



UltraLite™ for M-Business Anywhere User's Guide

Part number: DC20093-01-0902-01

Last modified: October 2004

Copyright © 1989–2004 Sybase, Inc. Portions copyright © 2001–2004 iAnywhere Solutions, Inc. All rights reserved.

No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of iAnywhere Solutions, Inc. iAnywhere Solutions, Inc. is a subsidiary of Sybase, Inc.

Sybase, SYBASE (logo), AccelaTrade, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, AnswerBase, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, ASEP, AvantGo, AvantGo Application Alerts, AvantGo Mobile Delivery, AvantGo Mobile Document Viewer, AvantGo Mobile Inspection, AvantGo Mobile Marketing Channel, AvantGo Mobile Pharma, AvantGo Mobile Sales, AvantGo Pylon, AvantGo Pylon Application Server, AvantGo Pylon Conduit, AvantGo Pylon PIM Server, AvantGo Pylon Pro, Backup Server, BayCam, Bit-Wise, BizTracker, Certified PowerBuilder Developer, Certified SYBASE Professional, Certified SYBASE Professional Logo, ClearConnect, Client Services, Client-Library, CodeBank, Column Design, ComponentPack, Connection Manager, Convoy/DM, Copernicus, CSP, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, Dynamic Mobility Model, Dynamo, e-ADK, E-Anywhere, e-Biz Integrator, EC Gateway, ECMAP, ECRTP, eFulfillment Accelerator, Electronic Case Management, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise Portal (logo), Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, eProcurement Accelerator, eremote, Everything Works Better When Everything Works Together, EWA, E-Whatever, Financial Fusion, Financial Fusion (and design), Financial Fusion Server, Formula One, Fusion Powered e-Finance, Fusion Powered Financial Destinations, Fusion Powered STP, Gateway Manager, GeoPoint, GlobalFIX, iAnywhere, iAnywhere Solutions, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InstaHelp, Intelligent Self-Care, InternetBuilder, iremote, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, Logical Memory Manager, Mail Anywhere Studio, MainframeConnect, Maintenance Express, Manage Anywhere Studio, MAP, M-Business Channel, M-Business Network, M-Business Server, MDI Access Server, MDI Database Gateway, media.splash, Message Anywhere Server, MetaWorks, MethodSet, ML Query, MobicATS, My AvantGo, My AvantGo Media Channel, My AvantGo Mobile Marketing, MySupport, Net-Gateway, Net-Library, New Era of Networks, Next Generation Learning, Next Generation Learning Studio, O DEVICE, OASis, OASis logo, ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Biz, Open Business Interchange, Open Client, Open Client/Server, Open Client/Server Interfaces, Open ClientConnect, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, Orchestration Studio, Partnerships that Work, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, PhysicalArchitect, Pocket PowerBuilder, PocketBuilder, Power Through Knowledge, power.stop, Power++, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, Powering the New Economy, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, Powersoft Portfolio, Powersoft Professional, PowerStage, PowerStudio, PowerTips, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, QAnywhere, Rapport, Relational Beans, RepConnector, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Report Workbench, Report-Execute, Resource Manager, RW-DisplayLib, RW-Library, S.W.I.F.T. Message Format Libraries, SAFE, SAFE/PRO, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILLS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL Server SNMP SubAgent, SQL Server/CFT, SQL Server/DBM, SQL SMART, SQL Station, SQL Toolset, SQLJ, Stage III Engineering, Startup.Com, STEP, SupportNow, Sybase Central, Sybase Client/Server Interfaces, Sybase Development Framework, Sybase Financial Server, Sybase Gateways, Sybase Learning Connection, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase Synergy Program, Sybase User Workbench, Sybase Virtual Server Architecture, SybaseWare, Syber Financial, SyberAssist, SybMD, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, The Enterprise Client/Server Company, The Extensible Software Platform, The Future Is Wide Open, The Learning Connection, The Model For Client/Server Solutions, The Online Information Center, The Power of One, TotalFix, TradeForce, Transact-SQL, Translation Toolkit, Turning Imagination Into Reality, UltraLite, UltraLite.NET, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Versacore, Viewer, VisualWriter, VQL, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, WarehouseArchitect, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server, and XP Server are trademarks of Sybase, Inc. or its subsidiaries.

Certicom and SSL Plus are trademarks and Security Builder is a registered trademark of Certicom Corp. Copyright © 1997–2001 Certicom Corp. Portions are Copyright © 1997–1998, Consensus Development Corporation, a wholly owned subsidiary of Certicom Corp. All rights reserved. Contains an implementation of NR signatures, licensed under U.S. patent 5,600,725. Protected by U.S. patents 5,787,028; 4,745,568; 5,761,305. Patents pending.

All other trademarks are property of their respective owners.

Contents

About This Manual	v
SQL Anywhere Studio documentation	vi
Documentation conventions	ix
The CustDB sample database	xi
Finding out more and providing feedback	xii
1 Introduction to UltraLite for M-Business Anywhere	1
UltraLite for M-Business Anywhere features	2
UltraLite for M-Business Anywhere architecture	3
2 Understanding UltraLite for M-Business Anywhere Development	5
UltraLite for M-Business Anywhere Quick Start	6
Working with the database schema	11
Connecting to an UltraLite database	13
Maintaining connections and application state across pages	16
Encryption and obfuscation	17
Working with data using dynamic SQL	18
Working with data using the table API	23
Accessing schema information	30
Handling errors	31
Authenticating users	32
Synchronizing data	33
Deploying UltraLite for M-Business Anywhere applications	37
3 Tutorial: A Sample Application for M-Business Anywhere	39
Introduction	40
Lesson 1: Create a project architecture	41
Lesson 2: Create the application files	43
Lesson 3: Set up the M-Business Anywhere Server and Client	45
Lesson 4: Add startup code to your application	47
Lesson 5: Add inserts to your application	50
Lesson 6: Add navigation to your application	55
Lesson 7: Add updates and deletes to your application	56
Lesson 8: Add synchronization to your application	58

4 UltraLite for M-Business Anywhere API Reference	61
Data types in UltraLite for M-Business Anywhere	62
Class AuthStatusCode	63
Class Connection	64
Class ConnectionParms	72
Class DatabaseManager	75
Class DatabaseSchema	79
Class IndexSchema	84
Class PreparedStatement	87
Class PublicationSchema	95
Class ResultSet	96
Class ResultSetSchema	103
Class SQLError	106
Class SQLType	110
Class SyncParms	112
Class SyncResult	120
Class TableSchema	126
Class ULTable	134
Class UUID	154
Index	155

About This Manual

Subject	<p>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.</p> <p>M-Business Anywhere is iAnywhere's platform for developing and deploying mobile web-based applications. The previous name for the product was AvantGo M-Business Server.</p>
Audience	<p>This manual is intended for application developers who wish to take advantage of the performance, resource efficiency, robustness, and security of an UltraLite relational database for data storage and synchronization.</p>

SQL Anywhere Studio documentation

The SQL Anywhere Studio 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 Studio documentation is available in a variety of forms: in an online form that combines all books in one large help file; as separate PDF files for each book; and as printed books that you can purchase. The documentation consists of the following books:

- ◆ **Introducing SQL Anywhere Studio** This book provides an overview of the SQL Anywhere Studio database management and synchronization technologies. It includes tutorials to introduce you to each of the pieces that make up SQL Anywhere Studio.
- ◆ **What's New in SQL Anywhere Studio** This book is for users of previous versions of the software. It lists new features in this and previous releases of the product and describes upgrade procedures.
- ◆ **Adaptive Server Anywhere Database Administration Guide** This book covers material related to running, managing, and configuring databases and database servers.
- ◆ **Adaptive Server Anywhere SQL User's Guide** 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.
- ◆ **Adaptive Server Anywhere SQL Reference Manual** This book provides a complete reference for the SQL language used by Adaptive Server Anywhere. It also describes the Adaptive Server Anywhere system tables and procedures.
- ◆ **Adaptive Server Anywhere Programming Guide** This book describes how to build and deploy database applications using the C, C++, and Java programming languages. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by those tools. It also describes the Adaptive Server Anywhere ADO.NET data provider.
- ◆ **Adaptive Server Anywhere SNMP Extension Agent User's Guide** This book describes how to configure the Adaptive Server Anywhere SNMP Extension Agent for use with SNMP management applications to manage Adaptive Server Anywhere databases.
- ◆ **Adaptive Server Anywhere Error Messages** This book provides a complete listing of Adaptive Server Anywhere error messages together with diagnostic information.

-
- ◆ **SQL Anywhere Studio Security Guide** This book provides information about security features in Adaptive Server Anywhere databases. Adaptive Server Anywhere 7.0 was awarded a TCSEC (Trusted Computer System Evaluation Criteria) C2 security rating from the U.S. Government. This book may be of interest to those who wish to run the current version of Adaptive Server Anywhere in a manner equivalent to the C2-certified environment.
 - ◆ **MobiLink Administration Guide** This book describes how to use the MobiLink data synchronization system for mobile computing, which enables sharing of data between a single Oracle, Sybase, Microsoft or IBM database and many Adaptive Server Anywhere or UltraLite databases.
 - ◆ **MobiLink Clients** This book describes how to set up and synchronize Adaptive Server Anywhere and UltraLite remote databases.
 - ◆ **MobiLink Server-Initiated Synchronization User's Guide** This book describes MobiLink server-initiated synchronization, a feature of MobiLink that allows you to initiate synchronization from the consolidated database.
 - ◆ **MobiLink Tutorials** This book provides several tutorials that walk you through how to set up and run MobiLink applications.
 - ◆ **QAnywhere User's Guide** This manual describes MobiLink QAnywhere, a messaging platform that enables the development and deployment of messaging applications for mobile and wireless clients, as well as traditional desktop and laptop clients.
 - ◆ **iAnywhere Solutions ODBC Drivers** This book describes how to set up ODBC drivers to access consolidated databases other than Adaptive Server Anywhere from the MobiLink synchronization server and from Adaptive Server Anywhere remote data access.
 - ◆ **SQL Remote User's Guide** This book describes all aspects of the SQL Remote data replication system for mobile computing, which enables sharing of data between a single Adaptive Server Anywhere or Adaptive Server Enterprise database and many Adaptive Server Anywhere databases using an indirect link such as e-mail or file transfer.
 - ◆ **SQL Anywhere Studio Help** This book includes the context-sensitive help for Sybase Central, Interactive SQL, and other graphical tools. It is not included in the printed documentation set.
 - ◆ **UltraLite Database User's Guide** This book is intended for all UltraLite developers. It introduces the UltraLite database system and provides information common to all UltraLite programming interfaces.

-
- ◆ **UltraLite Interface Guides** A separate book is provided for each UltraLite programming interface. Some of these interfaces are provided as UltraLite components for rapid application development, and others are provided as static interfaces for C, C++, and Java development.

In addition to this documentation set, PowerDesigner and InfoMaker include their own online documentation.

Documentation formats SQL Anywhere Studio provides documentation in the following formats:

- ◆ **Online documentation** The online documentation contains the complete SQL Anywhere Studio documentation, including both 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 9 ► 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.

- ◆ **PDF books** The SQL Anywhere books are provided as a set of PDF files, viewable with Adobe Acrobat Reader.

The PDF books are accessible from the online books, or from the Windows Start menu.

- ◆ **Printed books** The complete set of books is available from Sybase sales or from eShop, the Sybase online store, at <http://eshop.sybase.com/eshop/documentation>.

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 upper case, 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.

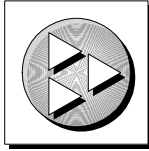
```
[ QUOTES { ON | OFF } ]
```

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

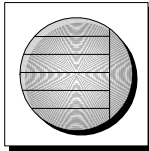
Graphic icons

The following icons are used in this documentation.

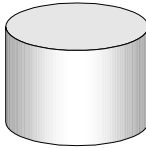
- ◆ A client application.



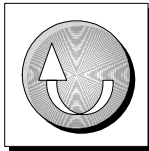
- ◆ A database server, such as Sybase Adaptive Server Anywhere.



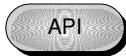
- ◆ A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



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



- ◆ A programming interface.



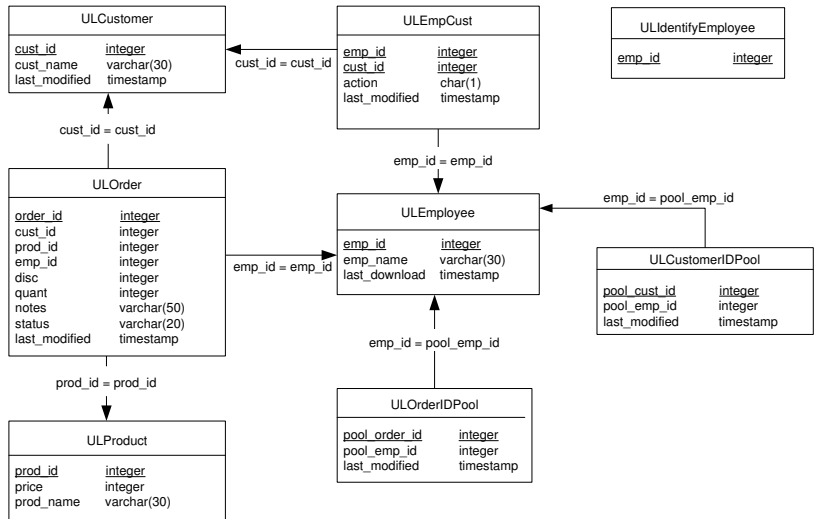
The CustDB sample database

Many of the examples in the MobiLink and UltraLite documentation use the UltraLite sample database.

The reference database for the UltraLite sample database is held in a file named *custdb.db*, and is located in the *Samples\UltraLite\CustDB* subdirectory of your SQL Anywhere directory. A complete application built on this database is also supplied.

The sample database is a sales-status database for a hardware supplier. It holds customer, product, and sales force information for the supplier.

The following diagram shows the tables in the CustDB database and how they are related to each other.



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 Studio. You can find this information by typing **dbeng9 -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 e-mail comments and suggestions to the SQL Anywhere documentation team at iasdoc@ianywhere.com. Although we do not reply to e-mails 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

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 [“Welcome to UltraLite”](#) [*UltraLite Database User’s Guide*, page 3].

Contents

Topic:	page
UltraLite for M-Business Anywhere features	2
UltraLite for M-Business Anywhere architecture	3

UltraLite for M-Business Anywhere features

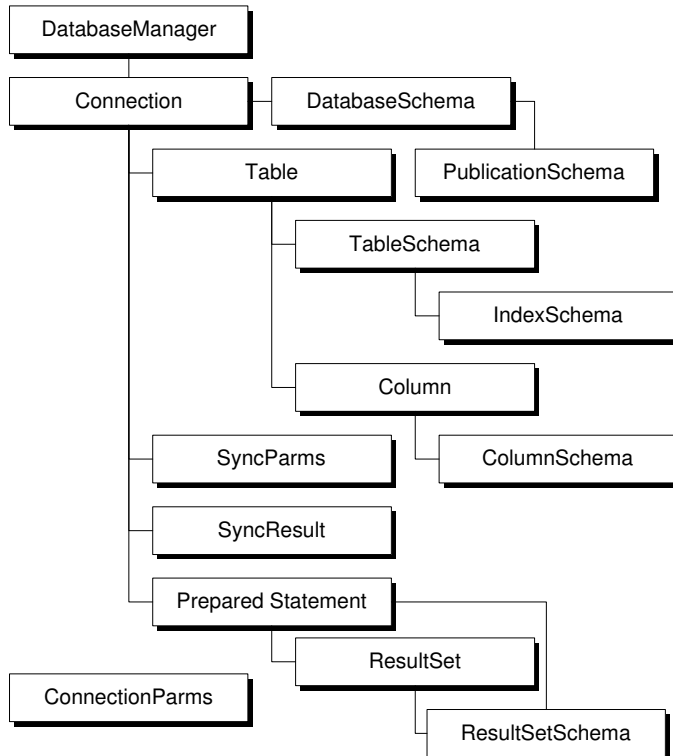
UltraLite for M-Business Anywhere is a relational data management system for mobile devices. It has the performance, resource efficiency, robustness, and security required by business applications. UltraLite also provides synchronization with enterprise data stores.

System requirements and supported platforms

Development platforms	<p>To develop applications using UltraLite for M-Business Anywhere, you require the following:</p> <ul style="list-style-type: none">◆ 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. <p>☞ For more information, see “UltraLite development platforms” [Introducing SQL Anywhere Studio, page 99].</p>
Target platforms	<p>UltraLite for M-Business Anywhere supports the following target platforms:</p> <ul style="list-style-type: none">◆ Windows CE 3.0 and higher, with Pocket PC on the ARM processor.◆ Palm OS version 5.0 and higher.◆ Windows 2000 and Windows XP, starting with M-Business Anywhere 5.5. <p>☞ For more information, see “UltraLite target platforms” [Introducing SQL Anywhere Studio, page 109].</p>

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 75](#).
- ◆ **ConnectionParms** holds a set of connection parameters.
 ➞ For more information, see [“Class ConnectionParms” on page 72](#).
- ◆ **Connection** represents a database connection, and governs transactions.
 ➞ For more information, see [“Class Connection” on page 64](#).
- ◆ **PreparedStatement, ResultSet, and ResultSetSchema** manage database requests and their results using SQL.

☞ For more information, see “Class `PreparedStatement`” on page 87, “Class `ResultSet`” on page 96, and “Class `ResultSetSchema`” on page 103.

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

☞ For more information, see “Class `ULTable`” on page 134.

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

☞ For more information about synchronization with MobiLink, see “UltraLite Clients” [*MobiLink Clients*, page 277].

CHAPTER 2

Understanding UltraLite for M-Business Anywhere Development

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

Contents	Topic:	page
	UltraLite for M-Business Anywhere Quick Start	6
	Working with the database schema	11
	Connecting to an UltraLite database	13
	Maintaining connections and application state across pages	16
	Encryption and obfuscation	17
	Working with data using dynamic SQL	18
	Working with data using the table API	23
	Accessing schema information	30
	Handling errors	31
	Authenticating users	32
	Synchronizing data	33
	Deploying UltraLite for M-Business Anywhere applications	37

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 5.3 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\UltraLiteForMBusinessAnywhere\custdb* subdirectory of your SQL Anywhere installation.

- b. Run the following command:

```
build.bat install-dir
```

where *install-dir* is the directory to which the CustDB and UltraLite files should be deployed.

The batch file copies the required files into the location you specify. To see what files are being copied, open

Samples\UltraLiteForMBusinessAnywhere\custdb\build.bat in a text editor.

2. Create a virtual directory in your web server that points to the install directory *install-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 point it at your install directory *install-dir*. 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:
 - ◆ dll
 - ◆ usm
 - ◆ cab
 - ◆ pdb
 - ◆ prc
 - 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. Click the Users button.
- c. Click the New User link. The Create New 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 Create to add the user, or click Create and Edit to create the user and edit the user's details.

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.
- b. On the handheld device or PC, configure AvantGo 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 once to install the client onto the device.
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:8092`.
- b. Log in using the administrator account.
- c. Click the Groups tab, and click New 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 new group UltraLite Samples.
 - b. User the group's "web" tab, create the following channel.

Setting	Value
Channel Name	CustDB
Location	http://localhost/CustDB/main.htm or the URL from step 2.
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:
 - ◆ `ulpod9.dll`
 - ◆ `ul_custdb.usm`

If any of these files are 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`
 - ◆ `ul_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` to the device.
 - ◆ On Windows, after you have synchronized your device, check that the following files are in the `AvantGo\Pods` subdirectory of your `AvantGo Connect` folder:

- ◆ ulpod9.dll
- ◆ ul_custdb.usm

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 synchronization server on your desktop.

From the Start menu, choose Programs ► SQL Anywhere 9 ► MobiLink ► Synchronization Server Sample.

- b. Start the CustDB application on your AvantGo Client.

The CustDB application is a link on your AvantGo 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 “[The CustDB Sample Application](#)” [*MobiLink Tutorials*, page 99].

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 “[Understanding HotSync synchronization](#)” [*MobiLink Clients*, page 296].

All UltraLite for M-Business Anywhere applications inherit the creator ID of the AvantGo 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 “[Understanding the Palm Creator ID](#)” [*UltraLite Database User's Guide*, page 191].

Working with the database schema

The schema is the structure of a database. It is the collection of table definitions, index definitions, and publication definitions within the database, and all the relationships between them.

You create UltraLite databases by creating an UltraLite database schema file and apply that file to a database by calling a function in your application.

Creating UltraLite database schema files

You can create an UltraLite schema file using the UltraLite Schema Painter or the *ulinit* utility.

- ◆ **UltraLite Schema Painter** The UltraLite Schema Painter is a graphical utility for creating and editing UltraLite schema files.

To start the Schema painter, choose Start ► Programs ► SQL Anywhere 9 ► UltraLite ► UltraLite Schema Painter, or double-click a schema (.usm) file in Windows Explorer.

☞ For more information about using the UltraLite Schema Painter, see “Lesson 1: Create an UltraLite database schema” [*UltraLite Database User’s Guide*, page 130].

- ◆ **The ulinit utility** If you have the Adaptive Server Anywhere database management system, you can generate an UltraLite schema file using the *ulinit* command line utility.

☞ For more information about using the *ulinit* utility, see “The *ulinit* utility” [*UltraLite Database User’s Guide*, page 112].

Changing the schema of a database

To change the schema of an existing database, create a schema file with the new schema and apply this schema to the existing database. In most cases there will be no data loss, but data loss can occur if columns are deleted or if the data type for a column is changed to an incompatible type.

☞ For more information about these methods, see “Method `applyFileWithParms`” on page 79 and “Method `applyFile`” on page 79.

You can also change the schema by executing the following SQL statements:

- ◆ “CREATE INDEX statement” [*UltraLite Database User’s Guide*, page 172]
- ◆ “CREATE TABLE statement” [*UltraLite Database User’s Guide*, page 173]
- ◆ “DROP INDEX statement” [*UltraLite Database User’s Guide*, page 178]

-
- ◆ “DROP TABLE statement” [*UltraLite Database User’s Guide*, page 178]“DROP INDEX statement” [*UltraLite Database User’s Guide*, page 178]

Connecting to an UltraLite database

UltraLite applications must connect to a database before carrying out operations on the data in it. This section describes how to connect to an UltraLite database.

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 64](#).

- ◆ **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 66](#).

- ◆ **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 32](#).

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

☞ For more information, see [“Synchronizing data” on page 33](#).

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

☞ For more information, see [“Method getTable” on page 68](#).

Connecting to a database

You can connect to a database using either a `ConnectionParms` object or a connection string. Methods that use a `ConnectionParms` object allow you to manipulate connection parameters with ease and accuracy. Methods that use a connection string require that you successfully create a connection string.

Using ConnectionParms to connect to a database

The following procedure uses a `ConnectionParms` object to connect to an UltraLite database.

❖ To connect to an UltraLite database using ConnectionParms

1. Create a DatabaseManager object.

You should create only one DatabaseManager object per application. This object is at the root of the object hierarchy. For this reason, it is often best to declare the DatabaseManager object as global to the application.

```
dbMgr = CreateObject(
    "iAnywhere.UltraLite.DatabaseManager.myDB" );
if ( dbMgr == null || dbMgr.sqlCode < 0 ) {
    alert(
        "Fatal Error! Cannot create database manager object: "
        + dbMgr.sqlCode );
}
```

2. Declare a ConnectionParms object and locate the directory where M-Business Anywhere places the downloaded files.

For example:

```
var parm = dbMgr.createConnectionParms();
var dir = dbMgr.directory;
```

3. Set the required properties of the ConnectionParms object.

For example:

```
parm.connectionName = "simpleCon";

parm.schemaOnPalm = "ul_custapi";
parm.databaseOnPalm = "AvGo";
parm.databaseOnCE = dir + "\\ul_custapi.udb";
parm.schemaOnCE = dir + "\\ul_custapi.usm";
```

4. Open a connection to the database.

CreateDatabaseWithParms and OpenConnectionWithParms return an open connection as a Connection object. Each method takes a single ConnectionParms object as its argument.

For example:

```
try {
    Connection = dbMgr.reOpenConnection( "custdb" );
    if ( Connection == null ) {
        Connection = dbMgr.openConnectionWithParms( parm );
    }
} catch ( ex ) {
    if ( DatabaseMgr.sqlCode !=
        DatabaseMgr.SQLError.SQLE_ULTRALITE_DATABASE_NOT_
        FOUND ) {
        alert( "Database Error! " + ex.getMessage() );
        return;
    }
}
```

☞ For more information, see [“Method createDatabaseWithParams”](#) on page 76 and [“Method openConnectionWithParams”](#) on page 77.

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.

See also

- ◆ [“Method reOpenConnection” on page 78](#)
- ◆ [“Class PreparedStatement” on page 87](#)
- ◆ [“Class ResultSet” on page 96](#)
- ◆ [“Class ULTable” on page 134](#)
- ◆ [“Class PreparedStatement” on page 87](#)

Encryption and obfuscation

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

☞ For more information about database encryption, see [“Encrypting UltraLite databases”](#) [*UltraLite Database User’s Guide*, page 36].

Encryption

To create a database with encryption, set the `ConnectionParms.encryptionKey` property. When you call `CreateDatabaseWithParms` and pass in the `ConnectionParms` object, the database created and encrypted with the specified key.

☞ For more information about the `EncryptionKey` property, see [“Encryption Key connection parameter”](#) [*UltraLite Database User’s Guide*, page 75] and [“Method `changeEncryptionKey`”](#) on page 65.

Example

You can change the encryption key by specifying the new encryption key on the `Connection` object. In this example, “apricot” is the encryption key.

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

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

Obfuscation

To obfuscate the database, specify `obfuscate=1` in the `ConnectionParms.additionalParms` string. For example:

```
var open_parms = dbMgr.createConnectionParms();  
open_parms.additionalParms = "obfuscate=1";  
Connection = dbMgr.createDatabase( open_parms );
```

☞ For more information, see [“Properties”](#) on page 64.

Working with data using dynamic SQL

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

☞ For information about the Table API, see [“Working with data using the table API” on page 23](#).

This section explains how to perform the following tasks using dynamic 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 dynamic SQL features, see [“Dynamic SQL” \[UltraLite Database User’s Guide, page 159\]](#).

☞ The sequence of operations required is similar for any SQL operation. For an overview, see [“Using dynamic SQL” \[UltraLite Database User’s Guide, page 161\]](#).

Data manipulation: INSERT, UPDATE and DELETE

With UltraLite, you can perform SQL Data Manipulation Language 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 87](#).

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);
```

4. Execute the statement.

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

❖ To UPDATE a row

1. Declare a PreparedStatement object.

```
var PrepStmt;
```

2. 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.

3. 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);
```

4. Execute the statement

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

❖ To DELETE a row

1. Declare a PreparedStatement object.

```
var PrepStmt;
```

2. 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.

-
3. Assign parameter values for the statement.

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

4. 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.

☞ For more information about `ResultSet` objects, see “[Class `ResultSet`](#)” on [page 96](#).

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 dynamic SQL](#)” on [page 21](#).

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\UltraLiteForMBusinessAnywhere\CustDB\custdb.js*.

Navigation with dynamic SQL

UltraLite for M-Business Anywhere provides you with a number of methods to navigate a result set in order 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 96](#).

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 dynamic SQL or the Table API. This section describes data access using the Table API.

☞ For information about dynamic SQL, see [“Working with data using dynamic SQL”](#) on page 18.

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 in order 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.moveNext()) {
    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 134](#).
- ◆ **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 134](#).

❖ 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 wish 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 UTable 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 28](#).

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 wish 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 142](#).

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.

☞ For information about modifying the schema, see [“Changing the schema of a database” on page 11](#).

- ◆ **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 106](#).

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

- ◆ createDatabase
- ◆ createDatabaseWithParms
- ◆ dropDatabase
- ◆ dropDatabaseWithParms
- ◆ openConnection
- ◆ openConnectionWithParms

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 64](#) and [“Class DatabaseManager” on page 75](#).

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 68](#) and [“Method revokeConnectFrom” on page 69](#).

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

1. Connect to the database as a user with DBA authority.
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 a user with DBA authority.
2. Revoke the user's connection authority as follows.

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

Synchronizing data

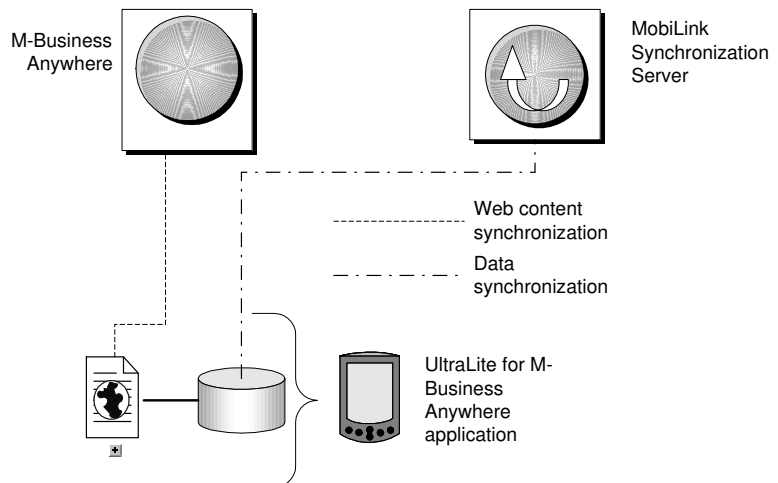
UltraLite for M-Business Anywhere applications typically involve two kinds of synchronization:

- ◆ **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 synchronization 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 116](#).

Otherwise, use the standard methods to set synchronization parameters. For more information, see [“Class SyncParms” on page 112](#).

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 70](#).

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 33](#).

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

Synchronization requires the MobiLink synchronization 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.

Note

To synchronize using encrypted synchronization (HTTPS) or to use encryption over TCP/IP you must obtain the separately-licensable security option. To order this option, see the card in your SQL Anywhere Studio package or see <http://www.sybase.com/detail?id=1015780>.

☞ For more information, see “Welcome to SQL Anywhere Studio” [[Introducing SQL Anywhere Studio, page 4](#)].

❖ **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 Clients, page 277](#)].

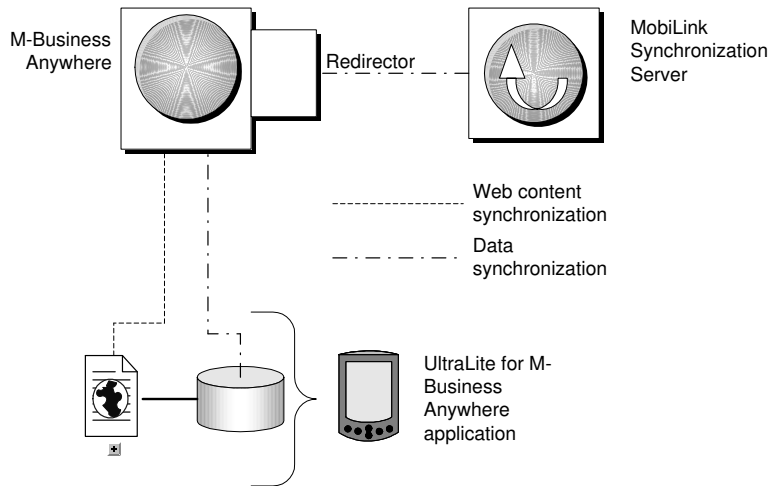
2. Synchronize.

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 synchronization 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 synchronization server.
 - ☞ For information on the MobiLink Redirector for M-Business Anywhere, see [“M-Business Anywhere Redirector”](#) [*MobiLink Administration Guide*, page 153].
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 116.
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 33, and [“Synchronizing data”](#) on page 34.

Deploying UltraLite for M-Business Anywhere applications

When you have completed your application or when you wish 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 or schema.
 - ☞ For instructions, see [“UltraLite for M-Business Anywhere Quick Start” on page 6](#).

In many situations it is sufficient to deploy an UltraLite schema file only. UltraLite creates a database file from the schema on the first connection attempt. You can then use synchronization to load an initial copy of the data.

You must place the database or schema file so that it can be located by the application. The Database On CE and Schema On CE connection parameters define the location for Windows CE. The Database on Desktop and Schema on Desktop define the location for Windows XP.

☞ See [“Database On CE connection parameter” \[UltraLite Database User’s Guide, page 69\]](#), [“Schema On CE connection parameter” \[UltraLite Database User’s Guide, page 78\]](#), [“Database On Desktop connection parameter” \[UltraLite Database User’s Guide, page 70\]](#), and [“Schema On Desktop connection parameter” \[UltraLite Database User’s Guide, page 79\]](#).

Deploying applications that use one-button synchronization

One-button synchronization requires a set of files, including *ulconnect.exe* and *ulconnect.usm*. These files are packaged together in the file *ultralite\UltraLiteForMBusinessAnywhere\ulpod.cab*. When you deploy the cab file to a Windows CE device, it installs its contents in the proper locations automatically. For Windows XP, you must deploy the files manually.

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 or schema.

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

In many situations it is sufficient to deploy an UltraLite schema file only. UltraLite creates a database file from the schema on the first connection attempt. 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 the Schema Painter, *ulxml*, and *ulinit*.

You must supply a schema or 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 [“Database On Palm connection parameter” \[UltraLite Database User’s Guide, page 71\]](#), and [“Schema On Palm connection parameter” \[UltraLite Database User’s Guide, page 80\]](#). For information about using the virtual file system, see [“VFS On Palm parameter” \[UltraLite Database User’s Guide, page 81\]](#).

- ◆ Deploy the MobiLink synchronization conduit for HotSync.

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

☞ For instructions, see [“Deploying the MobiLink HotSync conduit” \[MobiLink Clients, page 302\]](#).

CHAPTER 3

Tutorial: A Sample Application for M-Business Anywhere

About this chapter

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

Contents

Topic:	page
Introduction	40
Lesson 1: Create a project architecture	41
Lesson 2: Create the application files	43
Lesson 3: Set up the M-Business Anywhere Server and Client	45
Lesson 4: Add startup code to your application	47
Lesson 5: Add inserts to your application	50
Lesson 6: Add navigation to your application	55
Lesson 7: Add updates and deletes to your application	56
Lesson 8: Add synchronization to your application	58

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 in order 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 schema using the UltraLite Schema Painter. For more information, see [“The UltraLite Schema Painter”](#) [*UltraLite Database User’s Guide*, page 124].

Lesson 1: Create a project architecture

The first procedure describes how to create an UltraLite database schema. The database schema is a description of the database. It describes the tables, indexes, keys, and publications within the database, and all the relationships between them.

☞ For more information about database schemas, see “[Creating UltraLite database schema files](#)” on page 11.

❖ To create an UltraLite database schema

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`:

- ◆ `dll`
- ◆ `usm`
- ◆ `prc`
- ◆ `pdb`

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 schema using the UltraLite Schema Painter.

☞ For more information about creating a database schema, see the “Lesson 1: Create an UltraLite database schema” [*UltraLite Database User’s Guide*, page 130].

◆ **Schema filename** tutcustomer.usm.

◆ **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 schema.

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

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 schema 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 schema 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="ulpod9.prc"></a>
  <a href="tutCustomer.pdb"></a>
</html>

<!-- WINCE_OS\ul_deps.html -->
<html>
  <a href="AG_DEVICEPROCESSOR/ulpod9.dll"></a>
  <a href="tutCustomer.usm"></a>
</html>

<!-- WIN32_OS\ul_deps.html -->
<html>
  <a href="ulpod9.dll"></a>
  <a href="tutCustomer.usm"></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 schema be in the proper location relative to the tutorial directory. The schema file is already in place from earlier in the tutorial. You must now copy the UltraLite runtime library into place.

In the following instructions, paths for the source files are relative to your SQL Anywhere installation.

-
- ◆ For the Palm OS, copy *ulpod9.prc* from *UltraLite\UltraLiteForMBusinessAnywhere\palm\68k* to *c:\Tutorial\mbus\PALM_OS*.
 - ◆ For Windows CE, copy *ulpod9.dll* from *UltraLite\UltraLiteForMBusinessAnywhere\ce\arm* to *c:\Tutorial\mbus\WINCE_OS\arm*.
 - ◆ For Windows CP, copy *ulpod9.dll* from *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.

❖ 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 a blank 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 wish to use a user name that already exists.

- a. Click the Users heading and click New User.
 - b. Enter a User Name and Password. 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 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://host:port/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** 2
 - ◆ **Allow Binary Distribution** Yes (checked).
 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 group and click Update to add the user to this group.

The user, group, and channel are now all 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 wish 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**

1. 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>
  </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.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.schemaOnCE = dir + "\\tutCustomer.usm";  
  
    connParms.schemaOnDesktop = dir + "\\tutCustomer.usm";  
    connParms.databaseOnDesktop = dir + "\\tutCustomer.udb";  
  
    connParms.userID = "dba";  
    connParms.password = "sql";
```

```
try {
    // try to connect to an existing database
    Connection = DatabaseMgr.openConnectionWithParms(
        connParms );
    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.createDatabaseWithParms(
                connParms );
            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, so that 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

1. 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 forwards and backwards 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 ULSyncParms object. For example, the ULSyncParms.UserName property specifies the user name that MobiLink looks for. The ULSyncParms.SendColumnNames property specifies that the column names will be 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 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 118](#) and [“Method setStreamParms” on page 118](#)

```
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 ASA 9.0 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 ASA 9.0 Sample database.

❖ **To synchronize your application**

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

```
dbmlsrv9 -c "dsn=ASA 9.0 Sample" -v+ -zu+ -za
```

The `-zu+` and `-za` command line options provide automatic addition of users and generation of synchronization scripts. For more information about these options, see [“MobiLink Synchronization Server Options” \[MobiLink Administration Guide, page 189\]](#).

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 ASA 9.0 Sample database.

3. Synchronize your application.

Click Synchronize. The MobiLink synchronization 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

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 beneath.

Contents

Topic:	page
Data types in UltraLite for M-Business Anywhere	62
Class AuthStatusCode	63
Class Connection	64
Class ConnectionParms	72
Class DatabaseManager	75
Class DatabaseSchema	79
Class IndexSchema	84
Class PreparedStatement	87
Class PublicationSchema	95
Class ResultSet	96
Class ResultSetSchema	103
Class SQLError	106
Class SQLType	110
Class SyncParms	112
Class SyncResult	120
Class TableSchema	126
Class ULTable	134
Class UUID	154

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 time-related data types (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	1000	User ID and password were valid at time of synchronization.
VALID_BUT_EXPIRES_SOON	2000	User ID and password were valid at time of synchronization but will expire soon.
EXPIRED	3000	User ID or password has expired; authorization failed.
INVALID	4000	Bad user ID or password; authorization failed.
IN_USE	5000	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.openConnectionWithParms
- ◆ DatabaseManager.createDatabase
- ◆ DatabaseManager.createDatabaseWithParms

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.
Boolean databaseNew (read-only)	Indicates if the database was new when this connection was opened (true), or if the database already existed when this connection was opened (false).

Prototype	Description
String openParms (read-only)	Gets the open parameters string as a semicolon-separated list of name=value pairs. See “ Connection Parameters ” [<i>UltraLite Database User’s Guide</i> , page 63].
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). See “ One-button synchronization ” on page 33.
Int32 sqlCode (read-only)	Gets the SQL Code of the last operation on this connection. The SQL Code is the standard Adaptive Server 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 ” [<i>MobiLink Clients</i> , page 316].
SyncResult syncResult (read-only)	Gets the results of the most recent synchronization for this connection. See “ Synchronization parameters ” [<i>MobiLink Clients</i> , page 316].
INVALID_DATABASE_ID (read-only)	A constant indicating an invalid database.

Method `changeEncryptionKey`

Prototype	<code>changeEncryptionKey(String newKey)</code>
Parameters	◆ newKey The new encryption key for the database.
Remarks	Changes the database’s 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	<p>Closes this connection.</p> <p>Once a connection is closed, it can not be reopened. To reopen a connection, a new connection object must be created and opened.</p> <p>It is an error to use any object (table, schema, etc.) associated with a closed connection.</p> <p>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.</p>

Method commit

Prototype	commit()
Remarks	Commits outstanding changes to the database.

Method countUploadRow

Prototype	UInt32 countUploadRow (UInt32 <i>mask</i> , UInt32 <i>threshold</i>)
Parameters	<ul style="list-style-type: none">◆ mask set of publications to check. See PublicationSchema class for more information.◆ 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	<p>Gets the current Database ID value, as set by setDatabaseID().</p> <p>If the value has not been set, the constant Connection.INVALID_DATABASE_ID is returned.</p>

Method `getGlobalAutoIncrementUsage`

Prototype	<code>UInt16 getGlobalAutoIncrementUsage()</code>
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 <code>setDatabaseID</code> .

Method `getLastDownloadTime`

Prototype	<code>Date getLastDownloadTime(UInt32 mask)</code>
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 <code>PublicationSchema.SYNC_ALL_DB</code> for the time of the last download of the full database.
See also	◆ “Class PublicationSchema” on page 95

Method `getLastIdentity`

Prototype	<code>UInt64 getLastIdentity()</code>
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. This method is enabled only if user authentication was enabled when the connection was opened.

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-compile 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 from an UltraLite database for a user ID.

Method `rollback`

Prototype **`rollback()`**

Remarks Rolls back outstanding changes to the database.

Method `rollbackPartialDownload`

Prototype **`rollbackPartialDownload()`**

Remarks Roll 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 `[0,0xffffffff]`.

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 33](#)

Method **startSynchronizationDelete**

Prototype **startSynchronizationDelete()**

Remarks Mark 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 Synchronize 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 Synchronize 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 semi-colons.  See “Additional Parms connection parameter” [<i>UltraLite Database User's Guide</i> , page 68].
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 User's Guide</i> , page 73].
String connectionName (read-write)	A name for the connection. The connection name is used to share a single connection across multiple web pages.  See “Connection Name connection parameter” [<i>UltraLite Database User's Guide</i> , page 74], and “Maintaining connections and application state across pages” on page 16.

Prototype	Description
String databaseOnCE (read-write)	<p>The filename of the database deployed to PocketPC.</p> <p>☞ See “Database On CE connection parameter” [<i>UltraLite Database User’s Guide</i>, page 69].</p>
String databaseOnDesktop (read-write)	<p>The filename of the database deployed to Windows XP.</p> <p>☞ See “Database On Desktop connection parameter” [<i>UltraLite Database User’s Guide</i>, page 70].</p>
String databaseOnPalm (read-write)	<p>The UltraLite database creator ID on the Palm device.</p> <p>☞ See “Database On Palm connection parameter” [<i>UltraLite Database User’s Guide</i>, page 71].</p>
String encryptionKey (read-write)	<p>A key for encrypting the database. OpenConnection and OpenConnectionWithParms must use the same key as specified during database creation. Suggestions for keys are:</p> <ol style="list-style-type: none"> 1. Select an arbitrary, lengthy string 2. Select strings with a variety of numbers, letters and special characters, so as to decrease the chances of key penetration. <p>☞ See “Encryption Key connection parameter” [<i>UltraLite Database User’s Guide</i>, page 75].</p>
String parmsUsed (read-only)	<p>The connection parameters used by the DatabaseManager. Useful for debugging purposes.</p>
String password (read-write)	<p>The password for an authenticated user. Databases are initially created with one authenticated user 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>.</p> <p>☞ See “Password connection parameter” [<i>UltraLite Database User’s Guide</i>, page 76].</p>

Prototype	Description
String schemaOnCE (read-write)	The schema filename deployed to PocketPC.  See “ Schema On CE connection parameter ” [<i>UltraLite Database User’s Guide</i> , page 78].
String schemaOnDesktop (read-write)	The schema filename deployed to Windows XP.  See “ Schema On Desktop connection parameter ” [<i>UltraLite Database User’s Guide</i> , page 79].
String schemaOnPalm (read-write)	The schema PDB on the Palm device.  See “ Schema On Palm connection parameter ” [<i>UltraLite Database User’s Guide</i> , page 80].
String userID (read-write)	The authenticated user for the database. Databases are initially created with one authenticated user DBA. The UserID is case-insensitive if the database is case-insensitive and case-sensitive if the database is case-sensitive. The default value is <i>DBA</i> .  See “ User ID connection parameter ” [<i>UltraLite Database User’s Guide</i> , page 76].
Boolean VFSONPalm (read-write)	Indicates whether the Palm database is on a virtual file system (true) or on the Palm store (false).  See “ VFS On Palm parameter ” [<i>UltraLite Database User’s Guide</i> , page 81].

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. This directory is where M-Business Anywhere places the downloaded schema file. On Palm OS, this property is NULL.
UInt32 runtimeType	Set the runtime type to be either the UltraLite runtime library or the UltraLite database engine. The value is an enum, and is one of the following: 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 for the most recent operation.
SQLType SQLType (read-only)	Gets the SQLType object for the most recent operation.
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 *parms*)

Parameters	◆ parms parameters for creating the database and opening a connection to it. Parameter keywords are case-insensitive, and most values are case-sensitive. parms is used to create the database, specify the schema for the newly-created database, and open the connection.
Returns	An opened Connection object.
Remarks	<p>Creates a database and opens a connection to the database as specified by parms. If the database already exists, a <code>SQLiteDatabaseNotCreated</code> exception is thrown.</p> <p>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.</p> <p><code>Connection.databaseNew</code> is set to true to indicate that the database was created when the connection was opened. 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.</p>

Method `createDatabaseWithParams`

Prototype	<code>Connection createDatabaseWithParams(ConnectionParams parms)</code>
Parameters	◆ parms A <code>ConnectionParams</code> object holding parameters for creating the database and opening a connection to it.
Returns	An opened connection object
Remarks	<p>Creates a database and opens a connection to the database as specified by parms. If the database already exists, an error is thrown. You can check <code>Connection.sqlCode</code> within the error catching code to identify the cause of the error.</p> <p>Only one database may be active at a given time. Attempts to open a connection to a database will result in an error if there are connections open to a different database.</p> <p><code>Connection.databaseNew</code> is set to true to indicate that the database was created when the connection was opened. 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.</p>

Method `dropDatabase`

Prototype	<code>dropDatabase(String parms)</code>
-----------	---

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 can not drop a database that has open connections.

Method dropDatabaseWithParms

Prototype	dropDatabaseWithParms (ConnectionParms <i>parms</i>)
Parameters	◆ parms parameters for identifying a database. For more information see " Class ConnectionParms " on page 72.
Remarks	Deletes the specified database file. You can not drop a database that has open connections.

Method openConnection

Prototype	Connection openConnection (String <i>parms</i>)
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 will result in an error. Connection.databaseNew is set to false to indicate that the database was not created when the connection was opened. 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.

Method openConnectionWithParms

Prototype	Connection openConnectionWithParms (ConnectionParms <i>parms</i>)
Parameters	

	◆ parms A ConnectionParms object holding the parameters for opening a connection.
Returns	An opened Connection object.
Remarks	<p>Opens a connection to the database specified by parms. If the database does not exist, an exception is thrown. You can check Connection.sqlCode to find out more about the cause of the error.</p> <p>Only one database may be active at a given time. Attempts to open a connection to different database while other connections are open will result in an error.</p> <p>Connection.databaseNew is set to false to indicate that the database was not created when the connection was opened. 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.</p>

Method reOpenConnection

Prototype	Connection reOpenConnection (String <i>connectionName</i>)
Parameters	◆ persistName The name of the connection to be reopened, as specified in the Connection.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 applyFile

Prototype

applyFile(String *parms*)

Parameters

◆ **parms** parameters for specifying the schema to be applied to the database.

Remarks

Applies a database schema file to the database. All instances of **TableSchema**, **IndexSchema**, and **PublicationSchema** will be invalidated and will need to be replaced.

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.

Method applyFileWithParms

Prototype

applyFileWithParms(ConnectionParms *parms*)

Parameters

◆ **parms** A ConnectionParms object holding parameters for specifying the schema to be applied to the database.

Remarks

Applies a database schema file to the database. All instances of **TableSchema**, **IndexSchema**, and **PublicationSchema** will be invalidated and will need to be replaced.

Method getCollationName

Prototype

String **getCollationName**()

Returns

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

Method `getDatabaseProperty`

Prototype	String <code>getDatabaseProperty</code> (String <i>name</i>)
Parameters	◆ name name of the database property.
Returns	Returns the value of the specified database property. Recognized properties are: <ul style="list-style-type: none">◆ "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 <code>getDateFormat</code> ()
Returns	The date format used for string conversions.

Method `getDateOrder`

Prototype	String <code>getDateOrder</code> ()
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.

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

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`.

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 `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.

Column IDs and count may change during a schema upgrade. Column IDs from an index are different than the column IDs in a table or another index.

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*'th column in this index.

Column IDs and count may change during a schema upgrade. Column IDs from an index are different than the column IDs in a table or another 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 <i>name</i>)
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 <code>isUniqueKey()</code>
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	<pre>appendBytesParameter(UInt16 <i>parameterID</i>, Array <i>value</i>, UInt32 <i>srcOffset</i>, UInt32 <i>count</i>)</pre>
Parameters	<ul style="list-style-type: none"> ◆ 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	<p>Appends the specified subset of the specified array of bytes to the new value for the specified <code>SQLType.LONGBINARY</code> column. The bytes at position <code>srcOffset</code> (starting from 0) through <code>srcOffset+count-1</code> 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.</p> <p>If any of the following is true, an Error with code <code>SQLException.SQLE_INVALID_PARAMETER</code> is thrown and the destination is not modified:</p> <ul style="list-style-type: none"> ◆ 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 <i>parameterID</i> , String <i>value</i> ,)
Parameters	<ul style="list-style-type: none">◆ 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: <pre>for (i = 0; i < 100; i++){ stmt.appendStringChunkParameter(1, "XYZ"); }</pre>

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 <i>persistName</i>)
Parameter	<ul style="list-style-type: none">◆ 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 will be committed 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 optimization” [<i>UltraLite Database User’s Guide</i> , page 185].

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 <i>parameterID</i> , Boolean <i>value</i>)
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	<code>setBytesParameter(UInt16 parameterID, Array value)</code>
Parameters	<ul style="list-style-type: none">◆ 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 <code>SQLType.BINARY</code> or <code>SQLType.LONGBINARY</code> only.
Example	The following statement sets a value for the first parameter: <pre>var blob = new Array(3); blob[0] = 78; blob[1] = 0; blob[2] = 68; stmt.setBytesParameter(1, blob);</pre>

Method `setDateParameter`

Prototype	<code>setDateParameter(UInt16 parameterID, Date value)</code>
Parameters	<ul style="list-style-type: none">◆ 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 <code>SQLType.DATE</code> 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 2004/10/27: <pre>stmt.setDateParameter(1, new Date(2004,9,27,0,0,0));</pre>

Method `setDoubleParameter`

Prototype	<code>setDoubleParameter(UInt16 parameterID, Double value)</code>
Parameters	<ul style="list-style-type: none">◆ 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 <i>parameterID</i> , Float <i>value</i>)
Parameters	<ul style="list-style-type: none">◆ 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 SQLite.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 <i>parameterID</i> , UInt16 <i>value</i>)
Parameters	<ul style="list-style-type: none">◆ 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 <i>parameterID</i> , Int64 <i>value</i>)
Parameters	<ul style="list-style-type: none">◆ 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	<code>setNullParameter(UInt16 parameterID)</code>
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	<code>setUInt16Parameter(UInt16 parameterID, UInt16 value)</code>
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	<code>setStringParameter(UInt16 parameterID, String value)</code>
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	<code>setTimeParameter(UInt16 parameterID, Date value)</code>
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 1966/05/01 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. See class `Unsigned64` for more information.

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 close

Prototype **close()**

Remarks Frees all resources associated with this object.

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	<pre> UInt32 getBytesSection(UInt16 <i>index</i> UInt32 <i>srcOffset</i>, Array <i>dst</i>, UInt32 <i>dstOffset</i>, UInt32 <i>count</i>) </pre>
Parameters	<p>index The 1-based ordinal of the column containing the binary data.</p> <p>offset The offset into the underlying array of bytes. The source offset must be greater than or equal to 0, otherwise a <code>SQLITE_INVALID_PARAMETER</code> error will be raised. A buffer bigger than 64K is also permissible.</p> <p>data An array of bytes.</p> <p>data_len The length of the buffer, or array. The <code>data_len</code> must be greater than or equal to 0.</p>
Returns	The number of bytes read.
Remarks	<p>Copies a subset of the value for the specified <code>SQLType.LONGBINARY</code> column, beginning at the specified offset, to the specified offset of the destination byte array.</p> <p>The bytes at position <code>srcOffset</code> (starting from 0) through <code>srcOffset+count-1</code> of the value are copied into positions <code>dstOffset</code> through <code>dstOffset+count-1</code>, 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.</p> <p>If any of the following is true, an error is thrown, <code>SQLite</code> code <code>SQLCode.SQLITE_INVALID_PARAMETER</code> is set, and the destination is not modified:</p> <ul style="list-style-type: none"> ◆ The <code>dst</code> argument is null ◆ The <code>srcOffset</code> argument is negative ◆ The <code>dstOffset</code> argument is negative ◆ The <code>count</code> argument is negative ◆ <code>dstOffset + count</code> is greater than the length of the destination array, <code>dst.length</code>.
Errors set	<p>SQLITE_CONVERSION_ERROR The error occurs if the column data type is not <code>BINARY</code> or <code>LONG BINARY</code>.</p> <p>SQLITE_INVALID_PARAMETER The error occurs if the column data type is <code>BINARY</code> and the offset is not 0 or 1, or, the data length is less than 0.</p>

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 getInteger

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(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 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 if 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	Boolean <code>moveAfterLast()</code>
Remarks	Moves to a position after the last row of the <code>ULResultSet</code> .
Returns	true if successful. false if unsuccessful. The method fails, for example, if there are no rows.

Method `moveBeforeFirst`

Prototype	Boolean <code>moveBeforeFirst()</code>
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 <code>moveFirst()</code>
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 <code>moveLast()</code>
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 <code>moveNext()</code>
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 <code>movePrevious()</code>
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 <code>moveRelative(Int32 <i>index</i>)</code>
Remarks	Moves a certain number of rows relative to the current row. Relative to the current position of the cursor in the resultset, positive index values move forward in the resultset, negative index values move backward in the resultset 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.

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 <i>name</i>)
Parameters	◆ name 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 <i>columnID</i>)
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 <i>name</i>)
Parameters	◆ name name of the column.
Returns	The precision of the named column. The column must be of type SQLite.NUMERIC.

Method getColumnPrecisionByColID

Prototype	Int32 getColumnPrecisionByColID (UInt16 <i>columnID</i>)
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 <code>SQLType.NUMERIC</code> .

Method getColumnScale

Prototype	Int32 getColumnScale (String <i>name</i>)
Parameters	◆ name name of the column.
Returns	The scale of the column. The column must be of type <code>SQLType.NUMERIC</code> .

Method getColumnScaleByColID

Prototype	Int32 getColumnScaleByColID (UInt16 <i>columnID</i>)
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 <code>SQLType.NUMERIC</code> .

Method getColumnSize

Prototype	UInt32 getColumnSize (String <i>name</i>)
Parameters	◆ name name of the column.
Returns	The size of the named column. The column must be of type <code>SQLType.NUMERIC</code> .

Method getColumnSizeByColID

Prototype	UInt32 getColumnSizeByColID (UInt16 <i>columnID</i>)
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 <code>SQLType.NUMERIC</code> .

Method getColumnSQLType

Prototype	UInt16 getColumnSQLType (String <i>name</i>)
-----------	--

Parameters ♦ **name** 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 can not be directly instantiated.

☞ For more information about the meaning of these constants, see [“MobiLink Communication Error Messages” \[ASA Error Messages, page 549\]](#).

Constant	Value
<code>SQLException_AGGREGATES_NOT_ALLOWED</code>	-150
<code>SQLException_ALIAS_NOT_UNIQUE</code>	-830
<code>SQLException_ALIAS_NOT_YET_DEFINED</code>	-831
<code>SQLException_BAD_ENCRYPTION_KEY</code>	-840
<code>SQLException_BAD_PARAM_INDEX</code>	-689
<code>SQLException_CANNOT_ACCESS_FILE</code>	-602
<code>SQLException_CANNOT_CHANGE_USER_NAME</code>	-867
<code>SQLException_CANNOT_MODIFY</code>	-191
<code>SQLException_CANNOT_EXECUTE_STMT</code>	-111
<code>SQLException_COLUMN_AMBIGUOUS</code>	-144
<code>SQLException_COLUMN_CANNOT_BE_NL</code>	-195
<code>SQLException_COLUMN_IN_INDEX</code>	-127
<code>SQLException_COLUMN_NOT_FOUND</code>	-143
<code>SQLException_COMMUNICATIONS_ERROR</code>	-85
<code>SQLException_CONNECTION_NOT_FOUND</code>	-108
<code>SQLException_CONVERSION_ERROR</code>	-157
<code>SQLException_CURSOROP_NOT_ALLOWED</code>	-187
<code>SQLException_CURSOR_ALREADY_OPEN</code>	-172
<code>SQLException_CURSOR_NOT_OPEN</code>	-180
<code>SQLException_DATABASE_ERROR</code>	-301
<code>SQLException_DATABASE_NEW</code>	123
<code>SQLException_DATABASE_NOT_CREATED</code>	-645

Constant	Value
SQLiteDatabaseNotFound	-83
SQLiteDatabaseUpgradeFailed	-672
SQLiteDatabaseUpgradeNotPossible	-673
SQLiteDatatypeNotAllowed	-624
SQLiteDbSpaceFl	-604
SQLiteDivZeroError	-628
SQLiteDownloadConflict	-839
SQLiteDropDatabaseFailed	-651
SQLiteDynamicMemoryExhausted	-78
SQLiteEngineAlreadyRunning	-96
SQLiteEngineNotMtiuser	-89
SQLiteError	-300
SQLiteErrorCallingFunction	-622
SQLiteExpressionError	-156
SQLiteIdentifierTooLong	-250
SQLiteIndexNotFound	-183
SQLiteIndexNotUnique	-196
SQLiteInterrupted	-299
SQLiteInvalidAggregatePlacement	-862
SQLiteInvalidForeignKey	-194
SQLiteInvalidForeignKeyDef	-113
SQLiteInvalidGroupSelect	-149
SQLiteInvalidLogon	-103
SQLiteInvalidOptionSetting	-201
SQLiteInvalidOrder	-152
SQLiteInvalidOrderByColumn	-854
SQLiteInvalidParameter	-735

Constant	Value
SQL_INVALID_SQL_IDENTIFIER	-760
SQL_INVALID_STATEMENT	-130
SQL_LOCKED	-210,
SQL_MEMORY_ERROR	-309
SQL_METHOD_CANNOT_BE_CALLED	-669
SQL_NAME_NOT_UNIQUE	-110
SQL_NOERR	0
SQL_NOTFOUND	100
SQL_NOT_IMPLEMENTED	-134
SQL_NO_CURRENT_ROW	-197
SQL_NO_INDICATOR	-181
SQL_OVERFLOW_ERROR	-158
SQL_PERMISSION_DENIED	-121
SQL_PRIMARY_KEY_NOT_UNIQUE	-193
SQL_PRIMARY_KEY_VALUE_REF	-198
SQL_PUBLICATION_NOT_FOUND	-280
SQL_RESOURCE_GOVERNOR_EXCEEDED	-685
SQL_ROW_DROPPED_DURING_SCHEMA_UPGRADE	130
SQL_SERVER_SYNCHRONIZATION_ERROR	-857
SQL_START_STOP_DATABASE_DENIED	-75
SQL_STATEMENT_ERROR	-132
SQL_SYNTAX_ERROR	-131
SQL_STRING_RIGHT_TRUNCATION	-638
SQL_TABLE_HAS_PUBLICATIONS	-281
SQL_TABLE_IN_USE	-214
SQL_TABLE_NOT_FOUND	-141
SQL_TOO_MANY_CONNECTIONS	-102

Constant	Value
SQLE_UTRALITE_OBJ_CLOSED	-908
SQLE_UNABLE_TO_CONNECT_OR_START	-764
SQLE_UNABLE_TO_START_DATABASE	-82
SQLE_UNCOMMITTED_TRANSACTIONS	-755
SQLE_UNKNOWN_FUNC	-148
SQLE_UNKNOWN_USERID	-140
SQLE_UNSUPPORTED_CHARACTER_SET_ERROR	-869
SQLE_UPLOAD_FAILED_AT_SERVER	-794
SQLE_WRONG_PARAMETER_COUNT	-154

Class `SQLType`

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

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

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_-UNKNOWN	0	No stream type has been set. You must set a stream type before synchronization.
STREAM_TYPE_-TCPIP	1	TCP/IP stream
STREAM_TYPE_HTTP	2	HTTP stream
STREAM_TYPE_-HTTPS	3	HTTPS 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 will perform extra checkpoints, false if the client will only perform 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 will be 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 will only ping the server, false if client will perform a synchronization.

Method `getPublicationMask`

Prototype UInt32 `getPublicationMask();`

Returns The publications to be synchronized. See `PublicationSchema` class for more information.

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 will send column names to the MobiLink synchronization server during synchronization, false if client will not send column names.

Method `getSendDownloadAck`

Prototype Boolean `getSendDownloadAck()`

Returns true if client will provide a download acknowledgement to the MobiLink server, false if the client will 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 will 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 will be 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 pay 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 will block on UltraLite calls until the synchronization has completed.

Method `setDownloadOnly`

Prototype	<code>setDownloadOnly</code> (Boolean <i>value</i>)
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 `setMBA Server`

Prototype	<code>setMBA Server</code> (String <i>host</i> , String <i>port</i> , String <i>url_suffix</i>)
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 <code>url_suffix</code> parameter set in the <code>sync.conf</code> 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. You should use the MobiLink redirector for M-Business Anywhere to route data to and from the MobiLink synchronization server. If you are using one-button synchronization, you must save the synchronization parameters using <code>Connection.saveSyncParms</code> . ☞ For information about configuring M-Business Server to route HTTP database traffic through the M-Business Anywhere Redirector, see “M-Business Anywhere Redirector” [<i>MobiLink Administration Guide</i> , page 153].

Method `setNewPassword`

Prototype	<code>setNewPassword</code> (String <i>value</i>)
Parameters	◆ value new password for MobiLink user.
Remarks	Sets a new MobiLink password for the user specified with <code>setUserName</code> . The new password will take effect after the next synchronization.

Method `setPassword`

Prototype	<code>setPassword</code> (String <i>value</i>)
Parameters	◆ value 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 synchronization server.

Method `setPingOnly`

Prototype **`setPingOnly`**(Boolean *value*);

Parameters ♦ **value** set to true to only ping the MobiLink synchronization server, or to false to perform a synchronization.

Remarks Specifies whether the client should only ping the MobiLink synchronization 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. See **`PublicationSchema`** class for more information.

Method `setSendColumnNames`

Prototype **`setSendColumnNames`**(Boolean *value*)

Parameters ♦ **value** set to true to send column names, or set to false to not send column names.

Remarks Specifies whether the client should send column names to the MobiLink synchronization server during synchronization.

This parameter is typically used together with the `-za` or `-ze` switch on the MobiLink synchronization server for automatically generating synchronization scripts.

Method `setSendDownloadAck`

Prototype **`setSendDownloadAck`**(Boolean *value*)

Parameters ♦ **value** set to true to send a download acknowledgement, to false to tell the server that no download acknowledgement will be sent.

Remarks Specifies whether the client should send a download acknowledgement to the MobiLink synchronization server during synchronization.

If the client does send a download acknowledgement, the MobiLink synchronization server worker thread must wait for the client to apply the

download. If the client does not send a download acknowledgement, the MobiLink synchronization server is freed up sooner for its next synchronization.

Method `setStream`

Prototype	<code>setStream(UInt16 value)</code>
Parameters	◆ value type of MobiLink synchronization stream to use for synchronization. Valid choices are listed in For a list of valid choices, see “Constants” on page 112 .
Remarks	Sets the MobiLink synchronization stream to use for synchronization. Most synchronization streams require parameters to identify the MobiLink synchronization server address and control other behavior. These parameters are supplied with the <code>setStreamParms()</code> method. If the specified stream type is invalid for the platform, the stream type will be set to UNKNOWN .

Method `setStreamParms`

Prototype	<code>setStreamParms(String value)</code>
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	<code>setUploadOnly(Boolean value)</code>
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	<code>setUserName(String value)</code>
Parameters	◆ value MobiLink user name.

Remarks Sets the user name that uniquely identifies the MobiLink client to the MobiLink synchronization 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 synchronization 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 can not 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” \[ASA Error Messages, page 549\]](#)

Value	Description
0	None
1	Parameter

Value	Description
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
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

Value	Description
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
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

Value	Description
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

Value	Context
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 SQLite.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 `SQLType.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 `SQLType.BINARY` or `SQLType.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 <i>indexID</i>)
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 <i>name</i>)
Parameters	◆ name name of the column.
Returns	The optimal index for searching a table using the named column. The named column will be 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 <code>isColumnCurrentTimeByColID(UInt16 columnID)</code>
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 <code>SQLType.TIME</code> .

Method `isColumnCurrentTimestamp`

Prototype	Boolean <code>isColumnCurrentTimestamp(String name)</code>
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 <code>SQLType.TIMESTAMP</code> .

Method `isColumnCurrentTimestampByColID`

Prototype	Boolean <code>isColumnCurrentTimestampByColID(UInt16 columnID)</code>
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 <code>SQLType.TIME</code> .

Method `isColumnGlobalAutoIncrement`

Prototype	Boolean <code>isColumnGlobalAutoIncrement(String name)</code>
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 <code>isColumnGlobalAutoincrementByColID(UInt16 columnID)</code>
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.SQLC_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	appendChars (UInt16 <i>columnID</i> , String <i>value</i>)
Parameters	<ul style="list-style-type: none"> ◆ 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 <code>SQLType.LONGVARCHAR</code> column.
Example	<p>The following statements append one hundred instances of the string XYZ to the value in the first column:</p> <pre>for (i = 0; i < 100; i++){ t.appendStringChunk(1, "XYZ"); }</pre>

Method deleteRow

Prototype	deleteRow ()
Remarks	Deletes the current row.

Method deleteAllRows

Prototype	deleteAllRows ()
Remarks	<p>Deletes all rows in the table.</p> <p>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 <code>Connection.startSynchronizationDelete</code> method.</p>

Method `findBegin`

Prototype	<code>findBegin()</code>
Remarks	Prepares to perform a new find on this table. The value(s) to search for are specified by calling the appropriate <code>setType</code> method(s) on the columns in the index this table was opened with.

Method `findFirst`

Prototype	Boolean <code>findFirst()</code>
Returns	true if successful, false otherwise
Remarks	<p>Move forwards through the table from the beginning, looking for a row that exactly matches a value or full set of values in the current index.</p> <p>To specify the value 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()).</p> <p>Each search must be preceded by a call to <code>findBegin()</code>.</p>

Method `findFirstForColumns`

Prototype	Boolean <code>findFirstForColumns(UInt16 numColumns)</code>
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	<p>Move forwards through the table from the beginning, looking for a row that exactly matches a value or partial set of values in the current index.</p> <p>To specify the value 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()).</p> <p>Each search must be preceded by a call to <code>findBegin()</code>.</p>

Method findLast

Prototype	Boolean findLast()
Returns	true if successful, false otherwise.
Remarks	<p>Move backwards through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.</p> <p>To specify the value to search for, 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()).</p> <p>Each search must be preceded by a call to findBegin().</p>

Method findLastForColumns

Prototype	Boolean findLastForColumns(UInt16 numColumns)
Parameters	<ul style="list-style-type: none">◆ 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	<p>Move backwards through the table from the end, looking for a row that exactly matches a value or partial set of values in the current index.</p> <p>To specify the value to search for, 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()).</p> <p>Each search must be preceded by a call to findBegin().</p>

Method findNext

Prototype	Boolean findNext()
Returns	true if successful, false otherwise.
Remarks	<p>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.</p> <p>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()).</p>

findNext 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()**).

findNext 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()**).

findPrevious 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()). findPrevious behavior is undefined if the column values being searched for are modified during a row update.

Method getBoolean

Prototype	Boolean getBoolean (UInt16 <i>index</i>)
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 <i>index</i>)
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 <code>SQLType.BINARY</code> or <code>SQLType.LONGBINARY</code> .

Method getBytesSection

Prototype	UInt32 getBytesSection (UInt16 <i>index</i> UInt32 <i>srcOffset</i> , Array <i>dst</i> , UInt32 <i>dstOffset</i> , UInt32 <i>count</i>)
-----------	--

Parameters	<p>index The 1-based ordinal of the column containing the binary data.</p> <p>srcOffset The start position in the column value. Zero is the beginning of the value.</p> <p>dst The destination array.</p> <p>dstOffset The start position in the destination array.</p> <p>count The number of bytes to be copied</p>
Returns	The number of bytes read.
Remarks	<p>Copies a subset of the value for the specified <code>SQLType.LONGBINARY</code> column, beginning at the specified offset, to the specified offset of the destination byte array.</p> <p>The bytes at position <code>srcOffset</code> (starting from 0) through <code>srcOffset+count-1</code> of the value are copied into positions <code>dstOffset</code> through <code>dstOffset+count-1</code>, respectively, of the destination array. If the end of the value is encountered before <code>count</code> bytes are copied, the remainder of the destination array is left unchanged.</p> <p>If any of the following is true, an Error is thrown, <code>Connection.sqlCode</code> set to <code>SQLException.SQL_INVALID_PARAMETER</code> and the destination is not modified:</p> <ul style="list-style-type: none"> ◆ The <code>dst</code> argument is null ◆ The <code>srcOffset</code> argument is negative ◆ The <code>dstOffset</code> argument is negative ◆ The <code>count</code> argument is negative ◆ <code>dstOffset + count</code> is greater than <code>dst.length</code>, the length of the destination array.

Method getDate

Prototype	Date getDate (UInt16 <i>index</i>)
Parameters	index The 1-based ordinal in the result set to get.
Returns	The value as a Date.

Method getDouble

Prototype	Double getDouble (UInt16 <i>index</i>)
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 <i>index</i> , UInt32 <i>srcOffset</i> , UInt32 <i>count</i>)
Parameters	<ul style="list-style-type: none">◆ 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 <code>SQLType.LONGVARCHAR</code> column, starting at the specified offset, to the String object.

Method getTime

Prototype	Date getTime (UInt16 <i>index</i>)
Parameters	index The 1-based ordinal in the result set to get.
Returns	The value as a Date.

Method getTimestamp

Prototype	Date getTimestamp (UInt16 <i>index</i>)
Parameters	index The 1-based ordinal in the result set to get.
Returns	The value as a Date.

Method getULong

Prototype	UInt64 getULong (UInt16 <i>index</i>)
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 <i>index</i>)
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()**, and that change is not permanent until it is committed.

Method lookupBackward

Prototype Boolean **lookupBackward()**

Returns **true** if successful, **false** otherwise.

Remarks Move backwards 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 to search for, 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 **lookupBegin()**.

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 backwards 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 to search for, 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 **lookupBegin()**.

Method lookupBegin

Prototype **lookupBegin()**

Remarks Prepares to perform a new lookup on this table. The value(s) to search for are specified by calling the appropriate *setType* method(s) on the columns in the index this table was opened with.

Method lookupForward

Prototype Boolean **lookupForward()**

Returns **true** if successful, **false** otherwise.

Remarks Move forwards 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 to search for, 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 **lookupBegin()**.

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 forwards 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 to search for, 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 lookupBegin() .

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 <code>moveBeforeFirst()</code>
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 <code>moveFirst()</code>
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 <code>moveLast()</code>
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 <code>moveNext()</code>
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 <code>movePrevious()</code>
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 (<i>Int32 index</i>)
Remarks	Moves a certain number of rows relative to the current row. Relative to the current position of the cursor in the resultset, positive index values move forward in the resultset, negative index values move backward in the resultset 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 (<i>String index</i>)
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 (<i>short columnID</i> , <i>boolean value</i>)
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 <i>columnID</i> , Array <i>value</i>)
Parameters	<ul style="list-style-type: none">◆ 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: <pre>var blob = new Array(3); blob[0] = 78; blob[1] = 0'; blob[2] = 68; t.setBytes(1, blob);</pre>

Method setDate

Prototype	setDate (UInt16 <i>columnID</i> , Date <i>value</i>)
Parameters	<ul style="list-style-type: none">◆ 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 2004/10/27: <pre>t.setDate(1, new Date(2002,9,27,0,0,0));</pre>

Method setDouble

Prototype	setDouble (UInt16 <i>columnID</i> , Double <i>value</i>)
Parameters	<ul style="list-style-type: none">◆ 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	<ul style="list-style-type: none">◆ 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 <i>columnID</i>)
Parameters	<ul style="list-style-type: none">◆ 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 <i>columnID</i> , Int16 <i>value</i>)
Parameters	<ul style="list-style-type: none">◆ 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 <i>columnID</i> , String <i>value</i>)
Parameters	<ul style="list-style-type: none">◆ 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 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 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 1966/05/01 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 <i>columnID</i> , UInt64 <i>value</i>)
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 <i>columnID</i> , UUID <i>value</i>)
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 <code>SQLType.BINARY</code> and length 16.
Example	The following statement sets a new UUID value for the first column in the table:
See also	◆ “Maintaining unique primary keys using UUIDs” [<i>MobiLink Administration Guide</i> , page 56]

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 set*Type* 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 `SQLiteType.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 (<code>UUID other</code>)
Parameters	◆ other <code>UUID</code> with which to compare.
Returns	true if this <code>UUID</code> is the same as the other argument, false otherwise.

Method toString

Prototype	String toString ()
Returns	A string representation of this <code>UUID</code> .
Remarks	The string is of the format <code>XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX</code> , where X is a hexadecimal digit or null if the <code>Connection</code> associated with the <code>UUID</code> is closed.

Index

A

accessing schema information
 UltraLite for M-Business Anywhere 30

API reference
 UltraLite for M-Business Anywhere API 61

APIs
 UltraLite for M-Business Anywhere 61

appendBytes method
 UltraLite for M-Business Anywhere API 134

appendBytesParameter method
 UltraLite for M-Business Anywhere API 87

appendStringChunk method
 UltraLite for M-Business Anywhere API 135

appendStringChunkParameter method
 UltraLite for M-Business Anywhere API 88

applyFile method
 UltraLite for M-Business Anywhere API 79

applyFileWithParms method
 UltraLite for M-Business Anywhere API 79

architecture
 UltraLite for M-Business Anywhere 3

AuthStatusCode properties
 UltraLite for M-Business Anywhere API 63

AutoCommit mode
 UltraLite for M-Business Anywhere 28

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 28
 UltraLite for M-Business Anywhere 28

C

casting
 data types in UltraLite for M-Business Anywhere 25

changeEncryptionKey method
 UltraLite for M-Business Anywhere API 65

close method
 UltraLite for M-Business Anywhere API 66, 88, 96

columns
 accessing schema information in UltraLite for M-Business Anywhere 30

Columns collection
 UltraLite for M-Business Anywhere 23

Commit method
 UltraLite for M-Business Anywhere 28

commit method
 UltraLite for M-Business Anywhere API 66

commits
 UltraLite for M-Business Anywhere 28

connecting
 UltraLite for M-Business Anywhere databases 13

conventions
 documentation viii

countUploadRow method
 UltraLite for M-Business Anywhere API 66

createDatabase method
 UltraLite for M-Business Anywhere API 75

createDatabaseWithParms method
 UltraLite for M-Business Anywhere API 76

creator IDs
 UltraLite for M-Business Anywhere 9

D

data manipulation
 dynamic SQL in UltraLite for M-Business Anywhere 18
 table API in UltraLite for M-Business Anywhere 23
 UltraLite for M-Business Anywhere 18

data types
 accessing in UltraLite for M-Business Anywhere 24
 casting in UltraLite for M-Business Anywhere 25
 JavaScript 62
 UltraLite for M-Business Anywhere 62

database schemas	
accessing in UltraLite for M-Business Anywhere	
30	
UltraLite for M-Business Anywhere	11
databases	
accessing schema information in UltraLite for	
M-Business Anywhere	30
connecting in UltraLite for M-Business	
Anywhere	13
DatabaseSchema class	
UltraLite for M-Business Anywhere development	
30	
deleteAllRows method	
UltraLite for M-Business Anywhere API	135
deleteRow method	
UltraLite for M-Business Anywhere API	135
deleting	
rows in UltraLite for M-Business Anywhere	26
deploying	
UltraLite for M-Business Anywhere	37
UltraLite for M-Business Anywhere applications	
to Palm OS	37
UltraLite for M-Business Anywhere to Windows	
CE	37
UltraLite for M-Business Anywhere to Windows	
XP	37
development platforms	
UltraLite for M-Business Anywhere	2
DML operations	
UltraLite for M-Business Anywhere	18
documentation	
conventions	viii
SQL Anywhere Studio	vi
dropDatabase method	
UltraLite for M-Business Anywhere API	76
dropDatabaseWithParams method	
UltraLite for M-Business Anywhere API	77
dynamic SQL	
UltraLite for M-Business Anywhere development	
18	
E	
encryption	
UltraLite for M-Business Anywhere development	
17	
equals method	
UltraLite for M-Business Anywhere API	154

error handling	
UltraLite for M-Business Anywhere	31
errors	
handling in UltraLite for M-Business Anywhere	
31	
executeQuery method	
UltraLite for M-Business Anywhere API	88
executeStatement method	
UltraLite for M-Business Anywhere API	88
F	
features	
for M-Business Anywhere	2
feedback	
documentation	xii
providing	xii
find methods	
UltraLite for M-Business Anywhere	25
find mode	
UltraLite for M-Business Anywhere	26
findBegin method	
UltraLite for M-Business Anywhere API	136
findFirst method	
UltraLite for M-Business Anywhere API	136
findFirstForColumns method	
UltraLite for M-Business Anywhere API	136
findLast method	
UltraLite for M-Business Anywhere API	137
findLastForColumns method	
UltraLite for M-Business Anywhere API	137
findNext method	
UltraLite for M-Business Anywhere API	137
findNextForColumns method	
UltraLite for M-Business Anywhere API	138
findPrevious method	
UltraLite for M-Business Anywhere API	138
findPreviousForColumns method	
UltraLite for M-Business Anywhere API	138
firewalls	
M-Business Anywhere synchronization	35
G	
getAuthenticationParams method	
UltraLite for M-Business Anywhere API	112
getAuthStatus method	
UltraLite for M-Business Anywhere API	120
getBoolean method	

-
- UltraLite for M-Business Anywhere API 96, 139
 - GetByteChunk method
 - UltraLite for M-Business Anywhere 28
 - getBytes method
 - UltraLite for M-Business Anywhere API 96, 139
 - getBytesSection method
 - UltraLite for M-Business Anywhere API 97, 139
 - getCheckpointStore method
 - UltraLite for M-Business Anywhere API 112
 - getCollationName method
 - UltraLite for M-Business Anywhere API 79
 - getColumnCount method
 - UltraLite for M-Business Anywhere API 84, 103, 126
 - getColumnDefaultValue method
 - UltraLite for M-Business Anywhere API 126
 - getColumnDefaultValueByColID method
 - UltraLite for M-Business Anywhere API 126
 - getColumnID method
 - UltraLite for M-Business Anywhere API 103, 126
 - columnName method
 - UltraLite for M-Business Anywhere API 84, 103
 - getColumnPartitionSize method
 - UltraLite for M-Business Anywhere API 126
 - getColumnPartitionSizeByColID method
 - UltraLite for M-Business Anywhere API 127
 - getColumnPrecision method
 - UltraLite for M-Business Anywhere API 103, 127
 - getColumnPrecisionByColID method
 - UltraLite for M-Business Anywhere API 104, 127
 - getColumnScale method
 - UltraLite for M-Business Anywhere API 104, 127
 - getColumnScaleByColID method
 - UltraLite for M-Business Anywhere API 104, 127
 - getColumnSize method
 - UltraLite for M-Business Anywhere API 104, 128
 - getColumnSizeByColID method
 - UltraLite for M-Business Anywhere API 104, 128
 - getColumnSQLType method
 - UltraLite for M-Business Anywhere API 104, 128
 - getColumnSQLTypeByColID method
 - UltraLite for M-Business Anywhere API 105, 128
 - getDatabaseID method
 - UltraLite for M-Business Anywhere API 66
 - getDatabaseProperty method
 - UltraLite for M-Business Anywhere API 80
 - getDate method
 - UltraLite for M-Business Anywhere API 98, 140
 - getDateFormat method
 - UltraLite for M-Business Anywhere API 80
 - getDateOrder method
 - UltraLite for M-Business Anywhere API 80
 - getDisableConcurrency method
 - UltraLite for M-Business Anywhere API 112
 - getDouble method
 - UltraLite for M-Business Anywhere API 98, 140
 - getDownloadOnly method
 - UltraLite for M-Business Anywhere API 113
 - getFloat method
 - UltraLite for M-Business Anywhere API 98, 141
 - getGlobalAutoIncrementUsage method
 - UltraLite for M-Business Anywhere API 67
 - getIgnoredRows method
 - UltraLite for M-Business Anywhere API 120
 - getIndex method
 - UltraLite for M-Business Anywhere API 128
 - getIndexCount method
 - UltraLite for M-Business Anywhere API 128
 - getIndexName method
 - UltraLite for M-Business Anywhere API 129
 - getInt method
 - UltraLite for M-Business Anywhere API 141
 - getInteger method
 - UltraLite for M-Business Anywhere API 98
 - getKeepPartialDownload method
 - UltraLite for M-Business Anywhere API 113
 - getLastDownloadTime method
 - UltraLite for M-Business Anywhere API 67
 - getLastIdentity method
 - UltraLite for M-Business Anywhere API 67
 - getLong method
 - UltraLite for M-Business Anywhere API 98, 141
 - getMask method
 - UltraLite for M-Business Anywhere API 95
 - getName method
 - UltraLite for M-Business Anywhere API 84, 95, 129
 - getNearestCentury method
 - UltraLite for M-Business Anywhere API 81
 - getNewPassword method
 - UltraLite for M-Business Anywhere API 113
 - getNewUUID method
 - UltraLite for M-Business Anywhere API 68
 - getOptimalIndex method

UltraLite for M-Business Anywhere API	129	getStreamErrorSystem method	
getPartialDownloadRetained method		UltraLite for M-Business Anywhere API	124
UltraLite for M-Business Anywhere API	113, 120	getStreamParms method	
getPassword method		UltraLite for M-Business Anywhere API	114
UltraLite for M-Business Anywhere API	113	getString method	
getPingOnly method		UltraLite for M-Business Anywhere API	99, 141
UltraLite for M-Business Anywhere API	113	getStringChunk method	
getPlan method		UltraLite for M-Business Anywhere API	99, 142
UltraLite for M-Business Anywhere API	89	getTable method	
getPrecision method		UltraLite for M-Business Anywhere API	68
UltraLite for M-Business Anywhere API	81	getTableCount method	
getPrimaryKey method		UltraLite for M-Business Anywhere API	82
UltraLite for M-Business Anywhere API	129	getTableCountInPublications method	
getPublicationCount method		UltraLite for M-Business Anywhere API	82
UltraLite for M-Business Anywhere API	81	getTableName method	
getPublicationMask method		UltraLite for M-Business Anywhere API	82
UltraLite for M-Business Anywhere API	113	getTime method	
getPublicationName method		UltraLite for M-Business Anywhere API	99, 142
UltraLite for M-Business Anywhere API	81	getTimeFormat method	
getPublicationSchema method		UltraLite for M-Business Anywhere API	82
UltraLite for M-Business Anywhere API	81	getTimestamp method	
getReferencedIndexName method		UltraLite for M-Business Anywhere API	99, 125, 142
UltraLite for M-Business Anywhere API	84	getTimestampFormat method	
getReferencedTableName method		UltraLite for M-Business Anywhere API	83
UltraLite for M-Business Anywhere API	84	getULong method	
getResultSetSchema method		UltraLite for M-Business Anywhere API	100, 142
UltraLite for M-Business Anywhere API	89	getUploadOK method	
getResumePartialDownload method		UltraLite for M-Business Anywhere API	125
UltraLite for M-Business Anywhere API	114	getUploadOnly method	
getRowCount method		UltraLite for M-Business Anywhere API	114
UltraLite for M-Business Anywhere API	98, 141	getUploadUnchangedRows method	
getSendColumnNames method		UltraLite for M-Business Anywhere API	129
UltraLite for M-Business Anywhere API	114	getUserName method	
getSendDownloadAck method		UltraLite for M-Business Anywhere API	114
UltraLite for M-Business Anywhere API	114	getUUID method	
getShort method		UltraLite for M-Business Anywhere API	100, 142
UltraLite for M-Business Anywhere API	99, 141	getVersion method	
getSignature method		UltraLite for M-Business Anywhere API	115
UltraLite for M-Business Anywhere API	82	grantConnectTo method	
getStream method		UltraLite for M-Business Anywhere	32
UltraLite for M-Business Anywhere API	114	UltraLite for M-Business Anywhere API	68
getStreamErrorCode method			
UltraLite for M-Business Anywhere API	120		
getStreamErrorContext method			
UltraLite for M-Business Anywhere API	123		
getStreamErrorID method			
UltraLite for M-Business Anywhere API	124		

H

hasResultSet method	
UltraLite for M-Business Anywhere API	89
HotSync	

UltraLite for M-Business Anywhere	9	UltraLite for M-Business Anywhere API	132
HotSync synchronization		isColumnNullableByColID method	132
UltraLite for M-Business Anywhere synchronization parameters	70	UltraLite for M-Business Anywhere API	132
I			
icons		isEOF method	
used in manuals	x	UltraLite for M-Business Anywhere API	100, 145
insert method		isForeignKey method	
UltraLite for M-Business Anywhere API	143	UltraLite for M-Business Anywhere API	85
insert mode		isForeignKeyCheckOnCommit method	85
UltraLite for M-Business Anywhere	26	UltraLite for M-Business Anywhere API	85
insertBegin method		isForeignKeyNullable method	85
UltraLite for M-Business Anywhere API	143	isInPublication method	
inserting		UltraLite for M-Business Anywhere API	132
rows in UltraLite for M-Business Anywhere	26	isNeverSynchronized method	
isBOF method		UltraLite for M-Business Anywhere API	132
UltraLite for M-Business Anywhere API	100, 145	isNull method	
isCaseSensitive method		UltraLite for M-Business Anywhere API	100, 145
UltraLite for M-Business Anywhere API	83	isOpen method	
isColumnAutoIncrement method		UltraLite for M-Business Anywhere API	68, 83, 89, 100, 105, 145
UltraLite for M-Business Anywhere API	130	isPrimaryKey method	
isColumnAutoIncrementByColID method		UltraLite for M-Business Anywhere API	85
UltraLite for M-Business Anywhere API	130	isUniqueIndex method	
isColumnCurrentDate method		UltraLite for M-Business Anywhere API	85
UltraLite for M-Business Anywhere API	130	isUniqueKey method	
isColumnCurrentDateByColID method		UltraLite for M-Business Anywhere API	86
UltraLite for M-Business Anywhere API	130	J	
isColumnCurrentTime method		JavaScript	
UltraLite for M-Business Anywhere API	130	maintaining application state	16
isColumnCurrentTimeByColID method		JavaScript data types	
UltraLite for M-Business Anywhere API	131	UltraLite for M-Business Anywhere	62
isColumnCurrentTimestamp method		JavaScript programming language	
UltraLite for M-Business Anywhere API	131	UltraLite for M-Business Anywhere	61
isColumnCurrentTimestampByColID method		L	
UltraLite for M-Business Anywhere API	131	lookup methods	
isColumnDescending method		UltraLite for M-Business Anywhere	25
UltraLite for M-Business Anywhere API	85	lookup mode	
isColumnGlobalAutoIncrement method		UltraLite for M-Business Anywhere	26
UltraLite for M-Business Anywhere API	131	lookupBackward method	
isColumnGlobalAutoincrementByColID method		UltraLite for M-Business Anywhere API	143
UltraLite for M-Business Anywhere API	131	lookupBackwardForColumns method	
isColumnNewUUID method		UltraLite for M-Business Anywhere API	143
UltraLite for M-Business Anywhere API	132	lookupBegin method	
isColumnNewUUIDByColID method		UltraLite for M-Business Anywhere API	144
UltraLite for M-Business Anywhere API	132	lookupForward method	
isColumnNullable method			

UltraLite for M-Business Anywhere API	144
lookupForwardForColumns method	
UltraLite for M-Business Anywhere API	144

M

M-Business Anywhere	
UltraLite	2
modes	
UltraLite for M-Business Anywhere	26
moveAfterLast method	
UltraLite for M-Business Anywhere API 101,	145
moveBeforeFirst method	
UltraLite for M-Business Anywhere API 101,	146
MoveFirst method	
UltraLite for M-Business Anywhere	23
UltraLite for M-Business Anywhere development	20
moveFirst method	
UltraLite for M-Business Anywhere API 101,	146
moveLast method	
UltraLite for M-Business Anywhere API 101,	146
MoveNext method	
UltraLite for M-Business Anywhere	23
UltraLite for M-Business Anywhere development	20
moveNext method	
UltraLite for M-Business Anywhere API 101,	146
movePrevious method	
UltraLite for M-Business Anywhere API 102,	146
moveRelative method	
UltraLite for M-Business Anywhere API 102,	147

N

network protocol options	
UltraLite for M-Business Anywhere AP	118
newsgroups	
technical support	xii

O

obfuscation	
UltraLite for M-Business Anywhere	17
object hierarchy	
UltraLite for M-Business Anywhere	3
one-button synchronization	
UltraLite for M-Business Anywhere	33
Open method	

ULTable object in UltraLite for M-Business	
Anywhere	20, 23
open method	
UltraLite for M-Business Anywhere API	147
OpenByIndex method	
ULTable object in UltraLite for M-Business	
Anywhere	20
openConnection method	
UltraLite for M-Business Anywhere API	77
openConnectionWithParms method	
UltraLite for M-Business Anywhere API	77

P

passwords	
authentication in UltraLite for M-Business	
Anywhere	32
persistent names	
UltraLite for M-Business Anywhere	16
platforms	
supported in UltraLite for M-Business Anywhere	2
prepared statements	
UltraLite for M-Business Anywhere	18
PreparedStatement	
UltraLite for M-Business Anywhere	18
prepareStatement method	
UltraLite for M-Business Anywhere API	68
publications	
accessing schema information in UltraLite for	
M-Business Anywhere	30
PublicationSchema class	
UltraLite for M-Business Anywhere development	30

R

reOpenConnection method	
UltraLite for M-Business Anywhere API	78
resetLastDownloadTime method	
UltraLite for M-Business Anywhere API	69
revokeConnectFrom method	
UltraLite for M-Business Anywhere	32
UltraLite for M-Business Anywhere API	69
Rollback method	
UltraLite for M-Business Anywhere	28
rollback method	
UltraLite for M-Business Anywhere API	69
rollbackPartialDownload method	

UltraLite for M-Business Anywhere API	69	setDoubleParameter method	
rollbacks		UltraLite for M-Business Anywhere API	90
UltraLite for M-Business Anywhere	28	setDownloadOnly method	
rows		UltraLite for M-Business Anywhere API	116
accessing values in UltraLite for M-Business Anywhere	24	setFloat method	
		UltraLite for M-Business Anywhere API	149
S		setFloatParameter method	
saveSyncParms method		UltraLite for M-Business Anywhere API	91
UltraLite for M-Business Anywhere API	70	setInt method	
schema files		UltraLite for M-Business Anywhere API	149
creating in UltraLite for M-Business Anywhere	11	setIntParameter method	
UltraLite for M-Business Anywhere	11	UltraLite for M-Business Anywhere API	91
schemas		setLong method	
UltraLite for M-Business Anywhere	11, 30	UltraLite for M-Business Anywhere API	149
scope		setLongParameter method	
variables in UltraLite for M-Business Anywhere	16	UltraLite for M-Business Anywhere API	91
scrolling		setMBA Server method	
UltraLite for M-Business Anywhere	23	UltraLite for M-Business Anywhere API	116
security		setNewPassword method	
UltraLite for M-Business Anywhere	16	UltraLite for M-Business Anywhere API	116
SELECT statement		setNull method	
UltraLite for M-Business Anywhere development	20	UltraLite for M-Business Anywhere API	150
setAuthenticationParms method		setNullParameter method	
UltraLite for M-Business Anywhere API	115	UltraLite for M-Business Anywhere API	92
setBoolean method		setPassword method	
UltraLite for M-Business Anywhere API	147	UltraLite for M-Business Anywhere API	116
setBooleanParameter method		setPingOnly method	
UltraLite for M-Business Anywhere API	89	UltraLite for M-Business Anywhere API	117
setBytes method		setPublicationMask method	
UltraLite for M-Business Anywhere API	148	UltraLite for M-Business Anywhere API	117
setBytesParameter method		setSendColumnNames method	
UltraLite for M-Business Anywhere API	90	UltraLite for M-Business Anywhere API	117
setCheckpointStore method		setSendDownloadAck method	
UltraLite for M-Business Anywhere API	115	UltraLite for M-Business Anywhere API	117
setDatabaseID method		setShort method	
UltraLite for M-Business Anywhere API	69	UltraLite for M-Business Anywhere API	150
setDate method		setShortParameter method	
UltraLite for M-Business Anywhere API	148	UltraLite for M-Business Anywhere API	92
setDateParameter method		setStream method	
UltraLite for M-Business Anywhere API	90	UltraLite for M-Business Anywhere API	118
setDisableConcurrency method		setStreamParms method	
UltraLite for M-Business Anywhere API	115	UltraLite for M-Business Anywhere API	118
setDouble method		setString method	
UltraLite for M-Business Anywhere API	148	UltraLite for M-Business Anywhere API	150
		setStringParameter method	
		UltraLite for M-Business Anywhere API	92
		setTime method	

- UltraLite for M-Business Anywhere API 151
- setTimeParameter method
 - UltraLite for M-Business Anywhere API 92
- setTimestamp method
 - UltraLite for M-Business Anywhere API 151
- setTimestampParameter method
 - UltraLite for M-Business Anywhere API 93
- setDefault method
 - UltraLite for M-Business Anywhere API 151
- setULONG method
 - UltraLite for M-Business Anywhere API 152
- setULONGParameter method
 - UltraLite for M-Business Anywhere API 93
- setUpOnly method
 - UltraLite for M-Business Anywhere API 118
- setName method
 - UltraLite for M-Business Anywhere API 118
- setUUID method
 - UltraLite for M-Business Anywhere API 152
- setUUIDParameter method
 - UltraLite for M-Business Anywhere API 93
- setVersion method
 - UltraLite for M-Business Anywhere API 119
- SQL Anywhere Studio
 - documentation vi
- startSynchronizationDelete method
 - UltraLite for M-Business Anywhere API 70
- stopSynchronizationDelete method
 - UltraLite for M-Business Anywhere API 70
- support
 - newsgroups xii
- supported platforms
 - UltraLite for M-Business Anywhere 2
- synchronization
 - HTTP in UltraLite for M-Business Anywhere 35
 - TCP/IP in UltraLite for M-Business Anywhere 35
 - UltraLite for M-Business Anywhere development 33
- synchronize method
 - UltraLite for M-Business Anywhere API 70
- synchronizeWithParm method
 - UltraLite for M-Business Anywhere API 70
- synchronizing UltraLite applications
 - UltraLite for M-Business Anywhere development 33

T

- tables
 - accessing schema information in UltraLite for M-Business Anywhere 30
- TableSchema class
 - UltraLite for M-Business Anywhere development 30
- target platforms
 - UltraLite for M-Business Anywhere 2
- technical support
 - newsgroups xii
- toString method
 - UltraLite for M-Business Anywhere API 63, 110, 154
- transaction processing
 - UltraLite for M-Business Anywhere 28
- transactions
 - UltraLite for M-Business Anywhere 28
- truncate method
 - UltraLite for M-Business Anywhere API 152

U

- ULTable class
 - UltraLite for M-Business Anywhere development 20
- UltraLite databases
 - connecting in UltraLite for M-Business Anywhere 13
- UltraLite for M-Business Anywhere
 - about 1
 - accessing schema information 30
 - architecture 3
 - connecting to UltraLite databases 13
 - data manipulation using dynamic SQL 18
 - data manipulation with Table API 23
 - deploying applications 37
 - deploying applications to Palm OS 37
 - deploying applications to Windows CE 37
 - deploying applications to Windows XP 37
 - encryption 17
 - error handling 31
 - features 2
 - maintaining state 16
 - object hierarchy 3
 - project architecture 41
 - quick start 6

supported platforms	2
synchronizing UltraLite applications	33
User authentication	32
UltraLite for M-Business Anywhere API	
API reference	61
UltraLite for M-Business Anywhere API reference	
alphabetic listing	61
update method	
UltraLite for M-Business Anywhere API	153
update mode	
UltraLite for M-Business Anywhere	26
updateBegin method	
UltraLite for M-Business Anywhere API	153
updating	
rows UltraLite for M-Business Anywhere	26
user authentication	
UltraLite for M-Business Anywhere	32
users	
authentication in UltraLite for M-Business Anywhere	32
usm files	
creating in UltraLite for M-Business Anywhere	11
UltraLite for M-Business Anywhere	11

V

values	
accessing in UltraLite for M-Business Anywhere	24
Visual Basic	
supported versions in UltraLite for M-Business Anywhere	2

W

Windows CE	
target platform in UltraLite for M-Business Anywhere	2