLastForColumns**(UInt16 *numColumns*)

Parameters

- ◆ **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a **numColumns** value of **1**.

Returns

true if successful, **false** otherwise.

Remarks

Move backward through the table from the end, looking for a row that exactly matches a value or partial set of values in the current index.

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure the cursor position is before the first row (isBOF).

Each search must be preceded by a call to the findBegin method.

See also

- ◆ [“Method findBegin” on page 152](#)
- ◆ [“Method isBOF” on page 99](#)

Method findNext

Prototype

Boolean **findNext**()

Returns

true if successful, **false** otherwise.

Remarks

Continues a findFirst search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or full set of values in the current index.

The cursor is left on the next row if it exactly matches the index value. On failure the cursor position is after the last row (isEOF).

The findNext method behavior is undefined if the column values being searched for are modified during a row update.

Method findNextForColumns

Prototype

Boolean **findNextForColumns**(UInt16 *numColumns*)

Parameters

- ◆ **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a **numColumns** value of 1.

Returns

true if successful, **false** otherwise.

Remarks

Continues a findFirst search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or partial set of values in the current index.

The cursor is left on the next row if it exactly matches the index value. On failure the cursor position is after the last row (isEOF).

The findNext method behavior is undefined if the column values being searched for are modified during a row update.

Method findPrevious

Prototype

Boolean **findPrevious**()

Returns

true if successful, **false** otherwise.

Remarks

Continues a findLast search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or full set of values in the current index.

The cursor is left on the previous row if it exactly matches the index value. On failure the cursor position is before the first row (isBOF).

The findPrevious method behavior is undefined if the column values being searched for are modified during a row update.

Method findPreviousForColumns

Prototype

```
Boolean findPreviousForColumns(  
    UInt16 numColumns  
)
```

Parameters

- ◆ **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a **numColumns** value of **1**.

Returns

true if successful, **false** otherwise.

Remarks

Continues a findLast search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or partial set of values in the current index.

The cursor is left on the previous row if it exactly matches the index value. On failure the cursor position is before the first row (isBOF).

The findPrevious method behavior is undefined if the column values being searched for are modified during a row update.

Method getBoolean

Prototype

```
Boolean getBoolean( UInt16 index )
```

Parameters

- ◆ **index** The ID number of the column. The first column in the result set has an ID of one.

Returns

The value for the specified column as a Boolean.

Method getBytes

Prototype

```
Array getBytes( UInt16 index )
```

Parameters

- ◆ **index** The ID number of the column. The first column in the result set has an ID of one.

Returns

The value for the specified column as an array of bytes.

Remarks

Only valid for columns of type `SQLType.BINARY` or `SQLType.LONGBINARY`.

Method getBytesSection**Prototype**

```
UInt32 getBytesSection(  
    UInt16 index  
    UInt32 srcOffset,  
    Array dst,  
    UInt32 dstOffset,  
    UInt32 count  
)
```

Parameters

index The 1-based ordinal of the column containing the binary data.

srcOffset The start position in the column value. Zero is the beginning of the value.

dst The destination array.

dstOffset The start position in the destination array.

count The number of bytes to be copied

Returns

The number of bytes read.

Remarks

Copies a subset of the contents of the specified `SQLType.LONGBINARY` column, beginning at the specified offset, to the specified offset of the destination byte array.

The bytes at position `srcOffset` (starting from 0) through `srcOffset+count-1` of the source column are copied into positions `dstOffset` through `dstOffset+count-1`, respectively, of the destination array. If the end of the value is encountered before `count` bytes are copied, the remainder of the destination array is left unchanged.

If any of the following is true, an Error is thrown, `Connection.sqlCode` set to `SQLException.INVALID_PARAMETER` and the destination is not modified:

- ◆ The `dst` argument is null
- ◆ The `srcOffset` argument is negative
- ◆ The `dstOffset` argument is negative
- ◆ The `count` argument is negative
- ◆ `dstOffset + count` is greater than `dst.length`, the length of the destination array.

Method getDate

Prototype

Date **getDate**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as a Date.

Method getDouble

Prototype

Double **getDouble**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as a Double.

Method getFloat

Prototype

Float **getFloat**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value for the specified column.

Method getInt

Prototype

Int32 **getInt**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value for the specified column.

Method getLong

Prototype

Int64 **getLong**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value for the specified column.

Method getRowCount

Prototype

UInt32 **getRowCount**()

Returns

The number of rows in the result set.

Method getShort

Prototype

Int16 **getShort**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as an Int16.

Method getString

Prototype

String **getString**(UInt32 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as a String.

Method getStringChunk

Prototype

String **getStringChunk**(
 UInt16 *index*,

```
    UInt32 srcOffset,  
    UInt32 count  
)
```

Parameters

- ◆ **index** The 1-based ordinal in the result set to get
- ◆ **srcOffset** The 0-based start position in the string value.
- ◆ **count** The number of characters to be copied.

Returns

The string, with specified characters copied.

Remarks

Copies a subset of the value for the specified `SQLType.LONGVARCHAR` column, starting at the specified offset, to the String object.

Method `getTime`

Prototype

```
Date getTime( UInt16 index )
```

Parameters

- index** The 1-based ordinal in the result set to get.

Returns

The value as a Date.

Method `getTimestamp`

Prototype

```
Date getTimestamp( UInt16 index )
```

Parameters

- index** The 1-based ordinal in the result set to get.

Returns

The value as a Date.

Method `getULong`

Prototype

```
UInt64 getULong( UInt16 index )
```

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value as an unsigned 64-bit integer.

Method getUUID**Prototype**

UUID **getUUID**(UInt16 *index*)

Parameters

index The 1-based ordinal in the result set to get.

Returns

The value of the column as a UUID. The column must be of type `SQLType.BINARY` with length 16.

Method insert**Prototype**

insert()

Remarks

Inserts a new row with the current column values (specified using the set methods).

Each insert must be preceded by a call to `insertBegin`.

Method insertBegin**Prototype**

insertBegin()

Remarks

Prepares to insert a new row into this table by setting all current column values to their default values. Call the appropriate `setType` method(s) to specify the non-default values that are to be inserted.

The row is not actually inserted and the data in the row is not actually changed until you execute the `insert` method, and that change is not permanent until it is committed.

Method lookupBackward**Prototype**

Boolean **lookupBackward**()

Returns

true if successful, **false** otherwise.

Remarks

Move backward through the table from the end, looking for a row that matches or is less than a value or full set of values in the current index.

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that matches or is less than the index value. On failure (no rows less than the value being looked for) the cursor position is before the first row (isBOF).

Each search must be preceded by a call to the `lookupBegin` method.

Method `lookupBackwardForColumns`

Prototype

Boolean `lookupBackwardForColumns(UInt16 numColumns)`

Parameters

- ◆ **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a **numColumns** value of **1**.

Returns

true if successful, **false** otherwise.

Remarks

Move backward through the table from the beginning, looking for a row that matches or is less than a value or partial set of values in the current index.

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that matches or is less than the index value. On failure (no rows less than the value being looked for) the cursor position is before the first row (isBOF).

Each search must be preceded by a call to the `lookupBegin` method.

Method `lookupBegin`

Prototype

`lookupBegin()`

Remarks

Prepares to perform a new lookup on this table. The value(s) for which to search are specified by calling the appropriate `setType` method(s) on the columns in the index with which this table was opened.

Method lookupForward

Prototype

Boolean **lookupForward**()

Returns

true if successful, **false** otherwise.

Remarks

Move forward through the table from the beginning, looking for a row that matches or is greater than a value or full set of values in the current index.

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure (no rows greater than the value being looked for) the cursor position is after the last row (isEOF).

Each search must be preceded by a call to the lookupBegin method.

Method lookupForwardForColumns

Prototype

Boolean **lookupForwardForColumns**(UInt16 *numColumns*)

Parameters

◆ **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a **numColumns** value of 1.

Returns

true if successful, **false** otherwise.

Remarks

Move forward through the table from the beginning, looking for a row that matches or is greater than a value or partial set of values in the current index.

To specify the value for which to search, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure (no rows greater than the value being looked for) the cursor position is after the last row (isEOF).

Each search must be preceded by a call to the lookupBegin method.

Method isBOF

Prototype

Boolean **isBOF**()

Returns

true if successful, **false** otherwise.

Method isEOF

Prototype

Boolean **isEOF**()

Returns

true if successful, **false** otherwise.

Method isNull

Prototype

Boolean **isNull**(Uint16 *index*)

Parameters

index The column index value.

Returns

true if the value is null.

false otherwise.

Method isOpen

Prototype

Boolean **isOpen**()

Returns

true if the ResultSet is open, **false** otherwise.

Method moveAfterLast

Prototype

Boolean **moveAfterLast**()

Remarks

Moves to a position after the last row of the ULResultSet.

Returns

true if successful.

false if unsuccessful. The method fails, for example, if there are no rows.

Method moveBeforeFirst**Prototype**

Boolean **moveBeforeFirst()**

Remarks

Moves to a position before the first row.

Returns

true if successful.

false if unsuccessful. The method fails, for example, if there are no rows.

Method moveFirst**Prototype**

Boolean **moveFirst()**

Remarks

Moves to the first row.

Returns

True if successful.

False if unsuccessful. The method fails, for example, if there are no rows.

Method moveLast**Prototype**

Boolean **moveLast()**

Remarks

Moves to the last row.

Returns

True if successful.

False if unsuccessful. The method fails, for example, if there are no rows.

Method `moveNext`

Prototype

Boolean `moveNext()`

Remarks

Moves to the next row.

Returns

True if successful.

False if unsuccessful. The method fails, for example, if there are no rows.

Method `movePrevious`

Prototype

Boolean `movePrevious()`

Remarks

Moves to the previous row.

Returns

true if successful.

false if unsuccessful. The method fails, for example, if there are no rows.

Method `moveRelative`

Prototype

Boolean `moveRelative(Int32 index)`

Remarks

Moves a certain number of rows relative to the current row. Relative to the current position of the cursor in the result set, positive index values move forward in the result set, negative index values move backward in the result set and zero does not move the cursor.

Parameters

index The number of rows to move. The value can be positive, negative, or zero.

Returns

true if successful.

false if unsuccessful. The method fails, for example, if there are no rows.

Method open

Prototype
`open()`

Remarks

Opens this table for data access using its primary key.

Method openWithIndex

Prototype
`openWithIndex(String index)`

Parameters

- ◆ **index** The name of the index with which to open the table. If null, the primary key is used.

Remarks

Opens this table for data access using the specified index.

Method setBoolean

Prototype
`setBoolean(short columnID, boolean value)`

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a **boolean**. The data in the row is not actually changed until you execute an **insert** or **update**, and that change is not permanent until it is committed.

Example

The following statement sets the value for the first column to **false**:

```
t.setBoolean( 1, false );
```

Method setBytes

Prototype
`setBytes(UInt16 columnID, Array value)`

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using an array of **bytes**. Suitable for columns of type **SQLType.BINARY** or **SQLType.LONGBINARY** only. The data in the row is not actually changed until you execute an **insert** or **update**, and that change is not permanent until it is committed.

Example

The following statements set the value of the first column:

```
var blob = new Array( 3 );
blob[ 0 ] = 78;
blob[ 1 ] = 0';
blob[ 2 ] = 68;
t.setBytes( 1, blob );
```

Method setDate

Prototype

setDate(UInt16 *columnID*, Date *value*)

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a **Date**. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Example

The following statement sets the value of the first column to 2004/09/27:

```
t.setDate(
  1, new Date( 2002,9,27,0,0,0 )
);
```

Method setDouble

Prototype

setDouble(UInt16 *columnID*, Double *value*)

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a **double**. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Example

The following example sets the value of the first column:

```
t.setDouble( 1, Number.MAX_VALUE );
```

Method setFloat**Prototype**

```
setFloat( UInt16 columnID, Float value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a Float. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Example

The following statement sets the value of the first column:

```
t.setFloat(
    1,
    (2 - Math.pow(2,-23)) * Math.pow(2,127)
);
```

Method setInt**Prototype**

```
setInt( UInt16 columnID, Int32 value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using an Integer. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Example

The following statement sets the value of the first column to 2147483647:

```
t.setInt( 1, 2147483647 );
```

Method setLong

Prototype

```
setLong( UInt16 columnID, Int64 value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using an Int64. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Example

The following statement sets the value of the first column to 9223372036854770000:

```
t.setLong( 1, 9223372036854770000 );
```

Method setNull

Prototype

```
setNull( UInt16 columnID )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Remarks

Sets a column to the SQL NULL. The data is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Method setShort

Prototype

```
setShort( UInt16 columnID, Int16 value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a UInt16. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Example

The following statement sets the value of the first column to 32767:

```
t.setShort( 1, 32767 );
```

Method setString

Prototype

```
setString( UInt16 columnID, String value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a String. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Example

The following statement sets the value of the first column to **abc**.

```
t.setString( 1, "abc" );
```

Method setTime

Prototype

```
setTime( UInt16 columnID, Date value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a Date. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Example

The following statement sets the value for the first column to 18:02:13:0000:

```
t.setTime(  
1, new Date( 1966,4,1,18,2,13,0 )  
);
```

Method setTime

Prototype

```
setTimestamp( UInt16 columnID, Date value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a Date. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Example

The following statement sets the value of the first column to 1966/04/01 18:02:13:0000:

```
t.setTimestamp(  
1, new Date( 1966,4,1,18,2,13,0 )  
);
```

Method setToDefault

Prototype

```
setToDefault( UInt16 columnID )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.

Remarks

Sets the value for the specified column to its default value. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Method setULong

Prototype

```
setULong( UInt16 columnID, UInt64 value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a 64-bit integer treated as an unsigned value. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed.

Example

The following statement sets the value for the first column:

```
t.setULong(  
    1, 9223372036854770000 * 4096  
);
```

Method setUUID**Prototype**

```
setUUID( UInt16 columnID, UUID value )
```

Parameters

- ◆ **columnID** The ID number of the column. The first column in the table has an ID value of one.
- ◆ **value** The new value for the column.

Remarks

Sets the value for the specified column using a UUID. The data in the row is not actually changed until you execute an insert or update, and that change is not permanent until it is committed. Only valid for columns of type `SQLType.BINARY` and length 16.

Example

The following statement sets a new UUID value for the first column in the table:

```
t.setUUID( 1, conn.getNewUUID(); );
```

See also

- ◆ [“Using UUIDs” \[MobiLink - Server Administration\]](#)

Method truncate**Prototype**

```
truncate()
```

Remarks

Deletes all rows in the table while temporarily activating stop synchronization delete.

Method update

Prototype

`update()`

Remarks

Updates the current row with the current column values (specified using the set methods).

Each update must be preceded by a call to `updateBegin`.

Method updateBegin

Prototype

`updateBegin()`

Remarks

Prepares to update the current row in this table. Column values are modified by calling the appropriate `setType` method or methods.

The data in the row is not actually changed until you execute the update, and that change is not permanent until it is committed.

Modifying columns in the index used to open the table will affect any active searches in unpredictable ways. Columns in the primary key of the table can not be updated.

Class UUID

Represents a UUID. A UUID (Universally Unique Identifier) or GUID (Globally Unique Identifier) is a generated value guaranteed to be unique across all computers and databases. UUIDs are stored as `SQLType.BINARY(16)` values in UltraLite databases and can be used to uniquely identify rows. The `UUID` class stores immutable UUIDs.

A UUID is associated with the `Connection` that created it and can no longer be converted to a string after the connection is closed.

Method equals

Prototype

Boolean `equals(UUID other)`

Parameters

◆ **other** `UUID` with which to compare.

Returns

true if this `UUID` is the same as the other argument, **false** otherwise.

Method toString

Prototype

String `toString()`

Returns

A string representation of this `UUID`.

Remarks

The string is of the format `XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX`, where `X` is a hexadecimal digit or null if the `Connection` associated with the `UUID` is closed.

Index

A

- accessing schema information
 - UltraLite for M-Business Anywhere, 26
- AppendBytes method
 - UltraLite for M-Business Anywhere API, 150
- appendBytes method
 - UltraLite for M-Business Anywhere API, 92
- AppendBytesParameter method
 - UltraLite for M-Business Anywhere API, 81
- AppendStringChunk method
 - UltraLite for M-Business Anywhere API, 151
- appendStringChunk method
 - UltraLite for M-Business Anywhere API, 93
- AppendStringChunkParameter method
 - UltraLite for M-Business Anywhere API, 82
- architecture
 - UltraLite for M-Business Anywhere, 3
- AuthStatusCode class
 - UltraLite for M-Business Anywhere API, 57
- AuthStatusCode properties
 - UltraLite for M-Business Anywhere API, 57
- AutoCommit mode
 - UltraLite for M-Business Anywhere, 25
- AvantGo (see M-Business Anywhere)
- AvantGo M-Business Server (see M-Business Anywhere)
- AvGo
 - UltraLite for M-Business Anywhere creator ID, 9

B

- BLOBs
 - GetByteChunk method in UltraLite for M-Business Anywhere, 25
 - UltraLite for M-Business Anywhere, 25

C

- casting
 - data types in UltraLite for M-Business Anywhere, 22
- changeEncryptionKey method
 - UltraLite for M-Business Anywhere API, 59
- close method

- UltraLite for M-Business Anywhere API, 59, 82, 93
- columns
 - accessing schema information in UltraLite for M-Business Anywhere, 26
- Columns collection
 - UltraLite for M-Business Anywhere, 20
- Commit method
 - UltraLite for M-Business Anywhere, 25
- commit method
 - UltraLite for M-Business Anywhere API, 60
- commits
 - UltraLite for M-Business Anywhere, 25
- Connection class
 - UltraLite for M-Business Anywhere API, 58
- Connection properties
 - UltraLite for M-Business Anywhere API, 58
- ConnectionParms class
 - UltraLite for M-Business Anywhere API, 66
- ConnectionParms properties
 - UltraLite for M-Business Anywhere API, 66
- conventions
 - documentation, viii
 - file names in documentation, x
- countUploadRow method
 - UltraLite for M-Business Anywhere API, 60
- createDatabase method
 - UltraLite for M-Business Anywhere API, 70
- CreationParms class
 - UltraLite for M-Business Anywhere API, 68
- CreationParms properties
 - UltraLite for M-Business Anywhere API, 68
- creator IDs
 - UltraLite for M-Business Anywhere, 9

D

- data manipulation
 - SQL in UltraLite for M-Business Anywhere, 16
 - table API in UltraLite for M-Business Anywhere, 20
 - UltraLite for M-Business Anywhere, 16
- data types
 - accessing in UltraLite for M-Business Anywhere, 21
 - casting in UltraLite for M-Business Anywhere, 22
 - JavaScript, 56
 - UltraLite for M-Business Anywhere, 56

- database schemas
 - accessing in UltraLite for M-Business Anywhere, 26
- DatabaseManager class
 - UltraLite for M-Business Anywhere API, 70
- DatabaseManager properties
 - UltraLite for M-Business Anywhere API, 70
- DatabaseSchema class
 - UltraLite for M-Business Anywhere API, 73
 - UltraLite for M-Business Anywhere development, 26
- deleteAllRows method
 - UltraLite for M-Business Anywhere API, 151
- deleteRow method
 - UltraLite for M-Business Anywhere API, 93, 151
- deleting
 - rows in UltraLite for M-Business Anywhere, 22
- deploying
 - UltraLite for M-Business Anywhere, 32
 - UltraLite for M-Business Anywhere applications to Palm OS, 32
 - UltraLite for M-Business Anywhere to Windows CE, 32
 - UltraLite for M-Business Anywhere to Windows XP, 32
- development platforms
 - UltraLite for M-Business Anywhere, 2
- DML operations
 - UltraLite for M-Business Anywhere, 16
- documentation
 - conventions, viii
 - SQL Anywhere, vi
- dropDatabase method
 - UltraLite for M-Business Anywhere API, 71

E

- encryption
 - UltraLite for M-Business Anywhere development, 15
- equals method
 - UltraLite for M-Business Anywhere API, 175
- error handling
 - UltraLite for M-Business Anywhere, 27
- errors
 - handling in UltraLite for M-Business Anywhere, 27
- executeQuery method

- UltraLite for M-Business Anywhere API, 82
- executeStatement method
 - UltraLite for M-Business Anywhere API, 83

F

- features
 - for M-Business Anywhere, 2
- feedback
 - documentation, xiii
 - providing, xiii
- find methods
 - UltraLite for M-Business Anywhere, 22
- find mode
 - UltraLite for M-Business Anywhere, 23
- findBegin method
 - UltraLite for M-Business Anywhere API, 152
- findFirst method
 - UltraLite for M-Business Anywhere API, 152
- findFirstForColumns method
 - UltraLite for M-Business Anywhere API, 152
- findLast method
 - UltraLite for M-Business Anywhere API, 153
- findLastForColumns method
 - UltraLite for M-Business Anywhere API, 154
- findNext method
 - UltraLite for M-Business Anywhere API, 154
- findNextForColumns method
 - UltraLite for M-Business Anywhere API, 155
- findPrevious method
 - UltraLite for M-Business Anywhere API, 155
- findPreviousForColumns method
 - UltraLite for M-Business Anywhere API, 156
- firewalls
 - M-Business Anywhere synchronization, 31

G

- getAuthenticationParms method
 - UltraLite for M-Business Anywhere API, 122
- getAuthStatus method
 - UltraLite for M-Business Anywhere API, 133
- getBoolean method
 - UltraLite for M-Business Anywhere API, 94, 156
- GetByteChunk method
 - UltraLite for M-Business Anywhere, 25
- getBytes method
 - UltraLite for M-Business Anywhere API, 94, 156
- getBytesSection method

UltraLite for M-Business Anywhere API, 94, 157
 getCheckpointStore method
 UltraLite for M-Business Anywhere API, 122
 getCollationName method
 UltraLite for M-Business Anywhere API, 73
 getColumnCount method
 UltraLite for M-Business Anywhere API, 78, 108, 139
 getColumnDefaultValue method
 UltraLite for M-Business Anywhere API, 139
 getColumnDefaultValueByColID method
 UltraLite for M-Business Anywhere API, 139
 getColumnID method
 UltraLite for M-Business Anywhere API, 108, 139
 columnName method
 UltraLite for M-Business Anywhere API, 78, 108
 getColumnPartitionSize method
 UltraLite for M-Business Anywhere API, 140
 getColumnPartitionSizeByColID method
 UltraLite for M-Business Anywhere API, 140
 getColumnPrecision method
 UltraLite for M-Business Anywhere API, 109, 140
 getColumnPrecisionByColID method
 UltraLite for M-Business Anywhere API, 109, 141
 getColumnScale method
 UltraLite for M-Business Anywhere API, 109, 141
 getColumnScaleByColID method
 UltraLite for M-Business Anywhere API, 109, 141
 getColumnSize method
 UltraLite for M-Business Anywhere API, 110, 141
 getColumnSizeByColID method
 UltraLite for M-Business Anywhere API, 110, 142
 getColumnSQLType method
 UltraLite for M-Business Anywhere API, 110, 142
 getColumnSQLTypeByColID method
 UltraLite for M-Business Anywhere API, 110, 142
 getDatabaseID method
 UltraLite for M-Business Anywhere API, 60
 getDatabaseOptions method
 UltraLite for M-Business Anywhere API, 72
 getDatabaseProperty method
 UltraLite for M-Business Anywhere API, 73
 getDate method
 UltraLite for M-Business Anywhere API, 95, 158
 getDateFormat method
 UltraLite for M-Business Anywhere API, 74
 getDateOrder method
 UltraLite for M-Business Anywhere API, 74
 getDisableConcurrency method
 UltraLite for M-Business Anywhere API, 122
 getDouble method
 UltraLite for M-Business Anywhere API, 95, 158
 getDownloadOnly method
 UltraLite for M-Business Anywhere API, 123
 getFloat method
 UltraLite for M-Business Anywhere API, 96, 158
 getGlobalAutoIncrementUsage method
 UltraLite for M-Business Anywhere API, 60
 getIgnoredRows method
 UltraLite for M-Business Anywhere API, 133
 getIndex method
 UltraLite for M-Business Anywhere API, 142
 getIndexCount method
 UltraLite for M-Business Anywhere API, 143
 getIndexName method
 UltraLite for M-Business Anywhere API, 143
 getInt method
 UltraLite for M-Business Anywhere API, 96, 158
 getKeepPartialDownload method
 UltraLite for M-Business Anywhere API, 123
 getLastDownloadTime method
 UltraLite for M-Business Anywhere API, 61
 getLastIdentity method
 UltraLite for M-Business Anywhere API, 61
 getLong method
 UltraLite for M-Business Anywhere API, 96, 159
 getMask method
 UltraLite for M-Business Anywhere API, 91
 getName method
 UltraLite for M-Business Anywhere API, 78, 91, 143
 getNearestCentury method
 UltraLite for M-Business Anywhere API, 74
 getNewPassword method
 UltraLite for M-Business Anywhere API, 123
 getNewUUID method
 UltraLite for M-Business Anywhere API, 62
 getOptimalIndex method
 UltraLite for M-Business Anywhere API, 143
 getPartialDownloadRetained method
 UltraLite for M-Business Anywhere API, 123, 133
 getPassword method
 UltraLite for M-Business Anywhere API, 123
 getPingOnly method
 UltraLite for M-Business Anywhere API, 123
 getPlan method

- UltraLite for M-Business Anywhere API, 83
 - getPrecision method
 - UltraLite for M-Business Anywhere API, 74
 - getPrimaryKey method
 - UltraLite for M-Business Anywhere API, 144
 - getPublicationCount method
 - UltraLite for M-Business Anywhere API, 74
 - getPublicationMask method
 - UltraLite for M-Business Anywhere API, 124
 - getPublicationName method
 - UltraLite for M-Business Anywhere API, 75
 - getPublicationSchema method
 - UltraLite for M-Business Anywhere API, 75
 - getReferencedIndexName method
 - UltraLite for M-Business Anywhere API, 78
 - getReferencedTableName method
 - UltraLite for M-Business Anywhere API, 79
 - getResultSetSchema method
 - UltraLite for M-Business Anywhere API, 83
 - getResumePartialDownload method
 - UltraLite for M-Business Anywhere API, 124
 - getRowCount method
 - UltraLite for M-Business Anywhere API, 96, 159
 - getSendColumnNames method
 - UltraLite for M-Business Anywhere API, 124
 - getSendDownloadAck method
 - UltraLite for M-Business Anywhere API, 124
 - getShort method
 - UltraLite for M-Business Anywhere API, 97, 159
 - getSignature method
 - UltraLite for M-Business Anywhere API, 75
 - getStream method
 - UltraLite for M-Business Anywhere API, 124
 - getStreamErrorCode method
 - UltraLite for M-Business Anywhere API, 133
 - getStreamErrorContext method
 - UltraLite for M-Business Anywhere API, 137
 - getStreamErrorID method
 - UltraLite for M-Business Anywhere API, 137
 - getStreamErrorSystem method
 - UltraLite for M-Business Anywhere API, 138
 - getStreamParms method
 - UltraLite for M-Business Anywhere API, 125
 - getString method
 - UltraLite for M-Business Anywhere API, 97, 159
 - getStringChunk method
 - UltraLite for M-Business Anywhere API, 97, 159
 - getTable method
 - UltraLite for M-Business Anywhere API, 62
 - getTableCount method
 - UltraLite for M-Business Anywhere API, 76
 - getTableCountInPublications method
 - UltraLite for M-Business Anywhere API, 76
 - getTableName method
 - UltraLite for M-Business Anywhere API, 76
 - getTime method
 - UltraLite for M-Business Anywhere API, 98, 160
 - getTimeFormat method
 - UltraLite for M-Business Anywhere API, 76
 - getTimestamp method
 - UltraLite for M-Business Anywhere API, 98, 138, 160
 - getTimestampFormat method
 - UltraLite for M-Business Anywhere API, 77
 - getULong method
 - UltraLite for M-Business Anywhere API, 98, 160
 - getUploadOK method
 - UltraLite for M-Business Anywhere API, 138
 - getUploadOnly method
 - UltraLite for M-Business Anywhere API, 125
 - getUploadUnchangedRows method
 - UltraLite for M-Business Anywhere API, 144
 - getUserName method
 - UltraLite for M-Business Anywhere API, 125
 - getUUID method
 - UltraLite for M-Business Anywhere API, 98, 161
 - getVersion method
 - UltraLite for M-Business Anywhere API, 125
 - grantConnectTo method
 - UltraLite for M-Business Anywhere, 28
 - UltraLite for M-Business Anywhere API, 62
- ## H
- hasResultSet method
 - UltraLite for M-Business Anywhere API, 84
 - HotSync
 - UltraLite for M-Business Anywhere, 9
 - HotSync synchronization
 - UltraLite for M-Business Anywhere synchronization parameters, 64
- ## I
- icons
 - used in manuals, x
 - IndexSchema class

- UltraLite for M-Business Anywhere API, 78
- insert method
 - UltraLite for M-Business Anywhere API, 161
- insert mode
 - UltraLite for M-Business Anywhere, 23
- insertBegin method
 - UltraLite for M-Business Anywhere API, 161
- inserting
 - rows in UltraLite for M-Business Anywhere, 22
- install-dir
 - documentation usage, x
- isBOF method
 - UltraLite for M-Business Anywhere API, 99, 164
- isCaseSensitive method
 - UltraLite for M-Business Anywhere API, 77
- isColumnAutoIncrement method
 - UltraLite for M-Business Anywhere API, 144
- isColumnAutoIncrementByColID method
 - UltraLite for M-Business Anywhere API, 144
- isColumnCurrentDate method
 - UltraLite for M-Business Anywhere API, 145
- isColumnCurrentDateByColID method
 - UltraLite for M-Business Anywhere API, 145
- isColumnCurrentTime method
 - UltraLite for M-Business Anywhere API, 145
- isColumnCurrentTimeByColID method
 - UltraLite for M-Business Anywhere API, 146
- isColumnCurrentTimestamp method
 - UltraLite for M-Business Anywhere API, 146
- isColumnCurrentTimestampByColID method
 - UltraLite for M-Business Anywhere API, 146
- isColumnDescending method
 - UltraLite for M-Business Anywhere API, 79
- isColumnGlobalAutoIncrement method
 - UltraLite for M-Business Anywhere API, 147
- isColumnGlobalAutoincrementByColID method
 - UltraLite for M-Business Anywhere API, 147
- isColumnNewUUID method
 - UltraLite for M-Business Anywhere API, 147
- isColumnNewUUIDByColID method
 - UltraLite for M-Business Anywhere API, 147
- isColumnNullable method
 - UltraLite for M-Business Anywhere API, 148
- isColumnNullableByColID method
 - UltraLite for M-Business Anywhere API, 148
- isEOF method
 - UltraLite for M-Business Anywhere API, 99, 164
- isForeignKey method

- UltraLite for M-Business Anywhere API, 79
- isForeignKeyCheckOnCommit method
 - UltraLite for M-Business Anywhere API, 79
- isForeignKeyNullable method
 - UltraLite for M-Business Anywhere API, 79
- isInPublication method
 - UltraLite for M-Business Anywhere API, 148
- isNeverSynchronized method
 - UltraLite for M-Business Anywhere API, 148
- isNull method
 - UltraLite for M-Business Anywhere API, 99, 164
- isOpen method
 - UltraLite for M-Business Anywhere API, 62, 77, 84, 99, 111, 164
- isPrimaryKey method
 - UltraLite for M-Business Anywhere API, 80
- isUniqueIndex method
 - UltraLite for M-Business Anywhere API, 80
- isUniqueKey method
 - UltraLite for M-Business Anywhere API, 80

J

- JavaScript
 - maintaining application state, 11
- JavaScript data types
 - UltraLite for M-Business Anywhere, 56
- JavaScript programming language
 - UltraLite for M-Business Anywhere, 55

L

- lookup methods
 - UltraLite for M-Business Anywhere, 22
- lookup mode
 - UltraLite for M-Business Anywhere, 23
- lookupBackward method
 - UltraLite for M-Business Anywhere API, 161
- lookupBackwardForColumns method
 - UltraLite for M-Business Anywhere API, 162
- lookupBegin method
 - UltraLite for M-Business Anywhere API, 162
- lookupForward method
 - UltraLite for M-Business Anywhere API, 163
- lookupForwardForColumns method
 - UltraLite for M-Business Anywhere API, 163

M

- M-Business Anywhere

- UltraLite, 2
- modes
 - UltraLite for M-Business Anywhere, 23
- moveAfterLast method
 - UltraLite for M-Business Anywhere API, 100, 164
- moveBeforeFirst method
 - UltraLite for M-Business Anywhere API, 100, 165
- MoveFirst method
 - UltraLite for M-Business Anywhere, 20
 - UltraLite for M-Business Anywhere development, 17
- moveFirst method
 - UltraLite for M-Business Anywhere API, 100, 165
- moveLast method
 - UltraLite for M-Business Anywhere API, 100, 165
- MoveNext method
 - UltraLite for M-Business Anywhere, 20
 - UltraLite for M-Business Anywhere development, 17
- moveNext method
 - UltraLite for M-Business Anywhere API, 101, 166
- movePrevious method
 - UltraLite for M-Business Anywhere API, 101, 166
- moveRelative method
 - UltraLite for M-Business Anywhere API, 101, 166

N

- network protocol options
 - UltraLite for M-Business Anywhere AP, 131
- newsgroups
 - technical support, xiii

O

- obfuscation
 - UltraLite for M-Business Anywhere, 15
- object hierarchy
 - UltraLite for M-Business Anywhere, 3
- one-button synchronization
 - UltraLite for M-Business Anywhere, 29
- Open method
 - ULTable object in UltraLite for M-Business Anywhere, 17, 20
- open method
 - UltraLite for M-Business Anywhere API, 167
- OpenByIndex method
 - ULTable object in UltraLite for M-Business Anywhere, 17

- openConnection method
 - UltraLite for M-Business Anywhere API, 72

P

- passwords
 - authentication in UltraLite for M-Business Anywhere, 28
- persistent names
 - UltraLite for M-Business Anywhere, 11, 12
- platforms
 - supported in UltraLite for M-Business Anywhere, 2
- prepared statements
 - UltraLite for M-Business Anywhere, 16
- PreparedStatement class
 - UltraLite for M-Business Anywhere API, 81
 - UltraLite for M-Business Anywhere usage, 16
- prepareStatement method
 - UltraLite for M-Business Anywhere API, 63
- publications
 - accessing schema information in UltraLite for M-Business Anywhere, 26
- PublicationSchema class
 - UltraLite for M-Business Anywhere API, 91
 - UltraLite for M-Business Anywhere development, 26

R

- reOpenConnection method
 - UltraLite for M-Business Anywhere API, 72
- resetLastDownloadTime method
 - UltraLite for M-Business Anywhere API, 63
- ResultSet class
 - UltraLite for M-Business Anywhere API, 92
- ResultSet properties
 - UltraLite for M-Business Anywhere API, 92
- ResultSetSchema class
 - UltraLite for M-Business Anywhere API, 108
- revokeConnectFrom method
 - UltraLite for M-Business Anywhere, 28
 - UltraLite for M-Business Anywhere API, 63
- Rollback method
 - UltraLite for M-Business Anywhere, 25
- rollback method
 - UltraLite for M-Business Anywhere API, 63
- rollbackPartialDownload method
 - UltraLite for M-Business Anywhere API, 64

rollbacks
 UltraLite for M-Business Anywhere, 25

rows
 accessing values in UltraLite for M-Business
 Anywhere, 21

S

samples-dir
 documentation usage, x

saveSyncParms method
 UltraLite for M-Business Anywhere API, 64

schemas
 UltraLite for M-Business Anywhere, 26

scope
 variables in UltraLite for M-Business Anywhere,
 11

scrolling
 UltraLite for M-Business Anywhere, 20

security
 UltraLite for M-Business Anywhere, 11

SELECT statement
 UltraLite for M-Business Anywhere development,
 17

setAuthenticationParms method
 UltraLite for M-Business Anywhere API, 125

setBoolean method
 UltraLite for M-Business Anywhere API, 102, 167

setBooleanParameter method
 UltraLite for M-Business Anywhere API, 84

setBytes method
 UltraLite for M-Business Anywhere API, 102, 167

setBytesParameter method
 UltraLite for M-Business Anywhere API, 84

setCheckpointStore method
 UltraLite for M-Business Anywhere API, 126

setDatabaseID method
 UltraLite for M-Business Anywhere API, 64

setDate method
 UltraLite for M-Business Anywhere API, 102, 168

setDateParameter method
 UltraLite for M-Business Anywhere API, 85

setDateTime method
 UltraLite for M-Business Anywhere API, 102

setDisableConcurrency method
 UltraLite for M-Business Anywhere API, 126

setDouble method
 UltraLite for M-Business Anywhere API, 103, 168

setDoubleParameter method
 UltraLite for M-Business Anywhere API, 85

setDownloadOnly method
 UltraLite for M-Business Anywhere API, 126

setFloat method
 UltraLite for M-Business Anywhere API, 103, 169

setFloatParameter method
 UltraLite for M-Business Anywhere API, 86

setInt method
 UltraLite for M-Business Anywhere API, 103, 169

setIntParameter method
 UltraLite for M-Business Anywhere API, 86

setKeepPartialDownload
 UltraLite for M-Business Anywhere API, 127

setLong method
 UltraLite for M-Business Anywhere API, 104, 170

setLongParameter method
 UltraLite for M-Business Anywhere API, 87

setMBA Server method
 UltraLite for M-Business Anywhere API, 128

setMBA Server With More Parms method
 UltraLite for M-Business Anywhere API, 128

setNewPassword method
 UltraLite for M-Business Anywhere API, 129

setNull method
 UltraLite for M-Business Anywhere API, 104, 170

setNullParameter method
 UltraLite for M-Business Anywhere API, 87

setPassword method
 UltraLite for M-Business Anywhere API, 129

setPingOnly method
 UltraLite for M-Business Anywhere API, 129

setPublicationMask method
 UltraLite for M-Business Anywhere API, 130

setSendColumnNames method
 UltraLite for M-Business Anywhere API, 130

setSendDownloadAck method
 UltraLite for M-Business Anywhere API, 130

setShort method
 UltraLite for M-Business Anywhere API, 104, 170

setShortParameter method
 UltraLite for M-Business Anywhere API, 87

setStream method
 UltraLite for M-Business Anywhere API, 131

setStreamParms method
 UltraLite for M-Business Anywhere API, 131

setString method
 UltraLite for M-Business Anywhere API, 105, 171

- setStringParameter method
 - UltraLite for M-Business Anywhere API, 88
 - setTime method
 - UltraLite for M-Business Anywhere API, 105, 171
 - setTimeParameter method
 - UltraLite for M-Business Anywhere API, 88
 - setTimestamp method
 - UltraLite for M-Business Anywhere API, 105, 172
 - setTimestampParameter method
 - UltraLite for M-Business Anywhere API, 89
 - setDefault method
 - UltraLite for M-Business Anywhere API, 172
 - setULong method
 - UltraLite for M-Business Anywhere API, 105, 172
 - setULongParameter method
 - UltraLite for M-Business Anywhere API, 89
 - setUpOnly method
 - UltraLite for M-Business Anywhere API, 131
 - setUserName method
 - UltraLite for M-Business Anywhere API, 132
 - setUUID method
 - UltraLite for M-Business Anywhere API, 106, 173
 - setUUIDParameter method
 - UltraLite for M-Business Anywhere API, 89
 - setVersion method
 - UltraLite for M-Business Anywhere API, 132
 - SQL Anywhere
 - documentation, vi
 - SQLException class
 - UltraLite for M-Business Anywhere API, 112
 - SQLType class
 - UltraLite for M-Business Anywhere API, 120
 - startSynchronizationDelete method
 - UltraLite for M-Business Anywhere API, 64
 - stopSynchronizationDelete method
 - UltraLite for M-Business Anywhere API, 65
 - support
 - newsgroups, xiii
 - supported platforms
 - UltraLite for M-Business Anywhere, 2
 - synchronization
 - HTTP in UltraLite for M-Business Anywhere, 30, 31
 - HTTPS in UltraLite for M-Business Anywhere, 30
 - TCP/IP in UltraLite for M-Business Anywhere, 30, 31
 - UltraLite for M-Business Anywhere development, 29
 - synchronize method
 - UltraLite for M-Business Anywhere API, 65
 - synchronizeWithParm method
 - UltraLite for M-Business Anywhere API, 65
 - synchronizing UltraLite applications
 - UltraLite for M-Business Anywhere development, 29
 - SyncParms class
 - UltraLite for M-Business Anywhere API, 122
 - SyncResult class
 - UltraLite for M-Business Anywhere API, 133
- ## T
- tables
 - accessing schema information in UltraLite for M-Business Anywhere, 26
 - TableSchema class
 - UltraLite for M-Business Anywhere API, 139
 - UltraLite for M-Business Anywhere development, 26
 - target platforms
 - UltraLite for M-Business Anywhere, 2
 - technical support
 - newsgroups, xiii
 - toString method
 - UltraLite for M-Business Anywhere API, 57, 67, 120, 175
 - transaction processing
 - UltraLite for M-Business Anywhere, 25
 - transactions
 - UltraLite for M-Business Anywhere, 25
 - truncate method
 - UltraLite for M-Business Anywhere API, 173
- ## U
- ULTable class
 - UltraLite for M-Business Anywhere API, 150
 - UltraLite for M-Business Anywhere development, 17
 - ULTable properties
 - UltraLite for M-Business Anywhere API, 150
 - UltraLite databases
 - accessing schema information for M-Business Anywhere, 26
 - connecting in UltraLite for M-Business Anywhere, 10
 - UltraLite for M-Business Anywhere

- about, 1
- accessing schema information, 26
- architecture, 3
- connecting to UltraLite databases, 10
- data manipulation using SQL, 16
- data manipulation with Table API, 20
- deploying applications, 32
- deploying applications to Palm OS, 32
- deploying applications to Windows CE, 32
- deploying applications to Windows XP, 32
- encryption, 15
- error handling, 27
- features, 2
- maintaining state, 11
- object hierarchy, 3
- project architecture, 37
- quick start, 6
- supported platforms, 2
- synchronizing UltraLite applications, 29
- user authentication, 28
- UltraLite for M-Business Anywhere API
 - alphabetical listing, 55
- update method
 - UltraLite for M-Business Anywhere API, 106, 174
- update mode
 - UltraLite for M-Business Anywhere, 23
- updateBegin method
 - UltraLite for M-Business Anywhere API, 106, 174
- updating
 - rows UltraLite for M-Business Anywhere, 22
- user authentication
 - UltraLite for M-Business Anywhere, 28
- UUIDs
 - UltraLite class for M-Business Anywhere API, 175

V

- values
 - accessing in UltraLite for M-Business Anywhere, 21
- Visual Basic
 - supported versions in UltraLite for M-Business Anywhere, 2

W

- Windows CE
 - target platform in UltraLite for M-Business Anywhere, 2
