



UltraLite™ for MobileVB User's Guide

Part number: DC36292-01-0902-01
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 MobileVB	1
UltraLite for MobileVB features	2
UltraLite for MobileVB architecture	3
2 Understanding UltraLite for MobileVB Development	5
Preparing to use UltraLite for MobileVB	6
Working with the database schema	9
Connecting to an UltraLite database	11
Encryption and obfuscation	14
Working with data using dynamic SQL	15
Working with data using the table API	21
Accessing schema information	29
Handling errors	30
Authenticating users	31
Synchronizing data	32
Deploying UltraLite applications	35
Maintaining state in UltraLite Palm applications	38
3 Tutorial: A Sample UltraLite for MobileVB Application	41
Introduction	42
Lesson 1: Create a project architecture	43
Lesson 2: Create a form	46
Lesson 3: Write the sample code	48
Lesson 4: Deploy to a device	57
Summary	59
4 Tutorial: A Sample Application for AppForge Crossfire	61
Introduction	62
Lesson 1: Create a project architecture	63
Lesson 2: Create the application interface	66
Lesson 3: Write the sample code	68
Lesson 4: Deploy to a device	77

Summary	79
5 UltraLite for MobileVB API Reference	81
ULAuthStatusCode enumeration	83
ULColumn class	84
ULColumnSchema class	90
ULConnection class	91
ULConnectionParms class	102
ULDatabaseManager class	105
ULDatabaseSchema class	111
ULIndexSchema class	114
ULPreparedStatement class	116
ULPublicationSchema class	121
ULResultSet class	122
ULResultSetSchema class	128
ULSchemaUpgradeState enumeration	129
ULSQLCode enumeration	130
ULSQLType enumeration	134
ULStreamErrorCode enumeration	135
ULStreamErrorContext enumeration	138
ULStreamErrorID enumeration	139
ULStreamType enumeration	140
ULSyncParms class	141
ULSyncResult class	145
ULSyncState enumeration	146
ULTable class	148
ULTableSchema class	157
Index	159

About This Manual

Subject	This manual describes UltraLite for MobileVB. With UltraLite for MobileVB you can develop and deploy database applications to handheld, mobile, or embedded devices, running Palm OS or Windows CE.
Audience	This manual is intended for AppForge MobileVB and AppForge Crossfire 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.

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.

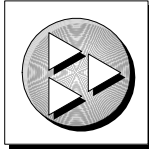
```
[ QUOTES { ON | OFF } ]
```

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

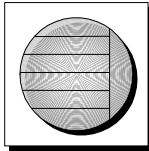
Graphic icons

The following icons are used in this documentation.

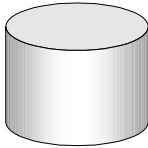
- ◆ A client application.



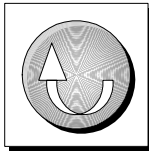
- ◆ A database server, such as Sybase Adaptive Server Anywhere.



- ◆ 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.



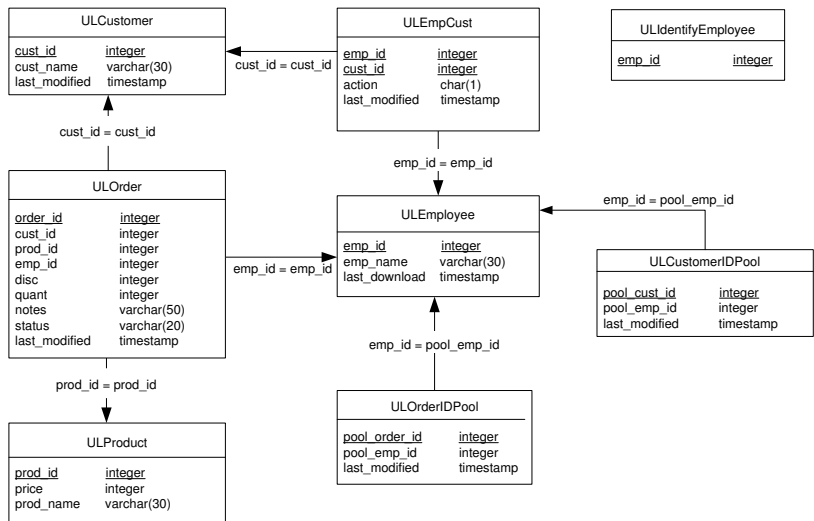
The CustDB sample database

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 MobileVB

About this chapter This chapter introduces UltraLite for MobileVB. 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 MobileVB features	2
	UltraLite for MobileVB architecture	3

UltraLite for MobileVB features

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

System requirements and supported platforms

Development platforms To develop applications using UltraLite for MobileVB, you require the following:

- ◆ Microsoft Visual Basic .NET or Visual Basic 6.

You must install a service pack that meets the requirements for the version of AppForge MobileVB or AppForge Crossfire that you are using. For more information, see [the AppForge web site](#). If you are using Visual Basic 6, it is recommended that you install at least service pack 5.

AppForge Booster

To deploy applications using UltraLite for MobileVB you need the AppForge Booster. If you are missing the AppForge Booster, you can get it from www.appforge.com/booster.html. BoosterPlus is not needed for UltraLite applications.

- ◆ AppForge MobileVB Version 3.x or later, or AppForge Crossfire.

Compatibility

If you are using versions of MobileVB earlier than 3.0 and are developing for Windows CE on an ARM device, you must copy `ultralite\UltraLiteForMobileVB\ce\arm\ulmvb9.dll` under your SQL Anywhere directory to the `\Program Files\AppForge` directory on your device.

☞ For more information, see “UltraLite development platforms” [[Introducing SQL Anywhere Studio, page 99](#)].

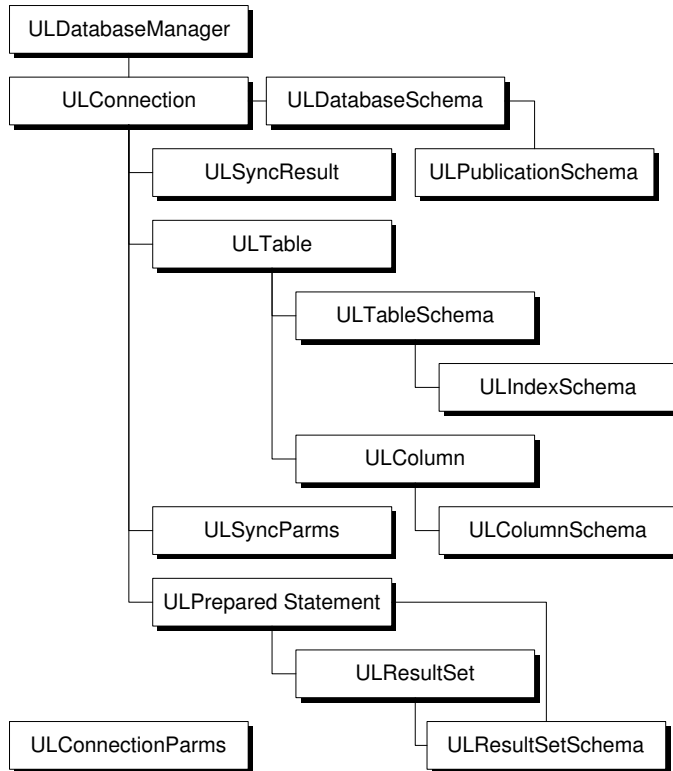
Target platforms UltraLite for MobileVB supports the following target platforms:

- ◆ Windows CE 3.0 and higher, with Pocket PC on the ARM and MIPS processors.
- ◆ Palm OS version 3.5 and higher.

☞ For more information, see “UltraLite target platforms” [[Introducing SQL Anywhere Studio, page 109](#)].

UltraLite for MobileVB 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.

- ◆ **ULDatabaseManager** manages connections to UltraLite databases.
 ➞ For more information, see [“ULDatabaseManager class” on page 105](#).
- ◆ **ULConnectionParms** holds a set of connection parameters.
 You can use a Connection Parameters control and specify connection parameters in a Visual Basic property sheet.
 ➞ For more information, see [“ULConnectionParms class” on page 102](#).
- ◆ **ULConnection** represents a database connection, and governs transactions.
 ➞ For more information, see [“ULConnection class” on page 91](#).

-
- ◆ **ULPreparedStatement, ULResultSet, and ULResultSetSchema** manage database requests and their results using SQL.
 - ☞ For more information, see [“ULPreparedStatement class” on page 116](#), [“ULResultSet class” on page 122](#), and [“ULResultSetSchema class” on page 128](#).
 - ◆ **ULTable and ULColumn** manage data using a table-based API.
 - ☞ For more information, see [“ULTable class” on page 148](#) and [“ULColumn class” on page 84](#).
 - ◆ **ULSyncParms and ULSyncResult** 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 MobileVB Development

About this chapter This chapter explains how to develop applications using UltraLite for MobileVB.

☞ For a hands-on tutorial, see [“Tutorial: A Sample UltraLite for MobileVB Application”](#) on page 41.

Contents	Topic:	page
	Preparing to use UltraLite for MobileVB	6
	Working with the database schema	9
	Connecting to an UltraLite database	11
	Encryption and obfuscation	14
	Working with data using dynamic SQL	15
	Working with data using the table API	21
	Accessing schema information	29
	Handling errors	30
	Authenticating users	31
	Synchronizing data	32
	Deploying UltraLite applications	35
	Maintaining state in UltraLite Palm applications	38

Preparing to use UltraLite for MobileVB

The following procedures describe the steps you must take before you can build an application using UltraLite for MobileVB.

Adding UltraLite to the MobileVB design environment

To access the UltraLite control from your MobileVB or Crossfire project, you must add UltraLite for MobileVB to the design environment.

❖ To add the UltraLite connection parameters control

1. From the Visual Basic menu, choose Project ► Components.
2. Click the Controls tab.
3. Scroll down the list to select UltraLite Connection Parameters 9.0. Click OK.

If this item does not appear in the list of available controls, complete the following steps:

- ◆ Close MobileVB and save your project.
- ◆ Open a command prompt at `ultralite\UltraLiteforMobileVB\win32` and run the following command:

```
ulmvbreg -r
```

- ◆ Restart MobileVB and open your project.
- ◆ Choose Project ► Components.
- ◆ Select UltraLite Connection Parameters 9.0.

A database icon is added to your toolbar. To add a ULConnectionParms object to your form you double-click this icon.

Adding a reference to UltraLite for MobileVB

Once SQL Anywhere Studio is installed, UltraLite for MobileVB is automatically added to any new MobileVB project. It is therefore not usually necessary to manually add a reference to UltraLite for MobileVB to a project. The following procedure is provided for occasional situations where you may need to add a reference manually, such as if you install MobileVB after installing SQL Anywhere Studio.

❖ **To add a reference to UltraLite for MobileVB**

1. From the Visual Basic menu, choose Project ► References.
2. If iAnywhere Solutions, UltraLite for MobileVB 9.0 is included in the list of available references, select it and click OK.

If iAnywhere Solutions, UltraLite for MobileVB 9.0 does not appear in the list of available references:

- ◆ Open a command prompt at *ultralite\UltraLiteforMobileVB\win32* and run the following command:

```
ulmbvreg -r
```

- ◆ Select iAnywhere Solutions, UltraLite for MobileVB 9.0 and click OK.

Adding UltraLite to the Crossfire design environment

Although the SQL Anywhere Studio Setup program automatically adds UltraLite to your Crossfire design environment, there are cases where you may have to add UltraLite to the environment manually. For example, if you install Crossfire after you install SQL Anywhere Studio, you may need to carry out this procedure.

To find out if you need to add UltraLite to Crossfire, check that a new Crossfire project includes a reference to *iAnywhere.UltraLiteForAppForge*. If it does not, you need to add UltraLite to the environment. Also, check if the *ULConnectionParms* class appears in the AppForge panel of the toolbox. If it does not, you need to add UltraLite to the environment.

❖ **To add UltraLite references and controls to your Crossfire project**

1. Register UltraLite for MobileVB with Crossfire.
 - a. Ensure that Crossfire is closed.
 - b. Open a command prompt at the *ultralite\UltraLiteforMobileVB\win32* subdirectory of your SQL Anywhere installation and run the following command:

```
ulmbvreg -r
```

- c. If you have upgraded a MobileVB project remove the reference to *UltraLiteAFLib* from the Visual Basic.NET Solution Explorer.
- d. Add a reference to *iAnywhere.UltraLiteForAppForge.dll*
 - i. From the Microsoft Development Environment menu, choose Project ► Add Reference and browse to the *ultralite\UltraLiteforMobileVB\win32* subdirectory of your SQL Anywhere installation.

-
- ii. Select *iAnywhere.UltraLiteForAppForge.dll* and click Open.
 - iii. Choose OK to add the reference.
 2. Add the ULConnectionParms control to the AppForge toolbox.
 - a. In the Microsoft Development Environment, right click the AppForge toolbox and choose Add/Remove Items. A dialog appears.
 - b. Click the COM Components tab.
 - c. Scroll down to the entry named ULConnectionParms Class. Check the box beside this component and click OK.
 - d. The ULConnectionParms control is added to the toolbox.

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.

For information on creating UltraLite database schema files, see

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 “ApplyFile method” on page 112 and “ApplyFileWithParms method” on page 112.

☞ For information about preparing a new schema file for deployment, see “Upgrading UltraLite database schemas” [*UltraLite Database User’s Guide*, page 54].

Example

The following code applies a new schema file.

```
Connection.Schema.ApplyFile("schema_file=\My Documents\  
myschema.usm")
```


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 ULConnection object

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

☞ For more information about the ULConnection object, see [“ULConnection class” on page 91](#).

- ◆ **Commit behavior** By default, UltraLite applications are in AutoCommit mode. Each insert, update, or delete statement is committed to the database immediately. Set ULConnection.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 [“Commit method” on page 93](#).

- ◆ **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 31](#).

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

☞ For more information, see [“Synchronizing data” on page 32](#).

- ◆ **Tables** UltraLite tables are accessed using the ULConnection.GetTable method.

☞ For more information, see [“GetTable method” on page 94](#).

Connecting to a database

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

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

☞ For more information about connecting to an UltraLite database using a ULConnectionParms object, see [“CreateDatabaseWithParms method” on page 107](#) and [“OpenConnectionWithParms method” on page 109](#).

❖ To connect to an UltraLite database using ULConnectionParms

1. Create a ULDatabaseManager 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 or as a class-level variable.

```
'MobileVB
Public DatabaseMgr As ULDatabaseManager
Set DatabaseMgr = New ULDatabaseManager

'Crossfire
Public DatabaseMgr As New UltraLiteAFLib.ULDatabaseManager
```

2. Declare a ULConnection object.

Most applications use a single connection to an UltraLite database, and keep the connection open all the time. For this reason, it is often best to declare the ULConnection object as global to the application.

```
'MobileVB
Public Connection As New ULConnection

'Crossfire
Public Connection As UltraLiteAFLib.ULDatabaseManager
```

3. Create a ULConnectionParms object.

Double-click the ULConnectionParms object on the MobileVB tool palette. A ULConnectionParms object appears on your form.

4. Set the required properties of the ULConnectionParms object.

In the ULConnectionParms properties window, specify properties such as the location of the database, the schema file, and a username and password for your database.

Using the following properties, you must specify a schema file for CreateDatabaseWithParms or a database file for OpenConnectionWithParms. For information about additional properties, see [“Properties” on page 102](#).

Keyword	Description
DatabaseOnCE	The path and filename of the UltraLite database on Windows CE.
DatabaseOnDesktop	The path and filename of the UltraLite database on the desktop machine
SchemaOnCE	The path and filename of the UltraLite schema on Windows CE.
SchemaOnDesktop	The path and filename of the UltraLite schema on the desktop machine.

5. Open a connection to the database.

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

The following code attempts to connect to an existing database. If the database does not exist, the OpenConnectionWithParms method returns an error. This causes CreateDatabaseWithParms to create a database using the specified schema file.

In Crossfire, be sure to include the GetOcx method on the ULConnectionParms object.

```
'MobileVB
On Error Resume Next
Set Connection = DatabaseMgr.OpenConnectionWithParms(
    LoginParms)
If Err.Number <> ULSQLCode.ulSQLE_NOERROR Then
    Set Connection = DatabaseManager.CreateDatabaseWithParms(
        LoginParms )
End If
'Crossfire
Try
    Connection = _
        DatabaseMgr.OpenConnectionWithParms( _
            ULConnectionParms1.GetOcx)
Catch
    If Err.Number = _
        UltraLiteAFLib.ULSQLCode.ulSQLE_ULTRALITE_DATABASE_NOT_
            FOUND _
    Then
        Err.Clear()
        Connection = _
            DatabaseMgr.CreateDatabaseWithParms( _
                ULConnectionParms1.GetOcx)
End Try
```

Encryption and obfuscation

You can encrypt or obfuscate your UltraLite database using UltraLite for MobileVB.

Encryption

To create a database with encryption, set the `ULConnectionParms.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 “[ChangeEncryptionKey method](#)” on page 92.

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.

```
Connection.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` as a creation parameter.

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

Example

The following code obfuscates a new database.

```
open_parms = "ce_file=\tutorial.udb;ce_schema=\  
tutorial.usm;obfuscate=1"  
Set Connection = DatabaseManager.CreateDatabase( open_parms )
```

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 21](#).

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

- ◆ Scrolling through the rows of a table.
- ◆ Accessing the values of the current row.
- ◆ Locating rows in a table.
- ◆ Inserting, deleting, and updating rows.

☞ 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 ULPreparedStatement class.

☞ For more information the ULPreparedStatement class, see [“ULPreparedStatement class” on page 116](#).

Using parameters in your prepared statements

Placeholders for parameters are supplied using the ? character. 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 `ULPreparedStatement` object.

```
'MobileVB
Dim PrepStmt As ULPreparedStatement

'Crossfire
Dimr PrepStmt As UltraLiteAFLib.ULPreparedStatement
```

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

```
'MobileVB
Set PrepStmt = Connection.PrepareStatement( _
    "INSERT INTO TableName(ColumnName) VALUES ( ? )" )

'Crossfire
PrepStmt = Connection.PrepareStatement( _
    "INSERT INTO TableName(ColumnName) VALUES( ? )" )
```

3. Assign parameter values for the statement.

```
Dim NewValue As String
NewValue = "Bob"
PrepStmt.SetStringParameter 1, NewValue
```

4. Execute the statement.

```
PrepStmt.ExecuteStatement
```

❖ To UPDATE a row

1. Declare a `ULPreparedStatement` object.

```
Dim PrepStmt As ULPreparedStatement
```

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

```
Set PrepStmt = Connection.PrepareStatement( _
    "UPDATE TableName SET ColumnName = ? WHERE ID = ?" )
```

3. Assign parameter values for the statement.

```
Dim NewValue As String
NewValue = "Bob"
PrepStmt.SetParameter 1, NewValue
PrepStmt.SetParameter 2, "6"
```

4. Execute the statement

```
PrepStmt.ExecuteStatement
```

❖ To DELETE a row

1. Declare a ULPreparedStatement object.

```
'MobileVB  
Dim PrepStmt As ULPreparedStatement  
  
'Crossfire  
Dim PrepStmt As UltraLiteAFLib.ULPreparedStatement
```

2. Assign a DELETE statement to your prepared statement object.

```
'MobileVB  
Set PrepStmt = Connection.PrepareStatement( _  
    "DELETE FROM customer WHERE ID = ?")  
  
'Crossfire  
PrepStmt = Connection.PrepareStatement( _  
    "DELETE FROM customer WHERE ID = ?")
```

3. Assign parameter values for the statement.

```
Dim IDValue As String  
IDValue = "6"  
PrepStmt.SetParameter 1, IDValue
```

4. Execute the statement.

```
PrepStmt.ExecuteStatement
```

Data retrieval: SELECT

When you execute a SELECT statement, the `ULPreparedStatement.ExecuteQuery` method returns a `ULResultSet` object.

The `ULResultSet` class contains methods for navigating within a result set. The values are then accessed using methods of the `ULResultSet` class.

☞ For more information about `ULResultSet` objects, see [“ULResultSet class” on page 122](#).

Example

In the following code, the results of a SELECT query are accessed through a `ULResultSet`. When first assigned, the `ULResultSet` is positioned before the first row. The `ULResultSet.MoveFirst` method is then called to navigate to the first record in the result set.

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

```

'MobileVB
Dim MyResultSet As ULResultSet
Dim PrepStmt As ULPreparedStatement
PrepStmt = Connection.PrepareStatement( _
    "SELECT ID, Name FROM customer")
MyResultSet = PrepStmt.ExecuteQuery
MyResultSet.MoveFirst

'Crossfire
Dim MyResultSet As UltraLiteAFLib.ULResultSet
Dim PrepStmt As UltraLiteAFLib.ULPreparedStatement
PrepStmt = Connection.PrepareStatement( _
    "SELECT ID, Name FROM customer")
MyResultSet = PrepStmt.ExecuteQuery
MyResultSet.MoveFirst

```

UltraLite for MobileVB provides you with methods to get data of particular types from the UltraLite database into a result set. MobileVB does not support the use of Variant data types and, because of this, UltraLite for MobileVB includes functions to handle all types of data. Each of these methods is called using the following template, where *Index* is the ordinal position of the column name in your SELECT statement:

```
MyResultSetName.MethodName( Index )
```

Example

The following code demonstrates how to use the GetString method to obtain the column values for the current row.

The GetString method uses the following syntax, where *Index* is the ordinal position of the column name in your SELECT statement.

```
MyResultSetName.GetString( Index )
```

The MoveRelative(0) method is called to refresh the contents of the current buffer from the result set, so that the effects of any data modification are included.

```

If MyResultSet.RowCount = 0 Then
    lblID.Caption = ""
    txtName.Text = ""
Else
    lblID.Caption = MyResultSet.GetString(1)
    txtName.Text = MyResultSet.GetString(2)
    MyResultSet.MoveRelative(0)
End If

```

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

❖ **To perform a SELECT statement**

1. Declare a ULPreparedStatement object.

```
'MobileVB
Dim PrepStmt As ULPreparedStatement
'Crossfire
Dim PrepStmt As UltraLiteAFLib.ULPreparedStatement
```

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

```
Set PrepStmt = Connection.PrepareStatement( _
    "SELECT ColumnName FROM TableName")
```

3. Execute the query.

In the code below, an AFListBox captures the result of the SELECT query.

```
Dim MyResultSet As ULResultSet
Set MyResultSet = PrepStmt.ExecuteQuery
While MyResultSet.MoveNext
    aflightbox.AddItem MyResultSet.GetString(1)
Wend
```

Navigation with dynamic SQL

UltraLite for MobileVB 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 ULResultSet 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 demonstrates how to use the `MoveFirst` method to navigate within a result set.

```
'MobileVB
Set PrepStmt = Connection.PrepareStatement( _
    "SELECT ID, Name FROM customer")
Set MyResultSet = PrepStmt.ExecuteQuery
MyResultSet.MoveFirst

'Crossfire
PrepStmt = Connection.PrepareStatement( _
    "SELECT ID, Name FROM customer")
MyResultSet = PrepStmt.ExecuteQuery
MyResultSet.MoveFirst
```

The same technique is used for all of the `Move` methods.

☞ For more information about these navigational methods, see [“ULResultSet class” on page 122](#).

ULResultSet schema property

The `ULResultSet.Schema` property allows you to retrieve information about the columns in the query. The properties of this `ULResultSetSchema` object include `ColumnName`, `ColumnCount`, `ColumnPrecision`, `ColumnScale`, `ColumnSize`, and `ColumnSQLType`.

Example

The following example shows how you can use `ULResultSet.Schema` to display schema information in a MobileVB grid. The example assumes you have a `ULResultSet` named `MyResultSet` and a MobileVB grid named `grdSchema`.

```
Dim i As Integer
For i = 1 To MyResultSet.Schema.ColumnCount
    grdSchema.AddItem (MyResultSet.Schema.ColumnName(i) _
        & Chr(9) & MyResultSet.Schema.ColumnSQLType(i)), 0
Next i
grdSchema.AddItem _
    ("Column Name" & Chr(9) & "Column Type"), 0
```

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 15](#).

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 MobileVB 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 a message box with the last name of each customer.

```

'MobileVB
Dim TCustomer as ULTable
Set TCustomer = Conn.GetTable("customer")
TCustomer.Open
While TCustomer.MoveNext
    MsgBox TCustomer.Column( "lname" ).StringValue
Wend

'Crossfire
Dim TCustomer as UltraLiteAFLib.ULTable
Set TCustomer = Conn.GetTable("Customer")
TCustomer.Open
While TCustomer.MoveNext
    MsgBox TCustomer.Column("LName").StringValue
Wend

```

Example

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.

```

'MobileVB
Set TCustomer = Conn.GetTable("customer")
TCustomer.Open "ix_name"
TCustomer.MoveFirst

'Crossfire
TCustomer = Conn.GetTable("customer")
TCustomer.Open "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 the Column method together with an appropriate property to get the value of that column for the current row.

Example

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

```
Dim colID, colFirstName, colLastName As ULColumn
Set colID = tCustomer.Column("ID")
Set colFirstName = tCustomer.Column("fname")
Set colLastName = tCustomer.Column("lname")
txtID.Text = colID.IntegerValue
txtFirstName.Text = colFirstName.StringValue
txtLastName.Text = colLastName.StringValue
```

You can also use the properties of ULColumn to set values.

```
colLastName.StringValue = "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 generates an error.

```
' This code is incorrect
TCustomer.MoveBeforeFirst
id = TCustomer.Column("ID").IntegerValue
```

To work with binary data, use the GetByteChunk method instead of a property.

☞ For more information, see [“GetByteChunk method” on page 86](#).

Casting values

The ULColumn property you choose must match the Visual Basic data type you wish to assign. UltraLite automatically casts incompatible data types, so that you could use the StringValue method to fetch an integer value into a string variable, and so on.

☞ For more information about accessing values of the current row, see [“ULColumn class” on page 84](#).

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 [“FindBegin method” on page 149](#).

- ◆ **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 [“LookupBackward method” on page 152](#).

❖ To search for a row

1. Enter find or lookup mode.

Call the FindBegin or LookupBegin method. For example, the following code 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 sets the last name column in the buffer to Kaminski.

```
tCustomer.Column("lname").StringValue = "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 ID column in the buffer to 3.

```
colID.IntegerValue = 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.

```
ColFirstName.StringValue = "Elizabeth"
```

4. Execute the Update.

```
tCustomer.Update
```

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

By default, UltraLite operates in `AutoCommit` mode, so that the update is immediately applied to the row in permanent storage. If you have disabled

AutoCommit mode, the update is not applied until you execute a commit operation. For more information, see [“Transaction processing in UltraLite” on page 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.

```
CustomerTable.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.

```
CustomerTable.Column("FName").StringValue = fname  
CustomerTable.Column("LName").StringValue = lname
```

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.

```
CustomerTable.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 deletion.

```
tCustomer.Delete
```

Working with BLOB data

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

☞ For more information, see [“GetByteChunk method” on page 86](#).

Example

The following code demonstrates how to use the ULColumn.GetByteChunk method to get BLOB data.

```
'MobileVB
Dim table as ULTable
Dim col As ULColumn
Dim data(1 to 1024) As Byte
Dim data_fit As Boolean
Dim size As Long
Set table = Conn.GetTable("image")
table.Open
size = 1024
Set col = table.Column("img_data")
data_fit = col.GetByteChunk(VarPtr(data(1)), size)
If data_fit Then
    'No truncation
Else
    'data truncated at 1024
End if
table.Close

'Crossfire
Dim table as ULTable
Dim col As ULColumn
Dim data(1 to 1024) As Byte
Dim data_fit As Boolean
Dim size As Long
Set table = Conn.GetTable("image")
table.Open
size = 1024
Set col = table.Column("img_data")
' The data argument must be a local variable
data_fit = col.GetByteChunk(data, size)
If data_fit Then
    'No truncation
Else
    'data truncated at 1024
End if
table.Close
```

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 `ULConnection.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 `ULConnection.Commit` statement to complete a transaction and make changes to your database permanent, or you may execute a `ULConnection.Rollback` statement to cancel all the operations of a transaction. Turning off `AutoCommit` improves performance.

Note

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

Accessing schema information

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

Note

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

☞ For information about modifying the schema, see [“Changing the schema of a database” on page 9](#).

- ◆ **ULDatabaseSchema** 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 `ULDatabaseSchema` object, access the `ULConnection.Schema` property.
- ◆ **ULTableSchema** The number and names of columns in the table, as well as the Indexes collections for the table.
To obtain a `ULTableSchema` object, access the `ULTable.Schema` property.
- ◆ **ULColumnSchema** Information about an individual column, including its default value, name, and whether it is autoincrement.
To obtain a `ULColumnSchema` object, access the `ULColumn.Schema` property.
- ◆ **ULIndexSchema** Information about the column, or columns, in the index. As an index has no data directly associated with it, there is no separate `ULIndex` object, only a `ULIndexSchema` object.
The `ULIndexSchema` objects are accessed using the `ULTableSchema.GetIndex` method.
- ◆ **ULPublicationSchema** The numbers and names of tables and columns contained in a publication. Publications are also comprised of schema only, so there is a `ULPublicationSchema` object but no `ULPublication` object.
The `ULPublicationSchema` objects are accessible using the `ULDatabaseSchema.GetPublicationSchema` method.
- ◆ **ULResultSetSchema** The number and names of the columns in a result set.
The `ULResultSetSchema` objects are accessible using the `ULPreparedStatement.ResultSetSchema` property.

Handling errors

In normal operation, UltraLite for MobileVB can throw errors. Errors are expressed as `SQLCODE` values, negative numbers indicating the particular kind of error.

☞ For a list of error codes thrown by UltraLite for MobileVB, see [“ULSQLCode enumeration” on page 130](#).

UltraLite for MobileVB throws errors only from the `ULDatabaseManager` and `ULConnection` objects. The following methods of `ULDatabaseManager` can throw errors.

- ◆ `CreateDatabase`
- ◆ `CreateDatabaseWithParms`
- ◆ `DropDatabase`
- ◆ `DropDatabaseWithParms`
- ◆ `OpenConnection`
- ◆ `OpenConnectionWithParms`

All other errors and exceptions within UltraLite for MobileVB are routed through the `ULConnection` object.

☞ For more information about accessing error numbers from `ULDatabaseManager` and `ULConnection` objects, see [“ULConnection class” on page 91](#) and [“ULDatabaseManager class” on page 105](#).

You can use the standard MobileVB or Crossfire error-handling features to handle errors. When an UltraLite object is the source of an error, the `Err` object is assigned a `ULSQLCode` number. `ULSQLCodes` are negative numbers indicating the particular kind of error. The `ULSQLCode` enumeration provides a set of descriptive constants associated with these values.

☞ For more information, see [“ULSQLCode enumeration” on page 130](#).

To make use of type completion in the MobileVB environment, you may want to create an error handling function such as the following:

```
'MobileVB
Public Function GetError() As ULSQLCode
    GetError = Err.Number
End Function
```

You can then easily access UltraLite errors using the `GetError` function.

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 [“GrantConnectTo method” on page 94](#) and [“RevokeConnectFrom method” on page 98](#).

❖ 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

You can UltraLite applications with a central database. Synchronization requires the MobiLink synchronization server and appropriate licensing.

This section provides a brief introduction to synchronization and describes some features of particular interest to users of UltraLite for MobileVB. For a detailed explanation of synchronization, see “UltraLite Clients” [*MobiLink Clients*, page 277].

You can also find a working example of synchronization in the CustDB sample application. This sample is described in “Tutorial: A Sample UltraLite for MobileVB Application” on page 41

UltraLite for MobileVB supports TCP/IP, HTTP, and HTTPS synchronization. Synchronization is initiated by the UltraLite application. In all cases, you use methods and properties of the ULConnection 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 ULConnection.SyncParms object.

☞ For information about the properties and the values that you should set, see “UltraLite Clients” [*MobiLink Clients*, page 277].

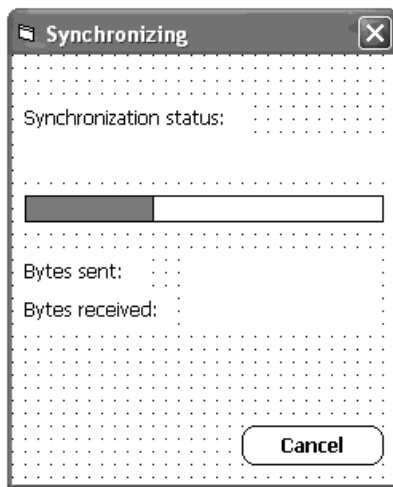
2. Synchronize.

Call the ULConnection.Synchronize method.

Adding the synchronization template

UltraLite for MobileVB includes a template form that can be used to monitor the status of a synchronization session. A version of this form is included for both Palm OS and Pocket PC. You can use these templates in

your application, you can customize them, or you can simply examine them to learn how UltraLite synchronization events work.



The way to add this template to your application depends on whether you are using MobileVB or Crossfire.

❖ **To add a synchronization template to your application (MobileVB)**

1. From the project menu, choose Add Form.
2. Select either UltraLite for MobileVB Sync Form (Windows CE) or UltraLite for MobileVB Sync Form (Palm).
3. Click Open. A copy of the form is added to your application.

❖ **To add a synchronization template to your application (Crossfire)**

1. From the project menu, choose Add New Item.
2. From Local Project Items ► Ultralite_Crossfire Forms, select UltraLite Crossfire 9 Sync Form for CE, Palm or PalmHires depending on your application.
3. Click Open. A copy of the form is added to your application.

Writing code to use the synchronization form

Call the `InitSyncForm` function, passing it your `ULConnection` object. This must be done before each synchronization.

Example

The following code uses a synchronization status form named `Form_Sync` and a `ULConnection` object named `Connection`.

```
Form_Sync.InitSyncForm Connection  
Connection.Synchronize
```

Each time your application synchronizes, the synchronization status form appears. As synchronization progresses, your end user can observe the progress bar and byte count. When synchronization completes, the form closes. The Cancel button instructs UltraLite to cancel the current synchronization.

☞ For more details, see [“Tutorial: A Sample UltraLite for MobileVB Application”](#) on page 41.

Deploying UltraLite 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 UltraLite for MobileVB applications to Windows CE

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

- ◆ Deploy your application and UltraLite component.
 1. Configure the application settings.
 - From the MobileVB menu, choose MobileVB Settings. A dialog appears.
 - In the left pane, choose Dependencies and click the User Dependencies tab.
 - Click the Add button and select the `c:\tutorial\mvp\tutCustomer.usm`. This indicates to MobileVB that the file should be included in the deployment.
 - Choose the PocketPC Settings item in the left pane
 - Enter `\tutorial\mvp` for the Device Installation Path.
 - Click OK to close the dialog.
 2. From the MobileVB menu, choose Deploy to Device, and select the PocketPC device. If a dialog appears asking if you want to save the project, choose Yes.
 3. If you are running MobileVB 3.0 or later, the UltraLite component is deployed automatically along with your application.

If you are using an older version of MobileVB, you need to copy the UltraLite component to the device. Copy the file `ulmvp9.dll` to `\Program Files\AppForge` on your device. The file is located in one of the following platform-specific subdirectories of your SQL Anywhere 9 installation: `\UltraLite\UltraLiteForMobileVB\ce\arm` or `\UltraLite\UltraLiteForMobileVB\ce\mips`. This step only needs to be performed once per device.
- ◆ Deploy an initial copy of the UltraLite database or schema.

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 deploy the `.prc` file in the standard manner from the Install Tool included with your Palm Desktop Organizer Software.

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.

☞ See “Database On CE connection parameter” [*UltraLite Database User’s Guide*, page 69], and “Schema On CE connection parameter ” [*UltraLite Database User’s Guide*, page 78].

- ◆ Deploy the MobiLink provider for ActiveSync.

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

☞ For instructions, see “Installing the MobiLink provider for ActiveSync” [*MobiLink Clients*, page 310]

Deploying UltraLite for MobileVB 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.
 1. Configure the application settings.
 - From the MobileVB menu, choose MobileVB Settings. A dialog appears.
 - In the left pane, choose Dependencies and click the User Dependencies tab.
 - Click the Add button and select the *c:\tutorial\mvb\tutCustomer.pdb*. This indicates to MobileVB that the file should be included in the deployment.
 - Choose the Palm OS Settings item in the left pane and enter the Creator ID of your application. Select a valid HotSync name. Click OK to close the dialog.
 2. From the MobileVB menu, choose Deploy to Device, and select the Palm OS device. If a dialog appears asking if you want to save the project, choose Yes.
 3. If you are running MobileVB 3.0 or later, the UltraLite component is deployed automatically along with your application.

If you are using an older version of MobileVB, you need to copy the UltraLite component to the device. Copy the file *ulmbv9.prc* to *\Program Files\AppForge* on your device. The file is located in the *\UltraLite\UltraLiteForMobileVB\palm\68k* subdirectory of your SQL Anywhere installation. This step only needs to be performed once per device.

- ◆ Deploy an initial copy of the UltraLite database or schema.

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 *.prc* 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 and Schema On Palm connection parameters use the creator ID to find the schema or 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].

Maintaining state in UltraLite Palm applications

Palm OS devices run only one application at a time. However, when a user switches from your application to another application, and then returns to your application, it is common to make applications appear as they were simply suspended while the user was working with other applications. To maintain this illusion, the application must save its internal state when the user switches to another application. When the application is launched again, it must restore its internal state.

Saving and restoring state in a database application can be challenging, as the application must re-open result sets and re-position the application within those result sets. UltraLite provides a way for you to save and restore application state.

The state of cursors on result sets is maintained automatically. MobileVB applications that use the table-based API provide a value for the persistent name parameter in the Open method of the ULTable object.

Understanding how state is maintained

For each table whose state is being preserved, UltraLite stores a name for the table as well as enough information to restore the state of the table. The name associated with the table may be, but is not required to be, the name of the table. It is called the **persistent name**.

UltraLite applications can open more than one instance of the same table at the same time. In this case, the table name is not unique. For example, the following code (using MobileVB) opens the same table twice:

```
' MobileVB
Set table1 = Connection.GetTable( "ULCustomer" )
table1.Open "", "customer1"
' operations here
Set table2 = Connection.GetTable( "ULCustomer" )
table2.Open "", "customer2"
```

When opening a table, an applications can optionally provide a persistent name as a parameter. If the state of the persistent name has been saved, the table is opened and positioned to the proper row. If not, the table is opened and positioned before the first row.

When an application terminates, it may or may not explicitly close open tables and close the connection. If it does not close an open table, then UltraLite records the current row of each open table that was supplied with a persistent name. Tables without a persistent name are closed.

Suppose the Connection object is of type ULConnection and a table called

ULCustomer exists in the database.

```
'MobileVB
Dim table As ULTable
Set table = Connection.GetTable( "ULCustomer" )
table.Open "", "customer"
```

The second line of code gets the table object representing the ULCustomer table. The table has not been opened for reading or writing yet.

In the Open call (the third line of code), the first parameter is the empty string, which indicates that the data will be ordered by the primary key. The second parameter is the persistent name being assigned to the table. If the application terminates while this table is still open, the state PDB will associate *customer* with the ULCustomer table and save the current position.

Persistent name notes

- ◆ If the persistent name is empty, UltraLite does not store state information for this table, or attempt to look it up when opening the table.
If you do not need to store the state of your tables, supply an empty persistent name. The state is then not looked up in the state database.
- ◆ HotSync synchronization does not affect the state of your open tables. When a user presses the HotSync button on a device, the operating system closes the application in the same way it closes the application when any other application is started. Consequently, the state of the open tables is recorded in the state PDB and when the user returns to the application and the tables are re-opened, the user is positioned on the expected row. If that row has been deleted as part of the synchronization, the user is positioned on the next row (or after the last row if it was the last row).
- ◆ Applications with auto-commit turned off could terminate with uncommitted transactions. UltraLite maintains these transactions so that when the application restarts, they are not lost.
- ◆ If UltraLite finds a table in the state PDB that matches the persistent name you have provided, it checks that the table and index are the same as the table and index used when the position information was recorded. If they are not, then the call to Open fails.
- ◆ The use of the persistent name is unique to the Palm OS. If you create UltraLite for MobileVB applications for Windows CE, they do not use the persistent name. Applications on Windows CE run more like they do on a desktop machine.

Example: Using the persistent name to maintain state information

The PersistentName example program is a relatively simple program that shows how to use maintained state information. It is available at

<http://www.sybase.com/detail?id=1022734>. Here are some highlights from the sample:

```
'MobileVB
CustomerTable.Open
AddRow "John", "Doe", "Atlanta"
AddRow "Mary", "Smith", "Toronto"
AddRow "Jane", "Anderson", "New York"
AddRow "Margaret", "Billington", "Vancouver"
AddRow "Fred", "Jones", "London"
AddRow "Jack", "Frost", "Dublin"
AddRow "David", "Reiser", "Berlin"
AddRow "Kathy", "Stevens", "Waterloo"
AddRow "Rebecca", "Gable", "Paris"
AddRow "George", "Jenkins", "Madrid"
CustomerTable.Close
```

This code adds ten rows to the ULCustomer table. It calls Open on the table, but in this case a persistent name is not supplied because we are not interested in maintaining the position in the table. Since the code only inserts data, the position in the table is not relevant.

The following line opens the ULCustomer table, ordering rows by the primary key and assigning a persistent name of customer.

```
CustomerTable.Open " " , "customer"
```

If the application has been run before, then a lookup is done in the state database for customer. Otherwise, customer is associated with this table.

The customer table is left open for the duration of the running application. If the user switches to another application, UltraLite records the position in the table where the user left off. When the application is started again, the table is opened and UltraLite determines that position information is known for a table with the persistent name customer, so it positions the user back on that row.

When the user clicks the End button, the customer table and the connection are closed before the application disappears. This has the effect of discarding any state information for the customer table, so when the application is restarted, the user is positioned on the first row.

CHAPTER 3

Tutorial: A Sample UltraLite for MobileVB Application

About this chapter

This chapter provides a tutorial to guide you through the process of building an UltraLite for MobileVB application for either a PocketPC or a Palm OS device.

Contents

Topic:	page
Introduction	42
Lesson 1: Create a project architecture	43
Lesson 2: Create a form	46
Lesson 3: Write the sample code	48
Lesson 4: Deploy to a device	57
Summary	59

Introduction

This tutorial guides you through the process of building an UltraLite for MobileVB application using the table API. At the end of the tutorial you will have an application and small database on your Windows CE device that synchronizes with a central database.

☞ For more information about the table API, see the “[UltraLite for MobileVB API Reference](#)” on page 81.

Timing

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

Competencies and experience

This tutorial assumes:

- ◆ you have MobileVB and Microsoft Visual Basic 6 installed on your computer
- ◆ you can write, test, and troubleshoot a MobileVB application
- ◆ 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].

- ◆ you have the AppForge Booster installed
If you are missing Booster, you can get it from <http://www.appforge.com/booster.html>.

Note

You can perform most of this tutorial without SQL Anywhere Studio. The synchronization sections of the tutorial require SQL Anywhere Studio.

Goals

The goals for the tutorial are to gain competence and familiarity with the process of developing an UltraLite application.

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 9.

❖ To create an UltraLite database schema

1. Create a directory for this tutorial.

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

2. 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

◆ **Primary key** ascending id

3. If you are using a Palm device, export the schema to Palm with a creator id of *Syb3*.
 - 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 your tutorial directory.

A note on Palm Creator ID's

A Palm 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.

Create a MobileVB project

The following procedure creates a MobileVB project for your application and adds a reference to the UltraLite for MobileVB control.

MobileVB tools appear in addition to the standard Visual Basic tools on the Visual Basic toolbar to the left of the Visual Basic environment.

❖ To add the UltraLite connection parameters control

1. Start MobileVB.
 - ◆ Choose Start ► Programs ► AppForge MobileVB ► Start MobileVB. The MobileVB Project Manager appears.
2. From the Visual Basic menu, choose Project ► Components.
3. Click the Controls tab.
4. Scroll down the list to select UltraLite Connection Parameters 9.0. Click OK.

If this item does not appear in the list of available controls, complete the following steps:

- ◆ Close MobileVB and save your project.
- ◆ Open a command prompt at *ultralite\UltraLiteforMobileVB\win32* and run the following command:

```
ulmvbreg -r
```

- ◆ Restart MobileVB and open your project.
- ◆ Choose Project ► Components.
- ◆ Select UltraLite Connection Parameters 9.0.

A database icon is added to your toolbar. Double-click this icon to add a ULConnectionParms object to your form.

❖ **To create a reference to the UltraLite for MobileVB control**

1. Start MobileVB.

- ◆ Choose Start ► Programs ► AppForge MobileVB ► Start MobileVB.
The MobileVB Project Manager appears.

2. Create a new project.

Click New Project. When asked for the Design Target, choose the appropriate target. This tutorial provides instructions for Palm OS and PocketPC devices.

3. Create a reference to UltraLite for MobileVB.

- ◆ Choose Project ► References
- ◆ Select iAnywhere Solutions, UltraLite for MobileVB 9.0 and click OK.
If the control does not appear in the list of available references, exit MobileVB and run the following command.

```
ulmvbreg -r
```

4. Give the Project a name.

- ◆ Click Project ► MobileVBProject1 Properties
- ◆ Under Project Name, type **UltraLiteTutorial**. This is the name of the application as it will appear on your device.
- ◆ Click OK.

5. Save the Project:

- ◆ Choose File ► Save Project.
- ◆ Save the form as `c:\tutorial\mvp\Tutorial.frm`.
- ◆ Save the project as `c:\tutorial\mvp\Tutorial.vbp`.

Lesson 2: Create a form

After completing the steps in “[Lesson 1: Create a project architecture](#)” on [page 43](#), the project should display a form. The following procedure uses the form to create a user interface. This example uses labels as outputs, and text boxes and buttons as inputs, similar to a real application.

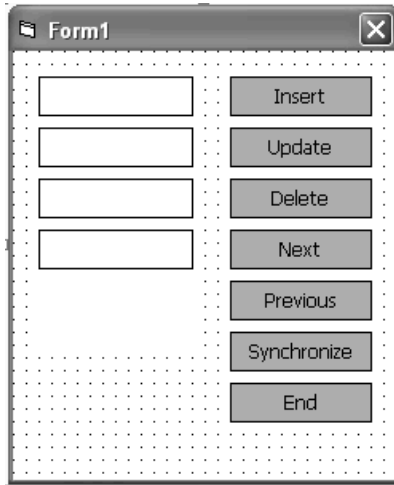
❖ To add controls to your project

1. Add the controls and properties given in the table below to your form:

Type	Name	Caption or text
AFTextBox	txtfname	
AFTextBox	txtlname	
AFTextBox	txtcity	
AFTextBox	txtphone	
AFLabel	lblID	
AFButton	btnInsert	Insert
AFButton	btnUpdate	Update
AFButton	btnDelete	Delete
AFButton	btnNext	Next
AFButton	btnPrevious	Previous
AFButton	btnSync	Synchronize
AFButton	btnDone	End

2. Check the application.
 - ◆ Choose MobileVB ► Compile and Validate.

Your form should look like the following screen capture:



Lesson 3: Write the sample code

This lesson guides you through the process of writing Visual Basic code to connect to a database, navigate within the database, and manipulate the data in the database.

This lesson also includes instructions for synchronizing your application with an Adaptive Server Anywhere database. This portion of the lesson is optional, and requires SQL Anywhere Studio.

Write code to connect to your database

In this application, you connect to the database during the `Form_Load` event. You can also connect to a database using the general module.

This example uses a `ULConnectionParms` object to connect to a database. You can also use a connection string.

☞ For reference information, see [“ULConnectionParms class” on page 102](#).

❖ Write code to connect to the UltraLite database

1. Double-click the form to open the Code window.
2. Declare the required UltraLite objects.

Enter the following code in the declarations area of your form.

```
Public DatabaseMgr As New ULDatabaseManager
Public Connection As ULConnection
Public CustomerTable As ULTable
```

3. Specify the connection parameters.
 - ◆ Add a `ULConnectionParms` object to your form named `ULConnectionParms1`. The `ULConnectionParms` control is a database icon on the toolbox.
 - ◆ In the Properties window, specify the `ULConnectionParms` properties. If you are deploying to a Windows CE device, specify the following properties:

Property	Value
DatabaseOnDesktop	c:\tutorial\mvb\tutCustomer.udb
DatabaseOnCE	\tutorial\mvb\tutCustomer.udb
SchemaOnDesktop	c:\tutorial\mvb\tutCustomer.usm
SchemaOnCE	\tutorial\mvb\tutCustomer.usm

If you are deploying to a Palm device, specify the following properties:

Property	Value
DatabaseOnDesktop	c:\tutorial\mvb\tutCustomer.pdb
DatabaseOnPalm	Syb3
SchemaOnDesktop	c:\tutorial\mvb\tutCustomer.usm
SchemaOnPalm	tutCustomer

4. Add code to connect to the database in the Form_Load event.

The database manager tries to open a connection to the database specified by the ULConnectionParms1 object. If the database does not exist, it creates a new database using the given schema.

```

Sub Form_Load()
    ' enable error handling
    On Error Resume Next
    ' try to connect to an existing database
    Set Connection = _
        DatabaseMgr.OpenConnectionWithParms( _
            ULConnectionParms1)
    ' if the database does not exist, create one
    If Err.Number = ULSQLCode.ulSQLE_NOERROR Then
        MsgBox "Connected to an existing database"
    ElseIf Err.Number = _
        ULSQLCode.ulSQLE_ULTRALITE_DATABASE_NOT_FOUND _
    Then
        Err.Clear
        Set Connection = _
            DatabaseMgr.CreateDatabaseWithParms( _
                ULConnectionParms1)
        If Err.Number = ULSQLCode.ulSQLE_NOERROR _
        Then
            MsgBox "Connected to a new database"
        Else
            MsgBox Err.Description
        End If
    Else
        MsgBox Err.Description
    End If
End Sub

```

5. Add code to end the application and close the connection when the End button is clicked.

```

Sub btnDone_Click()
    Connection.Close
End
End Sub

```

6. Run the application.

-
- ◆ Choose Run ► Execute.
 - ◆ After an initial message box, the form loads.
 - ◆ To terminate the application, click End.

Write code for navigation and data manipulation

The following procedures implement data manipulation and navigation.

❖ To open the table

1. Write code to initialize the table and move to the first row.

This code assigns the customer table in the database to the CustomerTable variable. The call to Open opens the table so that the table data can be read or manipulated. It also positions the application before the first row in the table.

Add the following code to the Form_Load event, just before the End Sub instruction.

```
Set CustomerTable = Connection.GetTable("customer")
CustomerTable.Open
If Err.Number <> ULSQLCode.ulSQLE_NOERROR Then
    MsgBox Err.Description
End If
```

2. Create a new procedure called DisplayCurrentRow and implement it as shown below.

If the table has no rows, the following procedure causes the application to display empty controls. Otherwise, it displays the values stored in each of the columns of the current row of the database.


```
Private Sub DisplayCurrentRow()  
    If CustomerTable.RowCount = 0 Then  
        txtFname.Text = ""  
        txtLname.Text = ""  
        txtCity.Text = ""  
        txtPhone.Text = ""  
        lblID.Caption = ""  
    Else  
        lblID.Caption = _  
            CustomerTable.Column("Id").StringValue  
        txtFname.Text = _  
            CustomerTable.Column("Fname").StringValue  
        txtLname.Text = _  
            CustomerTable.Column("Lname").StringValue  
        If CustomerTable.Column("City").IsNull Then  
            txtCity.text = ""  
        Else  
            txtCity.Text = _  
                CustomerTable.Column("City").StringValue  
        End If  
        If CustomerTable.Column("Phone").IsNull Then  
            txtphone.Text = ""  
        Else  
            txtphone.Text = _  
                CustomerTable.Column("Phone").StringValue  
        End If  
    End If  
End Sub
```

3. Call DisplayCurrentRow from the Form_Activate procedure. This call ensures that the fields get updated when the application starts.

```
Private Sub Form_Activate()  
    DisplayCurrentRow  
End Sub
```

❖ To insert rows into the table

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 the values in the row to their defaults. For example, the ID column receives the next autoincrement value. The column values are set and then the new row is inserted.

Add the following procedure to the form.

```
Private Sub btnInsert_Click()  
    Dim fname As String  
    Dim lname As String  
    Dim city As String  
    Dim phone As String  
  
    fname = txtFname.Text  
    lname = txtLname.Text  
    city = txtCity.Text  
    phone = txtPhone.Text  
  
    On Error GoTo InsertError  
    CustomerTable.InsertBegin  
    CustomerTable.Column("Fname").StringValue = _  
        fname  
    CustomerTable.Column("Lname").StringValue = _  
        lname  
    If Len(city) > 0 Then  
        CustomerTable.Column("City").StringValue = _  
            city  
    End If  
    If Len(phone) > 0 Then  
        CustomerTable.Column("Phone").StringValue = _  
            phone  
    End If  
    CustomerTable.Insert  
    CustomerTable.MoveLast  
    DisplayCurrentRow  
    Exit Sub  
  
InsertError:  
    MsgBox "Error: " & CStr(Err.Description)  
End Sub
```

2. Run the application.

After an initial message box, the form is displayed.

3. Insert two rows into the database.

- ◆ Enter a first name of Jane in the first text box and a last name of Doe in the second. 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.

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

4. Click End to end the program.

❖ **To move through the rows of the table**

1. Write code to implement the Next and Previous buttons.

Add the following procedures to the form.

```
Private Sub btnNext_Click()  
    If Not CustomerTable.MoveNext Then  
        CustomerTable.MoveLast  
    End If  
    DisplayCurrentRow  
End Sub  
Private Sub btnPrevious_Click()  
    If Not CustomerTable.MovePrevious Then  
        CustomerTable.MoveFirst  
    End If  
    DisplayCurrentRow  
End Sub
```

2. Run the application.

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.

❖ To update and delete rows in the table

1. Write code to implement the Update button.

In the code below, the call to UpdateBegin puts the application into update mode. The column values are updated and then the row itself is updated with a call to Update.

Add the following procedure to the form.

```
Private Sub btnUpdate_Click()  
    Dim fname As String  
    Dim lname As String  
    Dim city As String  
    Dim phone As String  
  
    fname = txtFname.Text  
    lname = txtLname.Text  
    city = txtCity.Text  
    phone = txtPhone.Text  
    On Error GoTo UpdateError  
    CustomerTable.UpdateBegin  
    CustomerTable.Column("Fname").StringValue = fname  
    CustomerTable.Column("Lname").StringValue = lname  
    If Len(city) > 0 Then  
        CustomerTable.Column("City").StringValue = city  
    Else  
        CustomerTable.Column("City").SetNull  
    End If  
    If Len(phone) > 0 Then  
        CustomerTable.Column("Phone").StringValue = phone  
    End If  
    CustomerTable.Update  
    DisplayCurrentRow  
    Exit Sub  
  
UpdateError:  
    MsgBox "Error: " & CStr(Err.Description)  
End Sub
```

2. Write code to implement the Delete button.

In the code below, the call to Delete deletes the current row on which the application is positioned.

Add the following procedure to the form.

```
Private Sub btnDelete_Click()  
    If CustomerTable.RowCount = 0 Then  
        Exit Sub  
    End If  
    CustomerTable.Delete  
    CustomerTable.MoveRelative 0  
    DisplayCurrentRow  
End Sub
```

3. Run the application.

Write code to synchronize

The following procedure implements synchronization. Synchronization requires SQL Anywhere Studio. This portion of the tutorial is optional.

❖ To write code for the synchronize button

1. Write code to implement the Synchronize button.

In the code below, the `ULSyncParms` object contains the synchronization parameters. For example, the `ULSyncParms.UserName` property specifies that when MobiLink is started, it will add a new user. The `ULSyncParms.SendColumnNames` property specifies that the column names will be sent to MobiLink so it can generate upload and download scripts.

Add the following procedure to the form.

```
Private Sub btnSync_Click()  
    With Connection.SyncParms  
        .UserName = "afsample"  
        .Stream = ULStreamType.ulTCPIP  
        .Version = "ul_default"  
        .SendColumnNames = True  
    End With  
    Connection.Synchronize  
    DisplayCurrentRow  
End Sub
```

Synchronize your application

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. Start the UltraLite application.

-
3. 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.

4. Synchronize your application.

Click Synchronize.

The MobiLink synchronization server window displays the synchronization progress.

5. When the synchronization is complete, click Next and Previous to move through the rows of the table.

Lesson 4: Deploy to a device

The following procedures deploy your application to either a Palm OS or PocketPC device.

❖ To deploy to a PocketPC device

1. Configure the application settings.
 - ◆ From the MobileVB menu, choose MobileVB Settings. A dialog appears.
 - ◆ In the left pane, choose Dependencies and click the User Dependencies tab.
 - ◆ Click the Add button and select the `c:\tutorial\mvb\tutCustomer.usm`. This indicates to MobileVB that the file should be included in the deployment.
 - ◆ Choose the PocketPC Settings item in the left pane
 - ◆ Enter `|tutorial|mvb` for the Device Installation Path.
 - ◆ Click OK to close the dialog.
2. From the MobileVB menu, choose Deploy to Device, and select the PocketPC device. If a dialog appears asking if you want to save the project, choose Yes.

3. If you are running a version of MobileVB that is older than 3.0, you will also need to copy the UltraLite control to the device.

Copy the file `ulmvb9.dll` to `|Program Files|AppForge` on your device. The file is located in one of the following platform-specific subdirectories of your SQL Anywhere 9 installation:

`|UltraLite|UltraLiteForMobileVB|ce|arm` or
`|UltraLite|UltraLiteForMobileVB|ce|mips`. This step only needs to be performed once per device.

4. On your device, tap Start ► Programs.
5. Tap UltraLiteTutorial to run your application.

❖ To deploy to the Palm device

1. Configure the application settings.
 - ◆ From the MobileVB menu, choose MobileVB Settings.
 - ◆ In the dialog that appears, choose Dependencies in the left pane and click the User Dependencies tab.

-
- ◆ Click the Add button and select `c:\tutorial\mvp\tutCustomer.pdb`. This indicates to MobileVB that the file should be included in the deployment.
 - ◆ Choose the Palm OS Settings item in the left pane and enter Syb3 for the Creator ID. Select a valid HotSync name.
 - ◆ Click OK to close the dialog.
2. From the MobileVB menu, choose Deploy to Device, and select the Palm OS device. If a dialog appears asking if you want to save the project, choose Yes.
 3. HotSync your device and make sure the application gets sent to the device. After the HotSync process is complete, your application files will be extracted on the device.
 4. Click Home on the device and choose UltraLiteTutorial to run your application.

Summary

Learning
accomplishments

During this tutorial, you:

- ◆ created a database schema
- ◆ created an UltraLite for MobileVB application
- ◆ synchronized an UltraLite remote database with an Adaptive Server Anywhere consolidated database
- ◆ gained competence with the process of developing an UltraLite for MobileVB application

More samples

You can find more sample applications and utilities at [iAnywhere CodeXchange](#).

CHAPTER 4

Tutorial: A Sample Application for AppForge Crossfire

About this chapter

This tutorial guides you through the process of using AppForge Crossfire to build an UltraLite application for either a PocketPC or a Palm OS device.

Contents

Topic:	page
Introduction	62
Lesson 1: Create a project architecture	63
Lesson 2: Create the application interface	66
Lesson 3: Write the sample code	68
Lesson 4: Deploy to a device	77
Summary	79

Introduction

This tutorial describes how to use AppForge Crossfire to build an UltraLite application. At the end of the tutorial you will have an application and small database on your Windows CE device that synchronizes with a central consolidated database.

☞ For more information about the table API, see the [“UltraLite for MobileVB API Reference” on page 81](#).

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 AppForge Crossfire installed on your computer. It also assumes a basic familiarity with Crossfire 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].

Note

Crossfire users can perform most of this tutorial without SQL Anywhere Studio. The synchronization sections of the tutorial require SQL Anywhere Studio.

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 9.

❖ To create an UltraLite database schema

1. Create a directory for this tutorial.

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

2. 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

In an application with synchronization, it is usual to choose a global autoincrement or UUID data type for primary keys. An autoincrement method is chosen here to make the tutorial quicker.

◆ **Primary key** Ascending ID

3. Save the database schema.

If you are developing an application for Windows or Windows CE, choose File ► Save and choose `tutcustomer.usm` in 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 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.

Create a Crossfire project

The following procedure creates an AppForge Crossfire project for your application and adds a reference to the UltraLite control.

AppForge tools appear in addition to the standard Visual Basic tools on the toolbar to the left of the development environment.

❖ To create a Crossfire project for UltraLite

1. Start Crossfire.
 - a. Choose Start ► Programs ► AppForge ► Crossfire. The Crossfire Project Manager appears.
 - b. Choose New Project. The Microsoft Development Environment New Project dialog appears.
 - c. In the Project Types window open the Visual Basic Projects folder.
 - d. In the Templates window, click Crossfire Application.
 - e. Leave the project name as CrossfireApp1, and enter your tutorial directory (*c:\tutorial\crossfire*) as the location. Click OK.
 - f. Choose your deployment platform and click OK to create the project. You should now see a Crossfire form in the Microsoft Visual Basic .NET Development Environment.
2. If the Toolbox is not displayed, choose View ► Toolbox to open it. Open the AppForge tab.
3. Scroll down the list of AppForge controls to and double click ULConnectionParms Class.

Troubleshooting

If your Crossfire project does not include a reference to *iAnywhere.UltraLiteForAppForge.dll*, and if the *ULConnectionParms* class does not appear in the list of AppForge controls, you need to register UltraLite with Crossfire. This may occur if, for example, you install Crossfire after installing SQL Anywhere.

☞ For instructions on adding UltraLite to Crossfire, see [“Adding UltraLite to the Crossfire design environment”](#) on page 7.

What's next

You now have an UltraLite database schema and a Crossfire project with an UltraLite control on a form. The next step is to create the application interface.

Lesson 2: Create the application interface

The following procedure uses the form to create a user interface. This example uses labels as outputs, and text boxes and buttons as inputs, similar to a real application.

❖ To add controls to your project

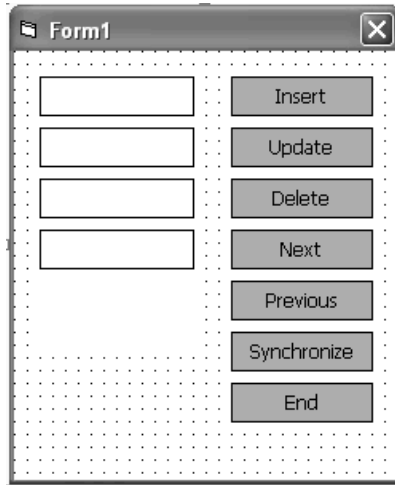
1. From the AppForge controls, add the following controls to your form:

Type	Name	Caption or Text
TextBox	txtFName	
TextBox	txtLName	
TextBox	txtcity	
TextBox	txtphone	
Label	lblID	
Button	btnInsert	Insert
Button	btnUpdate	Update
Button	btnDelete	Delete
Button	btnNext	Next
Button	btnPrevious	Previous
Button	btnSync	Synchronize
Button	btnDone	End

2. Check the application.

- ◆ Choose Build ► Build Solution.

Your form should look as follows:



Lesson 3: Write the sample code

This lesson guides you through writing code to connect to a database, navigate within the database, and manipulate the data in the database.

This lesson also includes instructions for synchronizing your application with an Adaptive Server Anywhere database. This portion of the lesson is optional, and requires SQL Anywhere Studio.

Write code to connect to your database

In this application, you connect to the database during the `Form_Load` event. You can also connect to a database using the general module.

This example uses a the `ULConnectionParms` object to connect to your *tutcustomer* database.

❖ Write code to connect to the UltraLite database

1. Double-click the form to open the Code window.
2. Declare the required UltraLite objects.

Enter the following code immediately after the line `Public Class CrossfireForm1 Inherits System.Windows.Forms.Form`.

```
Public DatabaseMgr As New UltraLiteAFLib.ULDatabaseManager
Public Connection As UltraLiteAFLib.ULConnection
Public CustomerTable As UltraLiteAFLib.ULTable
```

3. Specify the connection parameters.
 - ◆ Select the `ULConnectionParm1` control. In the Properties window, specify connection properties for this database. If you are deploying to a Windows CE device, specify the following properties:

Property	Value
DatabaseOnCE	\\tutorial\crossfire\tutCustomer.udb
DatabaseOnDesktop	c:\tutorial\crossfire\tutCustomer.-udb
SchemaOnCE	\\tutorial\crossfire\tutCustomer.usm
SchemaOnDesktop	c:\tutorial\crossfire\tutCustomer.-usm

If you are deploying to a Palm device, specify the following properties:

Property	Value
DatabaseOnDesktop	c:\tutorial\crossfire\tutCustomer- pdb
DatabaseOnPalm	Syb3
SchemaOnDesktop	c:\tutorial\crossfire\tutCustomer- usm
SchemaOnPalm	tutCustomer

4. In the Form Load event, add code to connect to the database.

The standard way of connecting is to try open a connection to the database specified by the connection string. If the database does not exist, create a new database using the schema. In Crossfire, you must use `add .GetOcx` to the `ULConnectionParms1` object.

```
Private Sub CrossfireForm1_Load(ByVal sender As
    System.Object, _
    ByVal e As System.EventArgs _
) Handles MyBase.Load
    Try
        Connection = _
            DatabaseMgr.OpenConnectionWithParms( _
                ULConnectionParms1.GetOcx)
        ' if the database does not exist, create one
        MsgBox("Connected to an existing database")
    Catch
        If Err.Number = _
            UltraLiteAFLib.ULSQLCode.ulSQLE_ULTRALITE_DATABASE_
            NOT_FOUND _
        Then
            Err.Clear()
            Connection = _
                DatabaseMgr.CreateDatabaseWithParms( _
                    ULConnectionParms1.GetOcx)
            If Err.Number = UltraLiteAFLib.ULSQLCode.ulSQLE_
            NOERROR _
            Then
                MsgBox("Connected to a new database")
            Else
                MsgBox(Err.Description)
            End If
        Else
            MsgBox(Err.Description)
        End If
    End Try
End Sub
```

5. Add the following code to the click event of the End button (btnDone):

```
Connection.Close
End
```

6. Run the application.

- ◆ Choose Debug ► Start.
- ◆ After an initial message box, the form loads.
- ◆ To terminate the application, click End.

On first connecting to the database, UltraLite creates the database file (*tutCustomer.udb*) from the schema file. If you run the application again, the message box indicates that the connection is to an existing database.

Troubleshooting

You now have the basic code in place to connect to your database.

If you see a data type conversion error on the attempt to connect, make sure you have supplied the `GetOcx` method on the `ULConnectionParms1` object.

If the schema file is not found, check that the file exists in the location pointed to by the `ULConnectionParms1` `SchemaOnDesktop` setting.

Write code for navigation and data manipulation

The following procedures implement data manipulation and navigation. The code uses the Table API, which provides methods for moving through and changing the rows of a table, one at a time. For more complex applications, UltraLite provides an implementation of SQL.

❖ To open the table

1. Write code to initialize the table and move to the first row.

This code assigns the Customer table in the database to the `CustomerTable` variable. The call to `Open` opens the table so that the table data can be read or manipulated. It also positions the application before the first row in the table.

Add the following code to the `Form_Load` event, just before the `End Sub` instruction.

```
Try
    CustomerTable = Connection.GetTable("Customer")
    CustomerTable.Open()
Catch
    If Err.Number <> UltraLiteAFLib.ULSQLCode.ulSQLite_NOERROR
        Then
            MsgBox(Err.Description)
        End If
    End Try
```

2. Create a new procedure called `DisplayCurrentRow` and implement it as shown below.

If the table has no rows, the following procedure causes the application to display empty controls. Otherwise, it displays the values stored in each of the columns of the current row of the database.

```
Private Sub DisplayCurrentRow()  
    If CustomerTable.RowCount = 0 Then  
        txtFname.Text = ""  
        txtLname.Text = ""  
        txtCity.Text = ""  
        txtPhone.Text = ""  
        lblID.Caption = ""  
    Else  
        lblID.Caption = _  
            CustomerTable.Column("ID").StringValue  
        txtFname.Text = _  
            CustomerTable.Column("FName").StringValue  
        txtLname.Text = _  
            CustomerTable.Column("LName").StringValue  
        If CustomerTable.Column("City").IsNull Then  
            txtCity.text = ""  
        Else  
            txtCity.Text = _  
                CustomerTable.Column("City").StringValue  
        End If  
        If CustomerTable.Column("Phone").IsNull Then  
            txtphone.Text = ""  
        Else  
            txtphone.Text = _  
                CustomerTable.Column("Phone").StringValue  
        End If  
    End If  
End Sub
```

3. Call `DisplayCurrentRow` from the Form's Activated event. This call ensures that the fields get updated when the application starts.

```
DisplayCurrentRow
```

❖ To insert rows into the table

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 the values in the row to their defaults. For example, the ID column receives the next autoincrement value. The column values are set and then the new row is inserted.

Add the following procedure to the Click event of the Insert button (`btnInsert`).

```

Dim fname As String
Dim lname As String
Dim city As String
Dim phone As String

fname = txtFname.Text
lname = txtLname.Text
city = txtCity.Text
phone = txtPhone.Text

Try
    CustomerTable.InsertBegin
    CustomerTable.Column("FName").StringValue = _
        fname
    CustomerTable.Column("LName").StringValue = _
        lname
    If Len(city) > 0 Then
        CustomerTable.Column("City").StringValue = _
            city
    End If
    If Len(phone) > 0 Then
        CustomerTable.Column("Phone").StringValue = _
            phone
    End If
    CustomerTable.Insert
    CustomerTable.MoveLast
    DisplayCurrentRow
    Exit Sub
Catch
    MsgBox "Error: " & CStr(Err.Description)
End Try

```

2. Run the application.

After an initial message box, the form is displayed.

3. Insert two rows into the database.

- ◆ Enter a first name of Jane in the first text box and a last name of Doe in the second. 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.

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

4. Click End to end the program.

❖ **To move through the rows of the table**

1. Write code to implement the Next and Previous buttons.

Add the following code to the Click event of the Next button (btnNext).

```
If Not CustomerTable.MoveNext Then  
    CustomerTable.MoveLast  
End If  
DisplayCurrentRow
```

Add the following code to the Click event of the Previous button (btnPrevious).

```
If Not CustomerTable.MovePrevious Then  
    CustomerTable.MoveFirst  
End If  
DisplayCurrentRow
```

2. Run the application.

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.

At this stage you can enter data and scroll through the rows of the table.

❖ To update and delete rows in the table

1. Write code to implement the Update button.

In the code below, the call to UpdateBegin puts the application into update mode. The column values are updated and then the row itself is updated with a call to Update.

Add the following code to the Click event of the Update button (btnUpdate):

```
Dim fname As String
Dim lname As String
Dim city As String
Dim phone As String

fname = txtFname.Text
lname = txtLname.Text
city = txtCity.Text
phone = txtPhone.Text
Try
    CustomerTable.UpdateBegin
    CustomerTable.Column("FName").StringValue = fname
    CustomerTable.Column("LName").StringValue = lname
    If Len(city) > 0 Then
        CustomerTable.Column("City").StringValue = city
    Else
        CustomerTable.Column("City").SetNull
    End If
    If Len(phone) > 0 Then
        CustomerTable.Column("Phone").StringValue = phone
    End If
    CustomerTable.Update
    DisplayCurrentRow
    Exit Sub
Catch
    MsgBox "Error: " & CStr(Err.Description)
End Try
```

2. Write code to implement the Delete button.

In the code below, the call to Delete deletes the current row on which the application is positioned.

Add the following code to the Click event of the Delete button (btnDelete):

```
If CustomerTable.RowCount = 0 Then
    Exit Sub
End If
CustomerTable.Delete
CustomerTable.MoveRelative 0
DisplayCurrentRow
```

3. Run the application.

Write code to synchronize

The following procedure implements synchronization. Synchronization requires SQL Anywhere Studio.

❖ To write code for the synchronize button

1. Write code to implement the Synchronize button.

In the code below, the `ULSyncParms` object contains the synchronization parameters. For example, the `ULSyncParms.UserName` property specifies that when MobiLink is started, it will add a new user. The `ULSyncParms.SendColumnNameNames` property specifies that the column names will be sent to MobiLink so it can generate upload and download scripts.

Add the following code to the Click event of the Synchronize button (`btnSync`):

```
With Connection.SyncParms
    .UserName = "CrossfireSample"
    .Stream = UltraLiteAFLib.ULStreamType.ulTCP/IP
    .Version = "ul_default"
    .SendColumnNameNames = True
End With
Connection.Synchronize
DisplayCurrentRow
```

Synchronize your application

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. Start your UltraLite Crossfire application.

-
3. 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.
 4. Synchronize your application.
Click Synchronize.
The MobiLink synchronization server window displays the synchronization progress.
 5. When the synchronization is complete, click Next and Previous to move through the rows of the table.

Lesson 4: Deploy to a device

The following procedures deploy your application to either a Palm OS or PocketPC device.

❖ To deploy to a PocketPC device

1. Configure the application settings.
 - ◆ From the AppForge menu, choose Crossfire Settings. A dialog appears.
 - ◆ In the left pane, choose Dependencies and click the User Dependencies tab.
 - ◆ Click the Add button and select the `c:\tutorial\crossfire\tutCustomer.usm`. This indicates to Crossfire that the file should be included in the deployment.
 - ◆ Choose the PocketPC Settings item in the left pane
 - ◆ Enter `|tutorial|crossfire` for the Device Installation Path.
 - ◆ Click OK to close the dialog.
2. From the AppForge menu, choose Deploy to Device, and select PocketPC/Windows Mobile. If a dialog appears asking if you want to save the project, choose Yes.
3. On your device, tap Start ► Programs.
4. Tap UltraLiteTutorial to run your application.

❖ To deploy to the Palm device

1. Configure the application settings.
 - ◆ From the AppForge menu, choose Crossfire Settings.
 - ◆ In the dialog that appears, choose Dependencies in the left pane and click the User Dependencies tab.
 - ◆ Click the Add button and select `c:\tutorial\mvp\tutCustomer.pdb`. This indicates to Crossfire that the file should be included in the deployment.
 - ◆ Choose the Palm OS Settings item in the left pane and enter Syb3 for the Creator ID. Select a valid HotSync name.
 - ◆ Click OK to close the dialog.
2. From the AppForge menu, choose Deploy to Device, and select the Palm OS device. If a dialog appears asking if you want to save the project, choose Yes.

-
3. HotSync your device and make sure the application gets sent to the device. After the HotSync process is complete, your application files will be extracted on the device.
 4. Click Home on the device and choose UltraLiteTutorial to run your application.

Summary

Learning
accomplishments

During this tutorial, you:

- ◆ created a database schema
- ◆ created an UltraLite application for Crossfire
- ◆ synchronized an UltraLite remote database with an Adaptive Server Anywhere consolidated database

More samples

You can find more sample applications and utilities at [iAnywhere CodeXchange](#).

CHAPTER 5

UltraLite for MobileVB API Reference

About this chapter

This chapter describes the UltraLite MobileVB API, a set of classes and methods that allow you to write MobileVB 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
ULAuthStatusCode enumeration	83
ULColumn class	84
ULColumnSchema class	90
ULConnection class	91
ULConnectionParms class	102
ULDatabaseManager class	105
ULDatabaseSchema class	111
ULIndexSchema class	114
ULPreparedStatement class	116
ULPublicationSchema class	121
ULResultSet class	122
ULResultSetSchema class	128
ULSchemaUpgradeState enumeration	129
ULSQLCode enumeration	130
ULSQLType enumeration	134
ULStreamErrorCode enumeration	135
ULStreamErrorContext enumeration	138
ULStreamErrorID enumeration	139
ULStreamType enumeration	140

Topic:	page
ULSyncParms class	141
ULSyncResult class	145
ULSyncState enumeration	146
ULTable class	148
ULTableSchema class	157

ULAuthStatusCode enumeration

The ULAuthStatusCode is the auth_status synchronization parameter used in the ULSyncResult object.

Constant	Value
ulAuthStatusUnknown	0
ulAuthStatusValid	1000
ulAuthStatusValidButExpiresSoon	2000
ulAuthStatusExpired	3000
ulAuthStatusInvalid	4000
ulAuthStatusInUse	5000

ULColumn class

The ULColumn object allows you to get and set values from a table in a database. Each column object represents a particular value in a table; the row is determined by the ULTable object.

A note on converting from UltraLite database types to Visual Basic types.

UltraLite attempts to convert from the database column data type to the Visual Basic data type. If a conversion cannot be successfully done, then a `ulSQLE_CONVERSION_ERROR` is raised.

☞ For information about the table object, see [“ULTable class” on page 148](#).

Properties

Prototype	Description
BooleanValue As Boolean	Gets or sets the value of this column for the current row as Boolean.
ByteValue As Byte	Gets or sets the value of this column for the current row as Byte.
DatetimeValue As Date	Gets or sets the value of this column for the current row as Date.
DoubleValue As Double	Gets or sets the value of this column for the current row as Double.
IntegerValue As Integer	Gets or sets the value of this column for the current row as Integer.
IsNull As Boolean (read only)	Indicates whether the column value is NULL.
LongValue As Long	Gets or sets the value of this column for the current row as Long.
RealValue As Single	Gets or sets the value of this column for the current row as Single.
Schema As ULColumn-Schema (read only)	Gets the object representing the schema of the column.
StringValue As String	Gets or sets the value of this column for the current row as a String.

Prototype	Description
UUIDValue As String	<p>Gets or sets the value of this column as a UNIQUEIDENTIFIER data type.</p> <p>When getting this property, UltraLite converts the column value to a string representation of the UUID. If the value is not a valid UUID, a <code>SQLLE_CONVERSION_ERROR</code> is raised.</p> <p>When setting this property, UltraLite converts the string form of the UUID to a binary value before storing it in the database.</p>

AppendByteChunk method

Prototype	<pre>AppendByteChunk(_ data As Long, _ data_len As Long _)</pre> <p>Member of UltraLiteAFLib.ULColumn</p>
Description	Appends bytes to the row's column if the type is <code>ulTypeLongBinary</code> or <code>TypeBinary</code> .
Parameters	<p>data In MobileVB, a pointer to an array of Bytes. To get the pointer to the array of bytes, use the Visual Basic <code>VarPtr()</code> function. In Crossfire, a local variable that is an array of Bytes.</p> <p>data_len The number of bytes from the array to append.</p>
Errors set	<p>uISQLE_INVALID_PARAMETER The error occurs if data length is less than 0.</p> <p>uISQLE_CONVERSION_ERROR The error occurs if the column data type is not <code>LONG BINARY</code>.</p>
Example	The following examples append data to the <code>edata</code> column.

```
'MobileVB
Dim data (1 to 512) As Byte
' ...
table.Column("edata").AppendByteChunk( _
    VarPtr(data(1)), 232)

'Crossfire
Dim data (1 to 512) As Byte
' ...
table.Column("edata").AppendByteChunk(data, 232)
```

AppendStringChunk method

Prototype	AppendStringChunk (<i>chunk</i> As String) Member of UltraLiteAFLib.ULColumn
Description	Appends the string to the column if the type is TypeLongString or TypeString.
Parameters	data A string to append to the existing string in a table.
Errors set	uISQLE_CONVERSION_ERROR The error occurs if the column data type is not CHAR or LONG VARCHAR.

GetByteChunk method

Prototype	GetByteChunk (_ <i>offset</i> As Long, _ <i>data</i> As Long, _ <i>data_len</i> As Long, _ <i>filled_len</i> As Long _) As Boolean Member of UltraLiteAFLib.ULColumn
Description	Gets data from a TypeBinary or TypeLongBinary column.
Parameters	offset The offset into the underlying array of bytes. The source offset must be greater than or equal to 0, otherwise a uISQLE_INVALID_PARAMETER error will be raised. data A pointer to an array of bytes. To get the pointer to the array of bytes, use the Visual Basic VarPtr() function. data_len The length of the buffer, or array. The data_len must be greater than or equal to 0. filled_len This is an OUT parameter. After the method is called, it indicates how many bytes were fetched with valid data. If the size of BLOB data is unknown in advance, it is fetched using a fixed-length chunk - one chunk at a time. The last chunk fetched can be smaller than chunk size, so filled_len informs how many bytes of valid data exist in the buffer.
Returns	True if this column value contains more data False if there is no more data for this column in the database.
Errors set	uISQLE_CONVERSION_ERROR The error occurs if the column data type isn't BINARY or LONG BINARY. uISQLE_INVALID_PARAMETER The error occurs if the column data type is BINARY and the offset is not 0 or 1, or, the data length is less than 0.

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

Example

In the following example, edata is a column name.

```
'MobileVB
Dim filled As Long
Dim more_data As Boolean
Dim data (1 to 512) As Byte
more_data = table.Column("edata").GetByteChunk(0, _
VarPtr(data(1)), 512, filled)

'Crossfire
Dim filled As Long
Dim more_data As Boolean
Dim data (1 to 512) As Byte
more_data = table.Column("edata").GetByteChunk(0, _
data, 512, filled)
```

GetStringChunk method

Prototype

```
GetStringChunk( _
    offset As Long, _
    data As String, _
    string_len As Long, _
    filled_len As Long _
) As Boolean
Member of UltraLiteAFLib.ULColumn
```

Description

Gets data from a TypeString or TypeLongString column.

Parameters

offset The character offset into the underlying data from which you start getting the String.

data The variable to receive the string data.

string_length The length of the String you want returned.

filled_len The length of the String fetched.

Returns

True if there is more data to be retrieved from the database.

False if there is no more data.


Errors

uISQLE_CONVERSION_ERROR The error occurs if the column data type isn't CHAR or LONG VARCHAR.

uISQLE_INVALID_PARAMETER The error occurs if the column data type is CHAR and the src_offset is greater than 64K.

The error also occurs if src_offset is less than 0 or string length is less than 0.

SetByteChunk method

Prototype	SetByteChunk (_ <i>data</i> As Long, _ <i>length</i> As Long _) Member of UltraLiteAFLib.ULColumn
Description	Sets data in a TypeBinary or TypeLongBinary column.  To append rather than overwriting data, use the “AppendByteChunk method” on page 85 .
Parameters	data In MobileVB, a pointer to an array of Bytes. To get the pointer to the array of bytes, use the Visual Basic VarPtr() function. In Crossfire, a local variable that is an array of Bytes. length The length of the array.
Errors set	uISQLE_CONVERSION_ERROR The error occurs if the column data type is not BINARY or LONG BINARY. uISQLE_INVALID_PARAMETER The error occurs if the data length is less than 0 or greater than 64K.
Example	In the following example, edata is a column name and the first 232 bytes of the data variable are stored in the database.

```
'MobileVB
Dim data (1 to 512) As Byte
' ...
table.Column("edata").SetByteChunk( VarPtr(data(1)), 232)
'Crossfire
Dim data (1 to 512) As Byte
' ...
table.Column("edata").SetByteChunk( data, 232)
```

SetNull method

Prototype	SetNull() Member of UltraLiteAFLib.ULColumn
Description	Sets the column value to null.

SetToDefault method

Prototype	SetToDefault() Member of UltraLiteAFLib.ULColumn
Description	Sets the current column to its default value as defined by the database

schema. For example, an autoincrement column will be assigned the next available value.

ULColumnSchema class

The ULColumnSchema object allows you to obtain metadata, the attributes of a column, in a table. The attributes are independent of the data in the table.

Properties

Prototype	Description
AutoIncrement As Boolean (read-only)	Indicates whether this column defaults to an autoincrement value. True if AutoIncrement.
DefaultValue As String (read-only)	Gets the value used if one was not provided when a row was inserted.
GlobalAutoIncrement As Boolean (read-only)	Indicates whether this column defaults to a global autoincrement value.
ID As Integer(read-only)	Gets the ID of the column.
Name As String (read-only)	Gets the column name.
Nullable As Boolean (read-only)	Indicates whether the column permits NULLs.
OptimalIndex As ULIndexSchema (read-only)	Gets the index with this column as its first column.
Precision As Integer (read-only)	Gets the precision value for the column if it is of type ulTypeNumeric.
Scale As Integer (read-only)	Gets the scale value for the column if it is of type ulTypeNumeric.
Size As Long (read-only)	Gets the column size for binary, numeric, and character data types.
SQLType As ULSQLType (read-only)	Gets the SQL type assigned to the column when it was created.

ULConnection class

The ULConnection object represents an UltraLite database connection. It provides methods to get database objects like tables, and to synchronize.

Use WithEvents when receiving synchronization progress

When synchronizing, the ULConnection object can also receive progress information. If you wish to receive this information, you must declare your connection WithEvents. You can perform synchronization without declaring your connection WithEvents; however, your connection object will not receive notification of synchronization progress.

Example

To declare a connection **WithEvents**, in a MobileVB form, use the following syntax:

```
Public WithEvents Connection As ULConnection
```

The addition of **WithEvents** makes receipt of synchronization progress information possible.

Properties

The following are properties of ULConnection:

Prototype	Description
AutoCommit As Boolean	Indicates the AutoCommit value. If true, all data changes are committed immediately after they are made. Otherwise, changes are not committed to the database until Commit is called. By default, this property is True.
CollationName As String (read-only)	Gets the database character set and sort order.
DatabaseID As Long	Gets or sets the database ID, which determines the starting value for global autoincrement columns. If the database ID has never been set, the value is -1.
DatabaseNew As Boolean (read-only)	Indicates whether the database was newly created for this connection or not.
GlobalAutoIncrementUsage As Integer (read-only)	Gets the percentage of available global autoincrement values that have been used.

Prototype	Description
IsCaseSensitive As Boolean (read-only)	Indicates whether the database is case sensitive or not.
LastIdentity As Long (read-only)	Gets the most recent value inserted into a column with a default of autoincrement or global autoincrement.
OpenParms As String (read-only)	Gets the string used to open the connection to the database.
Schema As ULDatabaseSchema (read-only)	Gets the ULDatabaseSchema object which represents the definition of the database.
SQLExceptionOffset As Integer (read-only)	If PrepareStatement raises an error, indicates the 1-based offset in the SQL statement where the error was noted. If this value is less than or equal to 0, no offset information is available.
SyncParms As ULSyncParms (read-only)	Gets the synchronization parameters object.
SyncResult As ULSyncResult (read-only)	Gets the results of the most recent synchronization.

CancelSynchronize method

Prototype	CancelSynchronize() Member of UltraLiteAFLib.ULConnection
Description	When called during synchronization, the method cancels the synchronization. The user can only call this method during one of the synchronization events. To allow this the ULConnection object must be declared WithEvents .

ChangeEncryptionKey method

Prototype	ChangeEncryptionKey(newkeyAs String) Member of UltraLiteAFLib.ULConnection
Description	Encrypt the database with the specified key.
Parameters	newkey The new encryption key value for the database.
Example	When you call CreateDatabaseWithParms and pass in the parms object, with a value in place for EncryptionKey, the database is created with encryption.

Another way to change the encryption key is by specifying the new encryption key on the ULConnection object. In this example, “apricot” is the key.

```
Connection.ChangeEncryptionKey( "apricot" )
```

Connections to the database, such as OpenConnectionWithParms, must, after the database is encrypted, specify *apricot* as the EncryptionKey property too. Otherwise, the connection will fail.

Close method

Prototype

Close()

Member of **UltraLiteAFLib.ULConnection**

Description

Closes the connection to the database. No methods on the ULConnection object or any other database object for this connection should be called after this method is called. If a connection is not explicitly closed, it will be implicitly closed when the application terminates.

Commit method

Prototype

Commit()

Member of **UltraLiteAFLib.ULConnection**

Description

Commits outstanding changes to the database. This is only useful if AutoCommit is false.

For more information, see Autocommit under ULConnection [“Properties” on page 91](#)

CountUploadRows method

Prototype

CountUploadRows(
 [*mask* As Long = 0], _
 [*threshold* As Long = -1] _
) As Long

Member of **UltraLiteAFLib.ULConnection**

Description

Returns the number of rows that need to be uploaded when synchronization next takes place.

Parameters

mask An optional, unique identifier that refers to the publications to check. Use 0 for all publications. If not specified, then the value is zero.

threshold An optional parameter representing the maximum number of rows to count. Use -1 to indicate no maximum. If not specified, this value is -1.

Returns Returns the number of rows that need to be uploaded in next synchronization.

GetNewUUID method

Prototype **GetNewUUID()** As String
Member of **UltraLiteAFLib.ULConnection**

Description Returns a new universally unique identifier. The value is a string of the form xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, and is typically stored in a column of data type UNIQUEIDENTIFIER.

Returns Each call returns a new UUID.

GetTable method

Prototype **GetTable(name As String)** As UTable
Member of **UltraLiteAFLib.ULConnection**

Description Returns the **UTable** object for the specified table. You must then open the table before data can be read from it.

Parameters **name** The name of the table sought.

Returns Returns the UTable object.

GrantConnectTo method

Prototype **GrantConnectTo(userid As String, _ password As String _)**
Member of **UltraLiteAFLib.ULConnection**

Description Grants the specified user permission to connect to the database with the given password.

Parameters **userid** The user ID being granted authority to connect.
password The password the user ID must specify to connect.

LastDownloadTime method

Prototype **LastDownloadTime([mask As Long = 0])** As Date
Member of **UltraLiteAFLib.ULConnection**

Description Returns the time of last download for the publication(s).

Parameters **mask** An optional, unique identifier that refers to the publications to check. Use 0 for all publications. If this parameter is omitted, 0 is used.

Returns The last download time in the form of a date.

OnReceive event

Prototype	<pre> OnReceive(nBytes As Long, _ nInserts As Long, _ nUpdates As Long, _ nDeletes As Long _) </pre> <p>Member of UltraLiteAFLib.ULConnection</p>
Description	Reports download information to the application from the consolidated database via MobiLink. This event may be called several times.
Parameters	<p>nBytes Cumulative count of bytes received at the remote application from the consolidated database.</p> <p>nInserts Cumulative count of inserts received at the remote application from the consolidated database.</p> <p>nUpdates Cumulative count of updates received at the remote application from the consolidated database.</p> <p>nDeletes Cumulative count of deletes received at the remote application from the consolidated database.</p>
Example	See the CustDB application for an example of this method.

OnSchemaUpgradeProgress event

Prototype	<pre> OnSchemaUpgradeProgress (nProgress As Long, _ nFinalProgress As Long, _ nOperations As Long _) </pre> <p>Member of UltraLiteAFLib.ULConnection</p>
Description	Reports the progress of schema upgrades, for display in a status dialog.
Parameters	<ul style="list-style-type: none"> ◆ nProgress An approximation of the progress so far. The value is a number between zero and nFinalProgress, enabling you to display a percentage complete value in a dialog box. ◆ nFinalProgress The value of the nProgress when the upgrade completes successfully. ◆ nOperations An approximation of the amount of work done during the upgrade. The value starts at zero and increases as the upgrade proceeds.

It is updated more frequently than `nProgress`. It can be used as a relative measure to compare against other schema upgrades.

See also [◆ “Monitoring schema upgrades” \[UltraLite Database User’s Guide, page 55\]](#)

OnSchemaUpgradeStateChange event

Prototype	OnSchemaUpgradeStateChange (<i>newState</i> As ULSchemaUpgradeState, _ <i>oldState</i> As ULSchemaUpgradeState _)
Description	This event is triggered during a database schema upgrade, when the upgrade reaches a new state. The states available are specified in the ULSchemaUpgradeState enumeration.
Parameters	<ul style="list-style-type: none">◆ newState The state the upgrade is starting.◆ oldState The state just completed.
See also	<ul style="list-style-type: none">◆ “Monitoring schema upgrades” [UltraLite Database User’s Guide, page 55]◆ “ULSchemaUpgradeState enumeration” on page 129

OnSend event

Prototype	OnSend (<i>nBytes</i> As Long, _ <i>nInserts</i> As Long, _ <i>nUpdates</i> As Long, _ <i>nDeletes</i> As Long _) Member of UltraLiteAFLib.ULConnection
Description	Reports upload information from the remote database via MobiLink to the consolidated database. This event may be called several times.
Parameters	<ul style="list-style-type: none">nBytes Cumulative count of bytes sent by the remote application to the consolidated database via MobiLink.nInserts Cumulative count of inserts sent by the remote application to the consolidated database via MobiLink.nUpdates Cumulative count of updates sent by the remote application to the consolidated database via MobiLink.nDeletes Cumulative count of deletes sent by the remote application to the consolidated database via MobiLink.
Example	See the CustDB application for an example of this method.

OnStateChange event

Prototype	OnStateChange (<i>newState</i> As ULSyncState, _ <i>oldState</i> As ULSyncState _) Member of UltraLiteAFLib.ULConnection
Description	This event is called whenever the state of the synchronization changes. For more information, see “ULSyncState enumeration” on page 146 .
Parameters	newState The state that the synchronization process is about to enter. oldState The state that the synchronization process just completed.
Example	See the CustDB application for an example of this method.

OnTableChange event

Prototype	OnTableChange (<i>newTableIndex</i> As Long, _ <i>numTables</i> As Long _) Member of UltraLiteAFLib.ULConnection
Description	This event is called whenever the synchronization process begins synchronizing another table.
Parameters	newTableIndex The index number of the table currently being synchronized. This number is not the same as the table ID, therefore, it cannot be used with the <code>ULDatabaseSchema.GetTableName</code> method. numTables The number of tables eligible to be synchronized.
Example	See the CustDB application for an example of this method.

PrepareStatement method

Prototype	PrepareStatement (<i>sqlStatement</i> As String, _ <i>persistent_name</i> As String _) As ULPreparedStatement Member of UltraLiteAFLib.ULConnection
Description	Prepares a SQL statement for execution.
Parameters	sqlStatement The SQL statement to prepare. persistent_name For Palm applications, the persistent name of the statement.

Returns Returns a `ULPreparedStatement`. If there was a problem preparing the statement, an error will be raised. The offset into the statement where the error occurred can be determined from the `SQLExceptionOffset` property.

ResetLastDownloadTime method

Prototype **ResetLastDownloadTime([mask As Long])**
Member of **UltraLiteAFLib.ULConnection**

Description Resets the time of the most recent download for the publications specified in the mask.

Parameters **mask** The mask of the publications to reset. The default is 0, specifying all publications.

RevokeConnectFrom method

Prototype **RevokeConnectFrom(userID As String)**
Member of **UltraLiteAFLib.ULConnection**

Description Revokes the specified user's ability to connect to the database.

Parameters **userid** The user ID whose authority to connect is being revoked.

Rollback method

Prototype **Rollback()**
Member of **UltraLiteAFLib.ULConnection**

Description Rolls back outstanding changes to the database. This is only useful if `AutoCommit` is false.

RollbackPartialDownload method

Roll back the changes from a failed synchronization.

Prototype **RollbackPartialDownload ()**
Member of **UltraLiteAFLib.ULConnection**

Description 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.

See also

- ◆ “Resuming failed downloads” [*MobiLink Administration Guide*, page 74]
- ◆ “Keep Partial Download synchronization parameter” [*MobiLink Clients*, page 321]

- ◆ “Partial Download Retained synchronization parameter” [*MobiLink Clients*, page 324]
- ◆ “Resume Partial Download synchronization parameter” [*MobiLink Clients*, page 327]

StartSynchronizationDelete method

Prototype	StartSynchronizationDelete() Member of UltraLiteAFLib.ULConnection
Description	Once StartSynchronizationDelete is called, all delete operations are again synchronized.

StopSynchronizationDelete method

Prototype	StopSynchronizationDelete() Member of UltraLiteAFLib.ULConnection
Description	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.

StringToUUID method

Prototype	StringToUUID(<i>s_uuid</i> As String, _ <i>buffer_16_bytes</i> As Long _) Member of UltraLiteAFLib.ULConnection
Description	Converts a universally unique identifier represented as a String in the form xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx to a Byte array of 16 bytes. In a MobileVB application, it may be useful to refer to them in their string format. Consequently, the UUIDValue property on the ULColumn object converts from string to binary(16) and vice versa. The StringToUUID function is provided as an easy way to convert a MobileVB String to a Byte array. It does not reference the UltraLite database in any way.

The pointer to the buffer:

The pointer to the buffer must be declared as at least 16 bytes. Since Visual Basic does not provide bounds checking, memory could be overwritten if the buffer is too small. In MobileVB, use the VarPtr() function to get the pointer to the buffer. See also ULColumn.UUIDValue property

Not needed in newer databases

In databases created before version 9.0.2, the UNIQUEIDENTIFIER data type is defined as a user-defined data type and functions are needed to convert between binary and string representations of UUID values.

In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type is a native data type and UltraLite carries out conversions as needed. The StringToUUID function is therefore not needed.

☞ For more information, see “[UNIQUEIDENTIFIER data type \[Binary\]](#)” [*ASA SQL Reference*, page 75].

Parameters

s_uuid A Universally Unique Identifier passed in as a string. You can obtain a new string UUID using GetNewUUID.

buffer_16_bytes A pointer to a byte array that has at least 16 elements. Use the VarPtr() function to get the pointer value.

Example

The following example will convert the string form of the UUID 0a141e28-323c-4650-5a64-6e78828c96a0 to a binary array:

```
Dim buff(1 to 16) As Byte
conn.StringToUUID( "0a141e28-323c-4650-5a64-6e78828c96a0",
    VarPtr(buff(1)) )
```

Synchronize method

Prototype

Synchronize()
Member of **UltraLiteAFLib.ULConnection**

Description

Synchronizes a consolidated database using MobiLink. This function does not return until synchronization is complete, but you can be notified of events if the connection was declared WithEvents.

UUIDToString method

Prototype

UUIDToString(*buffer_16_bytes* As Long) As String
Member of **UltraLiteAFLib.ULConnection**

Description

Converts a UUID from a byte array to a string of the form
XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX.

Not needed in newer databases

In databases created before version 9.0.2, the UNIQUEIDENTIFIER data type is defined as a user-defined data type and functions are needed to convert between binary and string representations of UUID values.

In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type is a native data type and UltraLite carries out conversions as needed. The UUIDToString function is therefore not needed.

☞ For more information, see “UNIQUEIDENTIFIER data type [Binary]” [ASA SQL Reference, page 75].

Parameters

buffer_16_bytes An array of 16 bytes containing a UUID.

Returns

Each call returns a string of the form

XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

ULConnectionParms class

The ULConnectionParms object allows you to set userID, password, schema file, file on your desktop, and numerous other parameters that specify your connection.

Properties

The ULConnectionParms class specifies parameters for opening a connection to an UltraLite database.

In UltraLite for MobileVB, ensure you have the ULConnectionParms object on your form and you set connection properties in the ConnectionParms dialog. You use the ULConnectionParms object in conjunction with **ULDatabaseManager.CreateDatabaseWithParms** and **ULDatabaseManager.OpenConnectionWithParms** methods.

Note

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.

☞ For more information about the meaning of these parameters, see “Connection Parameters” [*UltraLite Database User’s Guide*, page 63].

Prototype	Description
AdditionalParms As String (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].
CacheSize As Integer (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].

Prototype	Description
ConnectionName As String (read-write)	<p>A name for the connection. This is needed only if you create more than one connection to the database.</p> <p>☞ See “Connection Name connection parameter” [<i>UltraLite Database User’s Guide</i>, page 74].</p>
DatabaseOnCE As String (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>
DatabaseOnDesktop As String (read-write)	<p>The filename of the database during development.</p> <p>☞ See “Database On Desktop connection parameter” [<i>UltraLite Database User’s Guide</i>, page 70].</p>
DatabaseOnPalm As String (read-write)	<p>The creator ID for the UltraLite database on the Palm device.</p> <p>☞ See “Database On Palm connection parameter” [<i>UltraLite Database User’s Guide</i>, page 71].</p>
EncryptionKey As String (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>
PageSize As Integer (read-write)	<p>The page size for the database.</p> <p>☞ See “Page Size connection parameter” [<i>UltraLite Database User’s Guide</i>, page 83].</p>
ParmsUsed As String (read-only)	<p>The parameters used by the ULDatabaseManager. Useful for debugging purposes.</p>

Prototype	Description
Password As String (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>
ReserveSize As Integer (read-write)	<p>The amount of file system space to reserve for storage of UltraLite persistent data.</p> <p>☞ See “Reserve Size connection parameter” [<i>UltraLite Database User’s Guide</i>, page 84].</p>
SchemaOnCE As String (read-write)	<p>The schema filename deployed to PocketPC.</p> <p>☞ See “Schema On CE connection parameter” [<i>UltraLite Database User’s Guide</i>, page 78].</p>
SchemaOnDesktop As String (read-write)	<p>The schema filename during development.</p> <p>☞ See “Schema On Desktop connection parameter” [<i>UltraLite Database User’s Guide</i>, page 79].</p>
SchemaOnPalm As String (read-write)	<p>The schema PDB on the Palm device.</p> <p>☞ See “Schema On Palm connection parameter” [<i>UltraLite Database User’s Guide</i>, page 80].</p>
UserID As String (read-write)	<p>The authenticated user for the database. Databases are initially created with one authenticated user <i>DBA</i>. 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>.</p> <p>☞ See “User ID connection parameter” [<i>UltraLite Database User’s Guide</i>, page 76].</p>
VFSONPalm As Boolean (read-write)	<p>Indicates whether the Palm database is on a virtual file system (true) or on the Palm store (false).</p> <p>☞ See “VFS On Palm parameter” [<i>UltraLite Database User’s Guide</i>, page 81].</p>

ULDatabaseManager class

The ULDatabaseManager class is used to manage connections and databases. Your application should only have one instance of this object. Creating a database and establishing a connection to it is a necessary first step in using UltraLite. It is suggested that you use CreateDatabaseWithParms, OpenConnectionWithParms and DropDatabaseWithParms, and include checks in your code to ensure that you are connected properly before attempting any DML with the database.

Parms or no parms?

Two types of methods exist for creating, opening and dropping connections to your database: Methods WithParms and methods that do not use the ULConnectionParms object. Methods WithParms allow you to use a ULConnectionParms object to manipulate connection parameters with ease and accuracy. Methods that do not use the ULConnectionParms object require that you can successfully create a connections string and use that connection string in a CreateDatabase, OpenConnection or DropDatabase method.

Properties

The following are properties of ULDatabaseManager:

Prototype	Description
Version As String (read-only)	Gets the version string of the UltraLite component.

CreateDatabase method

CreateDatabase creates a new database and returns a connection to it.

Prototype

CreateDatabase(*parms* As String) As ULConnection
Member of **UltraLiteAFLib.ULDatabaseManager**

Description

Creates a new database and returns a connection to it. It fails if the specified database already exists. A valid schema file must be specified to successfully create a database. To alter the schema of an existing database, use the ULDatabaseSchema ApplyFile method.

Caution

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

☞ For more information about ApplyFile, see “[ULDatabaseSchema class](#)” on page 111 and “[ApplyFile method](#)” on page 112.

Parameters

parms A semicolon-separated list of database creation parameters.

Note for VFS card for Palm users

The Palm_fs=vfs parameter needs to be specified both for CreateDatabase and OpenConnection methods if you want to have the database reside on the virtual file system.

☞ For information about connection parameters, see “[Connection Parameters](#)” [*UltraLite Database User's Guide*, page 63].

☞ For more information about the Palm_fs parameter, see “[VFS On Palm parameter](#)” [*UltraLite Database User's Guide*, page 81].

Returns

Returns a connection to a newly created UltraLite database.

Examples

The following code creates a ULDatabaseManager object. This is the first object you create when writing for UltraLite for MobileVB. Note that CreateDatabase requires that no .udb file exists, and OpenConnection is used when a .udb file already exists.

```
Dim conn_parms As String
Dim open_parms As String
Dim schema_parms As String

conn_parms = "uid=DBA;pwd=SQL"
open_parms = conn_parms & ";" & _
    "PALM_DB=Syb3;file_name=c:\tutorial\tutCustomer.udb"
schema_parms = open_parms & ";" & _
    "PALM_SCHEMA=tutCustomer;" & _
    "schema_file=c:\tutorial\tutCustomer.usm"

On Error Resume Next

Set Connection = DatabaseMgr.OpenConnection(open_parms)
If Err.Number = _
    ULSQLCode.ulSQLE_DATABASE_NOT_FOUND _
Then
    Err.Clear
    Set Connection = _
        DatabaseMgr.CreateDatabase(schema_parms)
    If Err.Number <> 0 Then
        MsgBox Err.Description
    End If
End If
```

☞ For information about connection parameters, see “[OpenConnection method](#)” on page 108.

CreateDatabaseWithParms method

CreateDatabaseWithParms creates a new database using a connection parameter object, and returns a connection to it.

Prototype

CreateDatabaseWithParms(*parms* As ULConnectionParms)
As ULConnection
Member of **UltraLiteAFLib.ULDatabaseManager**

Description

Creates a new database and returns a connection to it. It fails if the specified database already exists. A valid schema file must be specified to successfully create a database. To alter the schema of an existing database, use the **ULDatabaseSchema.ApplyFileWithParms** method.

Caution

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

Parameters

parms A ULConnectionParms object that holds a set of connection parameters.

Note for VFS card for Palm users

You specify VFSOnPalm in the ULConnectionParms interface.

☞ For more information about the Palm_fs parameter, see “VFS On Palm parameter” [*UltraLite Database User's Guide*, page 81] .

Returns

Returns a connection to a newly created UltraLite database. Fails if the specified database already exists.

Examples

The following example assumes you have placed the ULConnectionParms object on your form, named it **LoginParms** and have specified the database locations and schema locations in the Connection parms properties window.

The following code creates a ULDatabaseManager object. This is the first object you create when writing for UltraLite for MobileVB.

Note that CreateDatabaseWithParms requires that no .udb file exists, and OpenConnectionWithParms is used when a .udb file already exists.

```
DatabaseMgr.DropDatabaseWithParms LoginParms
Set Connection = DatabaseMgr.CreateDatabaseWithParms(LoginParms)
```

DropDatabase method

The DropDatabase method deletes a database file.

Prototype	DropDatabase(<i>parms</i> As String) Member of UltraLiteAFLib.ULDatabaseManager
Description	Deletes the database file. All information in the database file is lost. Fails if the specified database does not exist, or if there exist open connections at the time of DropDatabase is executed.
Parameters	parms The filename for the database.
Example	The following example drops a database:

```
Dim parms As String
parms = "PALM_DB=Sybl;NT_FILE=c:\temp\ul_CustDB.udb"
DropDatabase(parms)
```

DropDatabaseWithParms method

The DropDatabaseWithParms method deletes a database file.

Prototype	DropDatabaseWithParms(<i>parms</i> As ULConnectionParms) Member of
Description	Deletes the database file. All information in the database file is lost.
Parameters	parms The ULConnectionParms object containing vital connection parameters .
Example	The following example assumes you have declared and instantiated a ULConnectionParms object named LoginParms and used it to specify the database location.

```
DatabaseMgr.DropDatabaseWithParms LoginParms
```

OpenConnection method

Prototype	OpenConnection(<i>connparms</i> As string) As ULConnection Member of UltraLiteAFLib.ULDatabaseManager
Description	<p>If a database exists, use this method to connect to the database. If a database does not exist, or the connection parameters are invalid, the call will fail. Use the error object to determine why the call failed.</p> <p>The function returns a ULConnection object which provides an open connection to a specified UltraLite database. The database filename is specified using the connparms string. Parameters are specified using a sequence of name=value pairs. If no user ID or password is given, the default is used.</p> <p>It should contain a value of the form</p>

```
file_name=UDBFILE
DBF=UDBFILE
palm_db=CreatorID.
```

Parameters	<p>connparms The parameter used to establish a connection to a database. Parameters are specified as a semicolon separated list of keyword=value pairs. If no user ID or password is given, the default is used.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Note for Palm users The Palm_fs=vfs parameter needs to be specified both for CreateDatabase and OpenConnection methods when using a database on the Palm virtual file system.</p> </div>
Returns	<p>☞ For more information about the Palm_fs parameter, see “VFS On Palm parameter” [<i>UltraLite Database User’s Guide</i>, page 81] .</p> <p>The ULConnection object is returned if the connection was successful.</p>
Example	<p>The following example creates a new database connection from the CustDB sample application:</p> <pre>Set Connection = DatabaseMgr.OpenConnection("file_name=d:\Dbfile.udb;palm_db=Syb3;CE_file=\myapp\MyDB.udb")</pre>

OpenConnectionWithParms method

Prototype	<p>OpenConnectionWithParms(<i>connparms</i> As ULConnectionParms) As ULConnection Member of UltraLiteAFLib.ULDatabaseManager</p>
Description	<p>If a database exists, use this method to receive a connection. If a database does not exist, or the connection parameters are invalid, the call will fail. Use the error object to determine why the call failed.</p> <p>The function returns a ULConnection object which provides an open connection to a specified UltraLite database. The database filename is specified using the connparms object. Parameters are specified using a sequence of name=value pairs. If no user ID or password is given, the default is used.</p>
Parameters	<p>connparms The parameters defining this connection.</p>
Returns	<p>The ULConnection object is returned if the connection was successful.</p>
Example	<p>The following example assumes you have placed the ULConnectionParms object on your form, named it LoginParms and have specified the database locations and schema locations in the ULConnection parms properties window.</p>

```
Set Connection = DatabaseMgr.OpenConnection(LoginParms)
```

ULDatabaseSchema class

The ULDatabaseSchema object allows you to obtain the attributes of the database to which you are connected.

Properties

The following are properties of ULDatabaseSchema:

Prototype	Description
DateFormat As String (read-only)	Gets the format for dates retrieved from the database; 'YYYY-MM-DD' is the default. The format of the date retrieved depends on the format used when you created the schema file.
DateOrder As String (read-only)	Indicates the interpretation of date formats; valid values are 'MDY', 'YMD', or 'DMY'.
NearestCentury As String (read-only)	Indicates the interpretation of two-digit years in string-to-date conversions. This is a numeric value that acts as a rollover point. Two digit years less than the value are converted to 20yy, while years greater than or equal to the value are converted to 19yy. The default is 50.
Precision As String (read-only)	Gets the maximum number of digits in the result of any decimal arithmetic.
PublicationCount As Integer (read-only)	Gets the number of publications in the connected database.
Signature As String (read-only)	Gets the database signature, an internal identifier representing the database schema.
TableCount As Integer (read-only)	Gets the number of tables in the connected database.
TimeFormat As String (read-only)	Gets the format for times retrieved from the database.
TimestampFormat As String (read-only)	Gets the format for timestamps retrieved from the database.

ApplyFile method

Prototype **ApplyFile(*parms* As String)**
Member of **UltraLiteAFLib.ULDatabaseSchema**

Description Changes the schema of this database. *Parms* points to the schema file(s) you are applying to the database. This method is only useful on those occasions where you want to modify your existing database structure.

Caution

Although ApplyFile is very safe, ApplyFile can cause data loss under a number of circumstances including (1) if columns are deleted, or (2) if the data type for a column is changed to an incompatible type, or (3) if you upgrade an 8.0.2 database using ApplyFile in UltraLite 9.0.

Parameters **parms** The files containing the changes you wish to make to your database schema.

Example

```
DatabaseSchema.ApplyFile( _  
    "schema_file=MySchemaFile.usm;palm_schema=MySchema" )
```

ApplyFileWithParms method

Prototype **ApplyFileWithParms(*parms* As ULConnectionParms)**
Member of **UltraLiteAFLib.ULDatabaseSchema**

Description Upgrades the schema of this database using the parameter object *Parms*, which points to the schema file(s) you are applying to the database. This method is only useful on those occasions where you want to modify your existing database structure.

Caution

Although ApplyFile is very safe, ApplyFileWithParms can cause data loss under a number of circumstances including (1) if columns are deleted, or (2) if the data type for a column is changed to an incompatible type, or (3) if you upgrade an 8.0.2 database using ApplyFile in UltraLite 9.0.

Parameters **parms** The object identifying the schema file to apply.

GetPublicationName method

Prototype **GetPublicationName(*id* As Integer) As String**
Member of **UltraLiteAFLib.ULDatabaseSchema**

Description Returns the name of the specified publication. The publication *ID* can range from 1 to PublicationCount.

Parameters	id The <i>id</i> is the identifier of the publication whose name will be returned.
Returns	Returns the name of a publication in the connected database. For information about the <code>ULPublicationSchema</code> object, see “ULPublicationSchema class” on page 121 . For more information, see <code>ULDatabaseSchema</code> “Properties” on page 111

GetPublicationSchema method

Prototype	GetPublicationSchema(<i>Name</i> As String) As <code>ULPublicationSchema</code> Member of UltraLiteAFLib.ULDatabaseSchema
Description	Use the publication name to retrieve the <code>ULPublicationSchema</code> object.
Parameters	name The <i>name</i> of the publication.
Returns	Returns the <code>ULPublicationSchema</code> object.

GetTableName method

Prototype	GetTableName(<i>id</i> As Integer) As String Member of UltraLiteAFLib.ULDatabaseSchema
Description	Returns the name of the table in the connected database that corresponds to the <i>id</i> value you supply. The <code>TableCount</code> property returns the number of tables in the connected database. Each table has a unique number from 1 to the <code>TableCount</code> value, where 1 is the first table in the database, 2 is the second table in the database, and so on. The <i>id</i> for a table may change after a database has had its schema changed.
Parameters	id The <i>id</i> of the table.
Returns	Returns the name of the table for the specified <i>id</i> .

ULIndexSchema class

The ULIndexSchema object allows you to obtain the attributes of an index. An index is an ordered set of columns by which data in a table will be sorted. The primary use of an index is to order the data in a table by one or more columns.

An index can be a foreign key, which is used to maintain referential integrity in a database.

Properties

Prototype	Description
ColumnCount As Integer (read-only)	Gets the number of columns in the index
ForeignKey As Boolean (read-only)	Indicates whether this is a foreign key.
ForeignKeyCheckOnCommit (read-only)	Indicates whether referential integrity is checked only when a commit is done (TRUE) or immediately (FALSE).
ForeignKeyNullable (read-only)	Indicates whether the foreign key columns allow NULL.
Name As String (read-only)	Gets the name of the index
PrimaryKey As Boolean (read-only)	Gets whether this is the primary key for this table.
ReferencedIndexName As String (read-only)	Gets the name of the index referenced by this index if it is a foreign key
ReferencedTableName As String (read-only)	Gets the name of the table referenced by this index if it is a foreign key
UniqueIndex As Boolean (read-only)	Indicates whether values in the index must be unique.
UniqueKey As Boolean (read-only)	Indicates whether the index is a unique constraint on a table. If True, the columns in the index are unique and do not permit NL values

GetColumnName method

Prototype	GetColumnName (<i>col_pos_in_index</i> As Integer) As String Member of UltraLiteAFLib.ULIndexSchema
Description	Used to return the names of the columns in the index. The parameter <i>col_pos_in_index</i> must be at least 1 and at most ColumnCount.
Parameters	col_pos_in_index The column position in the index.
Returns	Returns the name of a column in the index.

IsColumnDescending method

Prototype	IsColumnDescending (<i>col_name</i> As String) As Boolean Member of UltraLiteAFLib.ULIndexSchema
Description	Indicates whether the specified column in the index is in descending order.
Parameters	col_name The index column name.
Returns	True if the column is descending. False if the column is ascending.

ULPreparedStatement class

The `ULPreparedStatement` represents a pre-compiled SQL statement ready for execution. You can use Prepared Statement to run a SQL query. You can also use the `ULPreparedStatement` to execute the same statement multiple times using numerous input parameters. Since the prepared statement is precompiled, any further additions beyond the first execution take very little extra processing. Use `ULPreparedStatement` and Dynamic SQL when you want relatively fast DML over multiple rows.

Properties

Prototype	Description
<code>HasResultSet</code> As Boolean (read-only)	Indicates whether the prepared statement generates a result set. True if the statement has a result set, otherwise, false. If true, <code>ExecuteQuery</code> should be called instead of <code>ExecuteStatement</code> .
<code>Plan</code> (read-only) As String	Gets the access plan UltraLite will use to execute a query. This property is intended primarily for use during development.
<code>ResultSetSchema</code> As <code>ULResultSetSchema</code> (read-only)	Gets the schema description for the result set if the statement is for a result set

AppendByteChunkParameter method

Prototype

AppendByteChunkParameter (
param_id As Integer,
data As Long,
data_len As Long)

Member of **UltraLiteAFLib.ULPreparedStatement**

Description

Appends the buffer of bytes to the row's column if the type is `ulTypeLongBinary`.

Parameters

parameter_id The 1-based parameter number to set.

data The array of bytes to append.

data_len The number of bytes from the array to append.

Errors set	uISQLE_INVALID_PARAMETER The error occurs if the data length is less than 0.
	uISQLE_CONVERSION_ERROR The error occurs if the column data type is not LONG BINARY.

AppendStringChunkParameter method

Prototype	AppendStringChunkParameter (<i>param_id</i> As Integer , <i>chunk</i> As String) Member of UltraLiteAFLib.ULPreparedStatement
Description	Appends the string to the column if the type is ulTypeLongString.
Parameters	parameter_id The 1-based parameter number to set. chunk A string to append to the existing string in a table.
Errors set	uISQLE_CONVERSION_ERROR The error occurs if the column data type is not LONG VARCHAR.

Close method

Prototype	Close () Member of UltraLiteAFLib.ULPreparedStatement
Description	Frees resources associated with the ULPreparedStatement.

ExecuteQuery method

Prototype	ExecuteQuery () As ULResultSet Member of UltraLiteAFLib.ULPreparedStatement
Description	Executes the query and returns a result set.
Returns	A ULResultSet object. The ULResultSet is the data you requested in your SELECT statement. To describe the product of your query, see “ULResultSetSchema class” on page 128

ExecuteStatement method

Prototype	ExecuteStatement () As Long Member of UltraLiteAFLib.ULPreparedStatement
Description	Executes the statement.
Returns	The number of rows updated.

SetBooleanParameter method

Prototype **SetBooleanParameter**(
 param_number As Integer
 param_value As Boolean
)
Member of **UltraLiteAFLib.ULPreparedStatement**

Description Set the parameter to the Boolean value passed in.

Parameters **param_number** The 1-based parameter number to set.
 param_value The value the parameter should receive.

SetByteChunkParameter method

Prototype **SetByteChunkParameter**(
 param_number As Integer,
 data As Long,
 data_len As Long
)
Member of **UltraLiteAFLib.ULPreparedStatement**

Description Sets data in a binary or long binary column.

Parameters **param_number** The 1-based parameter number to set.
 data An array of bytes.
 data_len The number of bytes from the array to set. SetByteChunk writes over the current content. To append to an existing value, see [“AppendByteChunkParameter method” on page 116](#)

SetByteParameter method

Prototype **SetByteParameter**(
 param_number As Integer
 param_value As Byte
)
Member of **UltraLiteAFLib.ULPreparedStatement**

Description Set the parameter to the Byte value passed in.

Parameters **param_number** The 1-based parameter number to set.
 param_value The value the parameter should receive.

SetDatetimeParameter method

Prototype	SetDatetimeParameter (<i>param_number</i> As Integer <i>param_value</i> As String)
	Member of UltraLiteAFLib.ULPreparedStatement
Description	Set the parameter to the Datetime value passed in.
Parameters	param_number The 1-based parameter number to set. param_value The value the parameter should receive.

SetDoubleParameter method

Prototype	SetDoubleParameter (<i>param_number</i> As Integer <i>param_value</i> As String)
	Member of UltraLiteAFLib.ULPreparedStatement
Description	Set the parameter to the Double value passed in.
Parameters	param_number The 1-based parameter number to set. param_value The value the parameter should receive.

SetIntegerParameter method

Prototype	SetIntegerParameter (<i>param_number</i> As Integer <i>param_value</i> As String)
	Member of UltraLiteAFLib.ULPreparedStatement
Description	Set the parameter to the Integer value passed in.
Parameters	param_number The 1-based parameter number to set. param_value The value the parameter should receive.

SetLongParameter method

Prototype	SetLongParameter (<i>param_number</i> As Integer <i>param_value</i> As String)
	Member of UltraLiteAFLib.ULPreparedStatement

Description Set the parameter to the Long value passed in.

Parameters **param_number** The 1-based parameter number to set.
param_value The value the parameter should receive.

SetNullParameter method

Prototype **SetNullParameter**(*param_id* As Integer)
Member of **UltraLiteAFLib.ULPreparedStatement**

Description Set the parameter to NL.

Parameters **parameter_id** The 1-based parameter number to set.

SetRealParameter method

Prototype **SetRealParameter**(
param_number As Integer
param_value As String
)
Member of **UltraLiteAFLib.ULPreparedStatement**

Description Set the parameter to the Long value passed in.

Parameters **param_number** The 1-based parameter number to set.
param_value The value the parameter should receive.

SetStringParameter method

Prototype **SetStringParameter**(
param_number As Integer
param_value As String
)
Member of **UltraLiteAFLib.ULPreparedStatement**

Description Set the parameter to the string passed in.

Parameters **param_number** The 1-based parameter number to set.
param_value The value the parameter should receive.

ULPublicationSchema class

The ULPublicationSchema object allows you to obtain the attributes of a publication.

Properties

Prototype	Description
Mask As Long (read-only)	Gets the mask for the publication
Name As String (read-only)	Gets the name of the publication

ContainsTable method

Prototype	ContainsTable (<i>name</i> As String) As Boolean Member of UltraLiteAFLib.ULPublicationSchema
Description	Indicates whether the specified table is part of this publication.
Parameters	name The target table name.
Returns	True if the table is in the publication. False if the table is not in the publication.

ULResultSet class

The ULResultSet object moves over rows returned by a SQL query. Since the ULResultSet object contains the data returned by a query, you must refresh any query resultset after you have performed DML operations such as INSERT, UPDATE or DELETE. To do this, you should perform ExecuteQuery after you perform ExecuteStatement.

Properties

Prototype	Description
BOF As Boolean (read-only)	Indicates whether the current row position is before the first row. Returns True if the current row position is before the first row, otherwise false.
EOF As Boolean (read-only)	Indicates whether the current row position is after the last row. EOF is true if beyond the last row, otherwise false.
RowCount As Long (read-only)	The number of rows in the result set.
Schema As ULResultSetSchema (read-only)	The schema description for this result set.

Close method

Prototype

Close()

Member of **UltraLiteAFLib.ULResultSet**

Description

Frees all resources associated with this object.

GetByteChunk method

Prototype

GetByteChunk (*_*
index As Integer, *_*
src_offset As Long, *_*
data As Long, *_*
data_len As Long, *_*
filled_len As Long *_*
) As Boolean

Member of **UltraLiteAFLib.ULResultSet**

Description

Fills the buffer passed in (which should be an array) with the binary data in the column. Suitable for BLOBS.

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>SQL_INVALID_PARAMETER</code> error will be raised. A buffer bigger than 64K is also permissible.</p> <p>data A pointer to an array of bytes. To get the pointer to the array of bytes, use the Visual Basic <code>VarPtr()</code> function.</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> <p>filled_len The number of bytes fetched. Because you don't know how big the BLOB data is in advance, you generally fetch it using a fixed-length chunk, one chunk at a time. The last chunk may be smaller than your chunk size. <code>filled_len</code> reports how many bytes were actually fetched.</p>
Returns	The number of bytes read.
Errors set	<p>uISQLE_CONVERSION_ERROR The error occurs if the column data type is not <code>BINARY</code> or <code>LONG BINARY</code>.</p> <p>uISQLE_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> <p>The error also occurs if the column data type is <code>LONG BINARY</code> and the offset is less than 1.</p>
Example	<p>In the following example, <code>edata</code> is a column name. If the <code>data_len</code> parameter passed in is not sufficiently long, the entire application will terminate.</p>

```
Dim data (512) As Byte
...
table.Column("edata").GetByteChunk(0,data)
```

GetStringChunk method

Prototype	<pre>GetStringChunk(_ index As Integer, _ offset As Long, _ data As String, _ string_len As Long, _ filled_len As Long _) As Boolean</pre> <p>Member of UltraLiteAFLib.ULResultSet</p>
Description	Fills the string passed in with the binary data in the column. Suitable for Long Varchars.
Parameters	<p>index The 1-based column ID of the target column.</p>

offset The character offset into the underlying data from which you start getting the string.

data The data string.

string_len The length of the string you want returned.

filled_len The length of the string filled.

Returns

Gets BLOB data from a binary or long binary column.

Errors set

uISQLE_CONVERSION_ERROR The error occurs if the column data type is not CHAR or LONG VARCHAR.

uISQLE_INVALID_PARAMETER The error occurs if the column data type is CHAR and the `src_offset` is greater than 64K.

This error also occurs if `offset` is less than 0 or `string length` is less than 0.

MoveAfterLast method

Prototype

MoveAfterLast()
Member of **UltraLiteAFLib.ULResultSet**

Description

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

MoveBeforeFirst method

Prototype

MoveBeforeFirst()
Member of **UltraLiteAFLib.ULResultSet**

Description

Moves to a position before the first row.

MoveFirst method

Prototype

MoveFirst() As Boolean
Member of **UltraLiteAFLib.ULResultSet**

Description

Moves to the first row.

Returns

True if successful.

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

MoveLast method

Prototype

MoveLast() As Boolean
Member of **UltraLiteAFLib.ULResultSet**

Description

Moves to the last row.

Returns

True if successful.

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

MoveNext method

Prototype	MoveNext() As Boolean Member of UltraLiteAFLib.ULResultSet
Description	Moves to the next row.
Returns	True if successful. False if unsuccessful. The method fails, for example, if there are no rows.

MovePrevious method

Prototype	MovePrevious() As Boolean Member of UltraLiteAFLib.ULResultSet
Description	Moves to the previous row.
Returns	True if successful. False if unsuccessful. The method fails, for example, if there are no rows.

MoveRelative method

Prototype	MoveRelative(<i>index</i> As Long) As Boolean Member of UltraLiteAFLib.ULResultSet
Description	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.

IsNull method

Prototype	IsNull(<i>index</i> As Integer) As Boolean Member of UltraLiteAFLib.ULResultSet
Description	Indicates whether this column contains a null value.
Parameters	index The column index value.
Returns	True if the value is Null.

GetDatetime method

Prototype **GetDatetime(*index* As Integer)** As Date
Member of **UltraLiteAFLib.ULResultSet**

Description Gets the column value as an Date.

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

Returns The value as a Date.

GetDouble method

Prototype **GetDouble(*index* As Integer)** As Double
Member of **UltraLiteAFLib.ULResultSet**

Description Gets the column value as a Double.

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

Returns The value as a Double.

GetInteger method

Prototype **GetInteger(*index* As Integer)** As Integer
Member of **UltraLiteAFLib.ULResultSet**

Description Gets the column value as an Integer.

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

Returns The value as an Integer.

GetLong method

Prototype **GetLong(*index* As Integer)** As Long
Member of **UltraLiteAFLib.ULResultSet**

Description Gets the column value as a Long.

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

Returns The value as a Long.

GetReal method

Prototype **GetReal(*index* As Integer)** As Single
Member of **UltraLiteAFLib.ULResultSet**

Description Gets the column value as a Single.

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

Returns The value as a Real.

GetString method

Prototype **GetString(*index* As Integer)** As String
Member of **UltraLiteAFLib.ULResultSet**

Description Gets the column value as a String.

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

Returns The value as a String.

ULResultSetSchema class

The ULResultSetSchema provides information about the schema of the result set.

Properties

Prototype	Description
ColumnCount As Integer (read-only)	Gets the number of columns in the result set
ColumnName As String (read-only)	Gets the name of the column in the result set.
ColumnPrecision As Integer (read-only)	Gets the precision of the datatype for the column if it is numeric.
ColumnScale As Integer (read-only)	Gets the scale of the datatype for the column if it is numeric.
ColumnSize As Integer (read-only)	Gets the size of the datatype for the column.
ColumnSQLType As ULSQLType (read-only)	Gets the ULSQLType of the column.

ULSchemaUpgradeState enumeration

The ULSchemaUpgradeState constants identify states during the upgrade of a database schema.

Constant	Description
ulUpgradeStateStarting	The schema upgrade is starting. This is the only state during which the upgrade may be canceled. If the upgrade is canceled, you receive a second event with state ulUpgradeStateAbort.
ulUpgradeStateUpgrading	The schema upgrade is in progress.
ulUpgradeStateAbort	The schema upgrade has been canceled and the old database is preserved. This state may occur as the result of a recoverable error or user action.
ulUpgradeStateDone	The schema upgrade completed successfully.
ulUpgradeStateError	A critical error occurred and the database is unusable.

See also

- ◆ [“Upgrading UltraLite database schemas”](#) [*UltraLite Database User’s Guide*, page 54]
- ◆ [“OnSchemaUpgradeProgress event”](#) on page 95
- ◆ [“OnSchemaUpgradeStateChange event”](#) on page 96

ULSQLCode enumeration

The ULSQLCode constants identify SQL codes that may be reported by UltraLite.

☞ For a description of the errors, see the *Adaptive Server Anywhere Error Messages* book.

Constant	Value
ulSQLE_AGGREGATES_NOT_ALLOWED	-150
ulSQLE_ALIAS_NOT_UNIQUE	-830
ulSQLE_ALIAS_NOT_YET_DEFINED	-831
ulSQLE_BAD_ENCRYPTION_KEY	-840
ulSQLE_BAD_PARAM_INDEX	-689
ulSQLE_CANNOT_ACCESS_FILE	-602
ulSQLE_CANNOT_CHANGE_USER_NAME	-867
ulSQLE_CANNOT_MODIFY	-191
ulSQLE_CANNOT_EXECUTE_STMT	-111
ulSQLE_COLUMN_AMBIGUOUS	-144
ulSQLE_COLUMN_CANNOT_BE_NL	-195
ulSQLE_COLUMN_IN_INDEX	-127
ulSQLE_COLUMN_NOT_FOUND	-143
ulSQLE_COMMUNICATIONS_ERROR	-85
ulSQLE_CONNECTION_NOT_FOUND	-108
ulSQLE_CONVERSION_ERROR	-157
ulSQLE_CURSOROP_NOT_ALLOWED	-187
ulSQLE_CURSOR_ALREADY_OPEN	-172
ulSQLE_CURSOR_NOT_OPEN	-180
ulSQLE_DATABASE_ERROR	-301
ulSQLE_DATABASE_NEW	123
ulSQLE_DATABASE_NOT_CREATED	-645
ulSQLE_DATABASE_NOT_FOUND	-83

Constant	Value
ulSQLiteDatabaseUpgradeFailed	-672
ulSQLiteDatabaseUpgradeNotPossible	-673
ulSQLiteDatabaseDatatypeNotAllowed	-624
ulSQLiteDatabaseSpaceFull	-604
ulSQLiteDatabaseDivZeroError	-628
ulSQLiteDatabaseDownloadConflict	-839
ulSQLiteDatabaseDropDatabaseFailed	-651
ulSQLiteDatabaseDynamicMemoryExhausted	-78
ulSQLiteDatabaseEngineAlreadyRunning	-96
ulSQLiteDatabaseEngineNotMTIUser	-89
ulSQLiteDatabaseError	-300
ulSQLiteDatabaseErrorCallingFunction	-622
ulSQLiteDatabaseExpressionError	-156
ulSQLiteDatabaseIdentifierTooLong	-250
ulSQLiteDatabaseIndexNotFound	-183
ulSQLiteDatabaseIndexNotUnique	-196
ulSQLiteDatabaseInterrupted	-299
ulSQLiteDatabaseInvalidAggregatePlacement	-862
ulSQLiteDatabaseInvalidForeignKey	-194
ulSQLiteDatabaseInvalidForeignKeyDef	-113
ulSQLiteDatabaseInvalidGroupSelect	-149
ulSQLiteDatabaseInvalidLogon	-103
ulSQLiteDatabaseInvalidOptionSetting	-201
ulSQLiteDatabaseInvalidOrder	-152
ulSQLiteDatabaseInvalidOrderByColumn	-854
ulSQLiteDatabaseInvalidParameter	-735

Constant	Value
ulSQL_INVALID_SQL_IDENTIFIER	-760
ulSQL_INVALID_STATEMENT	-130
ulSQL_LOCKED	-210,
ulSQL_MEMORY_ERROR	-309
ulSQL_METHOD_CANNOT_BE_CALLED	-669
ulSQL_NAME_NOT_UNIQUE	-110
ulSQL_NOERR	0
ulSQL_NOTFOUND	100
ulSQL_NOT_IMPLEMENTED	-134
ulSQL_NO_CURRENT_ROW	-197
ulSQL_NO_INDICATOR	-181
ulSQL_OVERFLOW_ERROR	-158
ulSQL_PERMISSION_DENIED	-121
ulSQL_PRIMARY_KEY_NOT_UNIQUE	-193
ulSQL_PRIMARY_KEY_VALUE_REF	-198
ulSQL_PUBLICATION_NOT_FOUND	-280
ulSQL_RESOURCE_GOVERNOR_- EXCEEDED	-685
ulSQL_ROW_DROPPED_DURING_- SCHEMA_UPGRADE	130
ulSQL_SERVER_SYNCHRONIZATION_- ERROR	-857
ulSQL_START_STOP_DATABASE_DENIED	-75
ulSQL_STATEMENT_ERROR	-132
ulSQL_SYNTAX_ERROR	-131
ulSQL_STRING_RIGHT_TRUNCATION	-638
ulSQL_TABLE_HAS_PUBLICATIONS	-281
ulSQL_TABLE_IN_USE	-214
ulSQL_TABLE_NOT_FOUND	-141

Constant	Value
ulSQLE_TOO_MANY_CONNECTIONS	-102
ulSQLE_TRALITE_OBJ_CLOSED	-908
ulSQLE_UNABLE_TO_CONNECT_OR_START	-764
ulSQLE_UNABLE_TO_START_DATABASE	-82
ulSQLE_UNCOMMITTED_TRANSACTIONS	-755
ulSQLE_UNKNOWN_FUNC	-148
ulSQLE_UNKNOWN_USERID	-140
ulSQLE_UNSUPPORTED_CHARACTER_SET_ERROR	-869
ulSQLE_UPLOAD_FAILED_AT_SERVER	-794
ulSQLE_WRONG_PARAMETER_COUNT	-154

ULSQLType enumeration

ULSQLType lists the available UltraLite SQL database types used as table column types.

Constant	UltraLite Database Type	Value
ulTypeLong	Integer	0
ulTypeUnsignedLong	SmallInt	2
ulTypeShort	UnsignedInteger	1
ulTypeUnsignedShort	UnsignedSmallInt	3
ulTypeBig	Big	4
ulTypeUnsignedBig	UnsignedBig	5
ulTypeByte	Byte	6
ulTypeBit	Bit	7
ulTypeDateTime	Time	8
ulTypeDate	Date	9
ulTypeTime	Timestamp	10
ulTypeDouble	Double	11
ulTypeReal	Real	12
ulTypeNumeric	(Var)Binary	17
ulTypeBinary	LongBinary	13
ulTypeString	(Var)Char	15
ulTypeLongString	LongVarchar	16
ulTypeLongBinary	Numeric	14
ulTypeUUID	UniqueIdentifier	18

ULStreamErrorCode enumeration

The ULStreamErrorCode constants identify communications errors during synchronization.

☞ For more information about these errors, see “[MobiLink Communication Error Messages](#)” [*ASA Error Messages*, page 549].

Constant	Value
ulStreamErrorCodeNone	0
ulStreamErrorCodeParameter	1
ulStreamErrorCodeParameterNotUInt32	2
ulStreamErrorCodeParameterNotUInt32Range	3
ulStreamErrorCodeParameterNotBoolean	4
ulStreamErrorCodeParameterNotHex	5
ulStreamErrorCodeMemoryAllocation	6
ulStreamErrorCodeParse	7
ulStreamErrorCodeRead	8
ulStreamErrorCodeWrite	9
ulStreamErrorCodeEndWrite	10
ulStreamErrorCodeEndRead	11
ulStreamErrorCodeNotImplemented	12
ulStreamErrorCodeWouldBlock	13
ulStreamErrorCodeGenerateRandom	14
ulStreamErrorCodeInitRandom	15
ulStreamErrorCodeSeedRandom	16
ulStreamErrorCodeCreateRandomObject	17
ulStreamErrorCodeShuttingDown	18
ulStreamErrorCodeDequeuingConnection	19
ulStreamErrorCodeSecureCertificateRoot	20
ulStreamErrorCodeSecureCertificateCompanyName	21
ulStreamErrorCodeSecureCertificateChainLength	22

Constant	Value
ulStreamErrorCodeSecureCertificateRef	23
ulStreamErrorCodeSecureCertificateNotTrusted	24
ulStreamErrorCodeSecureDuplicateContext	25
ulStreamErrorCodeSecureSetIo	26
ulStreamErrorCodeSecureSetIoSemantics	27
ulStreamErrorCodeSecureCertificateChainFunc	28
ulStreamErrorCodeSecureCertificateChainRef	29
ulStreamErrorCodeSecureEnableNonBlocking	30
ulStreamErrorCodeSecureSetCipherSuites	31
ulStreamErrorCodeSecureSetChainNumber	32
ulStreamErrorCodeSecureCertificateFileNotFound	33
ulStreamErrorCodeSecureReadCertificate	34
ulStreamErrorCodeSecureReadPrivateKey	35
ulStreamErrorCodeSecureSetPrivateKey	36
ulStreamErrorCodeSecureCertificateExpiryDate	37
ulStreamErrorCodeSecureExportCertificate	38
ulStreamErrorCodeSecureAddCertificate	39
ulStreamErrorCodeSecureTrustedCertificateFileNotFound	40
ulStreamErrorCodeSecureTrustedCertificateRead	41
ulStreamErrorCodeSecureCertificateCount	42
ulStreamErrorCodeSecureCreateCertificate	43
ulStreamErrorCodeSecureImportCertificate	44
ulStreamErrorCodeSecureSetRandomRef	45
ulStreamErrorCodeSecureSetRandomFunc	46
ulStreamErrorCodeSecureSetProtocolSide	47
ulStreamErrorCodeSecureAddTrustedCertificate	48
ulStreamErrorCodeSecureCreatePrivateKeyObject	49
ulStreamErrorCodeSecureCertificateExpired	50

Constant	Value
ulStreamErrorCodeSecureCertificateCompanyUnit	51
ulStreamErrorCodeSecureCertificateCommonName	52
ulStreamErrorCodeSecureHandshake	53
ulStreamErrorCodeHttpVersion	54
ulStreamErrorCodeSecureSetReadFunc	55
ulStreamErrorCodeSecureSetWriteFunc	56
ulStreamErrorCodeSocketHostNameNotFound	57
ulStreamErrorCodeSocketGetHostByAddr	58
ulStreamErrorCodeSocketLocalhostNameNotFound	59
ulStreamErrorCodeSocketCreateTcpi	60
ulStreamErrorCodeSocketCreateUdp	61
ulStreamErrorCodeSocketBind	62
ulStreamErrorCodeSocketCleanup	63
ulStreamErrorCodeSocketClose	64
ulStreamErrorCodeSocketConnect	65
ulStreamErrorCodeSocketGetName	66
ulStreamErrorCodeSocketGetOption	67
ulStreamErrorCodeSocketSetOption	68
ulStreamErrorCodeSocketListen	69
ulStreamErrorCodeSocketShutdown	70
ulStreamErrorCodeSocketSelect	71
ulStreamErrorCodeSocketStartup	72
ulStreamErrorCodeSocketPortOutOfRange	73
ulStreamErrorCodeLoadNetworkLibrary	74
ulStreamErrorCodeActsycNoPort	75
ulStreamErrorCodeHttpExpectedPost	89

ULStreamErrorContext enumeration

The ULStreamErrorContext constants identify constants you can use to specify ULStreamErrorContext. The ULStreamErrorContext is the network operation performed when the stream error happens.

Constant	Value
ulStreamErrorContextUnknown	0
ulStreamErrorContextRegister	1
ulStreamErrorContextUnregister	2
ulStreamErrorContextCreate	3
ulStreamErrorContextDestroy	4
ulStreamErrorContextOpen	5
ulStreamErrorContextClose	6
ulStreamErrorContextRead	7
ulStreamErrorContextWrite	8
ulStreamErrorContextWriteFlush	9
ulStreamErrorContextEndWrite	10
ulStreamErrorContextEndRead	11
ulStreamErrorContextYield	12
ulStreamErrorContextSoftshutdown	13

ULStreamErrorID enumeration

The ULStreamErrorID is an enumeration of the possible network layers that caused an error in an unsuccessful synchronization.

Constant	Value
ulStreamErrorIDTcpip	0
ulStreamErrorIDSerial	1
ulStreamErrorIDFake	2
ulStreamErrorIDPalmConduit	3
ulStreamErrorIDPalmSs	4
ulStreamErrorIDNettech	5
ulStreamErrorIDRimbb	6
ulStreamErrorIDHttp	7
ulStreamErrorIDHttps	8
ulStreamErrorIDDhCast	9
ulStreamErrorIDSecure	10
ulStreamErrorIDCerticom	11
ulStreamErrorIDJavaCerticom	12
ulStreamErrorIDCerticomSsl	13
ulStreamErrorIDCerticomTls	14
ulStreamErrorIDWirestrm	15
ulStreamErrorIDWireless	16
ulStreamErrorIDReplay	17
ulStreamErrorIDStrm	18
ulStreamErrorIDUdp	19
ulStreamErrorIDEmail	20
ulStreamErrorIDFile	21
ulStreamErrorIDActivesync	22
ulStreamErrorIDRsaTls	23
ulStreamErrorIDJavaRsa	24

ULStreamType enumeration

The ULStreamType constants identify constants you can use to specify stream type. These represent the types of MobiLink synchronization streams you can use for synchronization.

Constant	Value	Description
ulUnknown	0	No stream type has been set. You must set a stream type before synchronization.
ulTCPIP	1	TCP/IP stream
ulHTTP	2	HTTP stream
ulHTTPS	3	HTTPS synchronization
ulPalmConduit	4	For HotSync synchronization

ULSyncParms class

The attributes set for the ULSyncParms object determine how the database synchronizes with the consolidated or desktop database. Attributes that are read-only reflect the status of the last synchronization.

Properties

The following are properties of ULSyncParms:

Prototype	Description
CheckpointStore As Boolean	<p>If true, adds checkpoints of the database during synchronization to limit database growth during the synchronization process. This is most useful for large downloads with many updates.</p> <p>See “Checkpoint Store synchronization parameter” [<i>MobiLink Clients</i>, page 318].</p>
DownloadOnly As Boolean	<p>Indicates if a synchronization only downloads data.</p> <p>See “Download Only synchronization parameter” [<i>MobiLink Clients</i>, page 320].</p>
KeepPartialDownload As Boolean	<p>If the synchronisation fails during download because of a communications error, apply those changes that were successfully downloaded, rather than rolling back all the changes.</p> <p>See “Keep Partial Download synchronization parameter” [<i>MobiLink Clients</i>, page 321].</p>
NewPassword As String	<p>Change a user password to this new password string on the next synchronization.</p> <p>See “New Password synchronization parameter” [<i>MobiLink Clients</i>, page 322].</p>
Password As String	<p>The password corresponding to a given user name.</p> <p>See “Password synchronization parameter” [<i>MobiLink Clients</i>, page 324].</p>

Prototype	Description
PingOnly As Boolean	<p>If true, check the server for liveness, but do not synchronize data.</p> <p>See “Ping synchronization parameter” [<i>MobiLink Clients</i>, page 325].</p>
PublicationMask As Long	<p>Specify the publications to synchronize. The default is to synchronize all data.</p> <p>See “Publication synchronization parameter” [<i>MobiLink Clients</i>, page 326].</p>
ResumePartialDownload As Boolean	<p>Resume a synchronization that failed during download because of a communications error, applying only those changes that were scheduled to be downloaded in the failed synchronization.</p> <p>See “Resume Partial Download synchronization parameter” [<i>MobiLink Clients</i>, page 327].</p>
SendColumnNames As Boolean	<p>If SendColumnNames is true, column names are sent to the MobiLink synchronization server. Column names must be sent to the MobiLink synchronization server for automatic script generation.</p> <p>See “Send Column Names synchronization parameter” [<i>MobiLink Clients</i>, page 330].</p>
SendDownloadAck As Boolean	<p>If SendDownloadAck is true, a download acknowledgement is sent during synchronization.</p> <p>See “Send Download Acknowledgement synchronization parameter” [<i>MobiLink Clients</i>, page 331].</p>
Stream As UStreamType constants	<p>Set the type of stream to use during synchronization.</p> <p>See “Stream Type synchronization parameter” [<i>MobiLink Clients</i>, page 332].</p>

Prototype	Description
StreamParms As String	Set network protocol options for the given stream type. See “Stream Parameters synchronization parameter” [<i>MobiLink Clients</i> , page 334] and “Network protocol options for UltraLite synchronization clients” [<i>MobiLink Clients</i> , page 341].
UploadOnly As Boolean	Indicates whether a synchronization only uploads data. See “Upload Only synchronization parameter” [<i>MobiLink Clients</i> , page 337].
UserName As String	The MobiLink user name for synchronization. See “User Name synchronization parameter” [<i>MobiLink Clients</i> , page 338].
Version As String	The synchronization script version to run. See “Version synchronization parameter” [<i>MobiLink Clients</i> , page 339].

Examples

The following example sets synchronization parameters for an UltraLite for MobileVB application.

```
With Connection.SyncParms
    .UserName = "afsample"
    .Stream = ULStreamType.ulTCPIP
    .Version = "ul_default"
    .SendColumnNames = True
End With
Connection.Synchronize
```

AddAuthenticationParm method

Prototype	AddAuthenticationParm(BSTR parm) Member of UltraLiteAFLib.ULSyncParms
Description	Adds a parameter to be passed to the authenticate_parms MobiLink synchronization script.
Parameters	parm The parameter being added.
Returns	No return value.

See also

[“Authentication Parameters synchronization parameter”](#) [*MobiLink Clients*, page 316]

[“authenticate_parameters connection event”](#) [*MobiLink Administration Guide*, page 334]

ClearAuthenticationParms method

Prototype

ClearAuthenticationParms()
Member of **UltraLiteAFLib.ULSyncParms**

Description

Clears all parameters that were to be passed to the authenticate_parms MobiLink synchronization script.

Returns

No return value.

See also

[“Authentication Parameters synchronization parameter”](#) [*MobiLink Clients*, page 316]

[“authenticate_parameters connection event”](#) [*MobiLink Administration Guide*, page 334]

ULSyncResult class

The attributes of the ULSyncResult object store the results of the last synchronization.

Properties

The following are properties of ULSyncResult:

Prototype	Description
AuthStatus As AuthStatusCode (read-only)	Gets the authorization status code for the last synchronization. See “ Authentication Status synchronization parameter ” [<i>MobiLink Clients</i> , page 317].
PartialDownloadRetained (read-only)	Indicates that the synchronization failed during download, and that a partial download was kept. See “ Partial Download Retained synchronization parameter ” [<i>MobiLink Clients</i> , page 324].
IgnoredRows As Boolean (read-only)	Indicates whether rows were ignored during the last synchronization. See “ Ignored Rows synchronization parameter ” [<i>MobiLink Clients</i> , page 320].
StreamErrorCode As ULStream- ErrorCode (read-only)	Gets the error code reported by the synchronization stream.
StreamErrorContext As UL- StreamErrorContext (read-only)	Gets the basic network operation performed.
StreamErrorID As ULStreamEr- rorID (read-only)	Gets the network layer reporting the error.
StreamErrorSystem As Long (read-only)	Gets the stream error system-specific code.
UploadOK As Boolean (read- only)	Indicates whether data was uploaded successfully in the last synchronization. See “ Version synchronization parameter ” [<i>MobiLink Clients</i> , page 339].

ULSyncState enumeration

Constant	Description
ulSyncStateStarting	No synchronization actions have been taken yet.
ulSyncStateConnecting	The synchronization stream has been built, but not yet opened.
ulSyncStateSendingHeader	The synchronization stream has been opened and the header is about to be sent.
ulSyncStateSendingTable	A table is being sent.
ulSyncStateSendingData	Data for the current table is being sent.
ulSyncStateFinishingUpload	The upload is completing. The final count of rows sent is included with this event.
ulSyncStateReceivingUploadAck	An acknowledgement that the upload is complete is being received.
ulSyncStateReceivingTable	A table is being received.
ulSyncStateReceivingData	Data for the current table is being received.
ulSyncStateCommittingDownload	The download is being committed. The final count of rows received is included with this event.
ulSyncStateSendingDownloadAck	An acknowledgement that the download is complete is being sent.
ulSyncStateDisconnecting	The synchronization stream is about to be closed.
ulSyncStateDone	Synchronization has successfully completed. The SyncResult object has been updated.
ulSyncStateError	Synchronization has completed but an error occurred. Check SyncResult and SQLCode for details.

Constant	Description
<code>ulSyncStateRollingBackDownload</code>	Synchronization is rolling back the download because an error was encountered during the download. The error will be reported with a subsequent <code>ulSyncStateError</code> progress report.
<code>ulSyncStateCancelled</code>	Synchronization has been canceled.

ULTable class

The ULTable class is used to store, remove, update, and read data from a table.

Before you can work with table data, you must call the Open method.

ULTable uses table modes for table operations:

Mode	Description
FindBegin	Begins find mode
InsertBegin	Begins insert mode
LookupBegin	Begins lookup mode
UpdateBegin	Begins update mode

Properties

Prototype	Description
BOF As Boolean (read-only)	Indicates whether the current row position is before the first row. Returns True if the current row position is before the first row, otherwise false.
EOF As Boolean (read-only)	Indicates whether the current row position is after the last row. Returns True if the current row position is before the first row, otherwise false.
IsOpen As Boolean (read-only)	Indicates whether or not the table is currently open.
RowCount As Long (read-only)	Gets the number of rows in the table.
Schema As ULTableSchema (read-only)	Gets information about the table schema.

Close method

Prototype

Close()
Member of **UltraLiteAFLib.ULTable**

Description

Frees resources associated with the table. This method should be called after

all processing involving the table is complete.

For the Palm OS, if a table is not closed it can be reopened to its current position.

Column method

Column(*name* As String) As ULColumn
Member of **UltraLiteAFLib.ULTable**

Description Returns the object for the specified column name.

For information about the **ULColumn** object, see [“ULColumn class” on page 84](#)

Parameters **name** The name of the column to return.

Returns Returns a Columns object.

Delete method

Prototype **Delete**()
Member of **UltraLiteAFLib.ULTable**

Description Deletes the current row from the table.

DeleteAllRows method

Prototype **DeleteAllRows**()
Member of **UltraLiteAFLib.ULTable**

Description Deletes all rows in the table.

In some applications, it can be useful to delete all rows from tables 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 **ULConnection.StopSynchronizationDelete** method or calling **Truncate** instead of **DeleteAllRows**.

FindBegin method

Prototype **FindBegin**()
Member of **UltraLiteAFLib.ULTable**

Description Prepares a table for a find.

FindFirst method

Prototype **FindFirst**([*num_columns* As Long = 32767]) As Boolean
Member of **UltraLiteAFLib.ULTable**

Description	<p>Move forwards through the table from the beginning, looking for a row that exactly matches a value or set of values in the current index.</p> <p>The current index is that used to specify the sort order of the table. It is specified when your application calls the Open method. The default index is the primary key.</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 (EOF).</p> <p><i>Note</i> : Requires that FindBegin be called prior to using this method.</p>
Parameters	<p>num_columns An optional parameter referring to the number of columns to be used in the FindFirst. For example, if 2 is passed, the first two columns are used for the FindFirst. If num_columns exceeds the number of columns indexed, all columns are used in FindFirst.</p>
Returns	<p>True if successful.</p> <p>False if unsuccessful.</p>

FindLast method

Prototype	<p>FindLast([<i>num_columns</i> As Long = 32767]) As Boolean Member of UltraLiteAFLib.ULTable</p>
Description	<p>Move backwards through the table from the end, looking for a row that matches a value or set of values in the current index.</p> <p>The current index is used to specify the sort order of the table. It is specified when your application calls the Open method. The default index is the primary key.</p> <p>☞ For more information, see “Open method” on page 155.</p> <p>To specify the value to search for, set the column value for each column in the index for which you want to find the value. The cursor is left on the last row found that exactly matches the index value. On failure the cursor position is before the first row (BOF).</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Note Requires that FindBegin be called prior to using this method.</p> </div>
Parameters	<p>num_columns An optional parameter referring to the number of columns to be used in the FindLast. For example, if 2 is passed, the first two columns are used for the FindLast. If num_columns exceeds the number of columns indexed, all columns are used in FindLast.</p>
Returns	<p>True if successful.</p>

False if unsuccessful.

FindNext method

Prototype	FindNext ([<i>num_columns</i> As Long = 32767]) As Boolean Member of UltraLiteAFLib.ULTable
Description	<p>Move forwards through the table from the current position, looking for the next row that exactly matches a value or set of values in the current index.</p> <p>The current index is that used to specify the sort order of the table. It is specified when your application calls the Open method. The default index is the primary key.</p> <p>☞ For more information, see “Open method” on page 155.</p> <p>The cursor is left on the first row found that exactly matches the index value. On failure, the cursor position is after the last row (EOF).</p> <p><i>Note</i> : Must be preceded by FindFirst or FindLast.</p>
Parameters	num_columns An optional parameter referring to the number of columns to be used in the FindNext. For example, if 2 is passed, the first two columns are used for the FindNext. If num_columns exceeds the number of columns indexed, all columns are used in FindNext.
Returns	<p>True if successful.</p> <p>False if unsuccessful (EOF).</p>

FindPrevious method

Prototype	FindPrevious ([<i>num_columns</i> As Long = 32767]) As Boolean Member of UltraLiteAFLib.ULTable
Description	<p>Move backwards through the table from the current position, looking for the previous row that exactly matches a value or set of values in the current index.</p> <p>The current index is that used to specify the sort order of the table. It is specified when your application calls the Open method. The default index is the primary key.</p> <p>☞ For more information, see “Open method” on page 155.</p> <p>On failure it is positioned before the first row (BOF).</p>
Parameters	num_columns An optional parameter referring to the number of columns to be used in the FindPrevious. For example, if 2 is passed, the first two columns are used for the FindPrevious. If num_columns exceeds the number of columns indexed, all columns are used in FindPrevious.

Returns **True** if successful.
False if unsuccessful (BOF).

Insert method

Prototype **Insert()** As Boolean
Member of **UltraLiteAFLib.ULTable**

Description Inserts a row in the table with values specified in previous **Set** methods. Must be preceded by **InsertBegin**. Set for each ULColumn object.

Returns **True** if successful.
False if unsuccessful (BOF).

InsertBegin method

Prototype **InsertBegin()**
Member of **UltraLiteAFLib.ULTable**

Description Prepares a table for inserting a new row, setting column values to their defaults.

Examples In this example, InsertBegin sets insert mode to allow you to begin assigning data values to CustomerTable columns.

```
CustomerTable.InsertBegin  
CustomerTable.Column("Fname").StringValue = fname  
CustomerTable.Column("Lname").StringValue = lname  
CustomerTable.Insert
```

See also [“UpdateBegin method” on page 156](#)

LookupBackward method

Prototype **LookupBackward([num_columns As Long = 32767])** As Boolean
Member of **UltraLiteAFLib.ULTable**

Description Move backwards through the table starting from the end, looking for the first row that matches or is less than a value or set of values in the current index.

The current index is that used to specify the sort order of the table. It is specified when your application calls the **Open** method. The default index is the primary key.

☞ For more information, see [“Open method” on page 155](#).

To specify the value to search for, set the column value for each column in the index. The cursor is left on the last row that matches or is less than the index value. On failure (that is, if no row is less than the value being looked


for), the cursor position is before the first row (**BOF**).

Parameters	num_columns For composite indexes, the number of columns to use in the lookup.
Returns	True if successful. False if unsuccessful.

LookupBegin method

Prototype	LookupBegin() Member of UltraLiteAFLib.ULTable
Description	Prepares a table for a lookup.

LookupForward method

Prototype	LookupForward([num_columns As Long = 32767]) As Boolean Member of UltraLiteAFLib.ULTable
Description	<p>Move forward through the table starting from the beginning, looking for the first row that matches or is greater than a value or set of values in the current index.</p> <p>The current index is that used to specify the sort order of the table. It is specified when your application calls the Open method. The default index is the primary key.</p> <p> For more information, see “Open method” on page 155.</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 matches or is greater than the index value. On failure (that is, if no rows are greater than the value being looked for), the cursor position is after the last row (EOF).</p>
Parameters	num_columns For composite indexes, the number of columns to use in the lookup.
Returns	True if successful. False if unsuccessful.

MoveAfterLast method

Prototype	MoveAfterLast() As Boolean Member of UltraLiteAFLib.ULTable
Description	Moves to a position after the last row.
Returns	True if successful.

False if the operation fails.

MoveBeforeFirst method

Prototype **MoveBeforeFirst()** As Boolean
Member of **UltraLiteAFLib.ULTable**

Description Moves to a position before the first row.

Returns **True** if successful.
False if the operation fails.

MoveFirst method

Prototype **MoveFirst()** As Boolean
Member of **UltraLiteAFLib.ULTable**

Description Moves to the first row.

Returns **True** if successful.
False if there is no data in the table.

MoveLast method

Prototype **MoveLast()** As Boolean
Member of **UltraLiteAFLib.ULTable**

Description Moves to the last row.

Returns **True** if successful.
False if there is no data in the table.

MoveNext method

Prototype **MoveNext()** As Boolean
Member of **UltraLiteAFLib.ULTable**

Description Moves to the next row.

Returns **True** if successful.
False if there is no more data in the table. For example, MoveNext fails if there are no more rows.

MovePrevious method

Prototype **MovePrevious()** As Boolean
Member of **UltraLiteAFLib.ULTable**

Description Moves to the previous row.

Returns **True** if successful.

False if there is no more data in the table. For example, `MovePrevious` fails if there are no rows.

MoveRelative method

Prototype **MoveRelative(*index* As Long)** As Boolean
Member of **UltraLiteAFLib.ULTable**

Description Moves a certain number of rows relative to the current row.

Parameters **index** The number of rows to move. The value can be positive, negative, or zero. Zero is useful if you want to repopulate a row buffer.

Returns **True** if successful.

False if the move failed, as may happen, for example, if the cursor is positioned beyond the first or last row.

Open method

Prototype **Open(**
 [*index_name* As String], _
 [*persistent_name* As String] _
)
Member of **UltraLiteAFLib.ULTable**

Description Opens the table so it can be read or manipulated. By default, the rows are ordered by primary key. By supplying an index name, the rows can be ordered in other ways.

The cursor is positioned before the first row in the table.

Parameters **index_name** An optional parameter referring to the name of the index.

persistent_name For Palm Computing Platform applications, an optional parameter referring to the stored name of the table.

Truncate method

Prototype **Truncate()**
Member of **UltraLiteAFLib.ULTable**

Description Removes all data from this table. The changes are not synchronized, so that on synchronization, it does not affect the data in the consolidated database.

☞ For more information, see [“StartSynchronizationDelete method” on page 99](#).

Update method

Prototype	Update() Member of UltraLiteAFLib.ULTable
Description	Updates a row in the table with values specified in ULColumn methods. <i>Note</i> : Must be preceded by a call to UpdateBegin.

UpdateBegin method

Prototype	UpdateBegin() Member of UltraLiteAFLib.ULTable
Description	Prepares a table for modifying the contents of the current row.
Example	<pre>CustomerTable.UpdateBegin CustomerTable.Column("Fname").StringValue = fname ' ... CustomerTable.Update</pre>

ULTableSchema class

The ULTableSchema object allows you to obtain the attributes of a table.

Properties

The ULTableSchema represents metadata about the table. The following are properties of the ULTableSchema class:

Prototype	Description
ColumnCount As Integer (read-only)	The number of columns in this table
IndexCount As Integer (read-only)	The number of indexes on this table
Name As String (read-only)	This table's name
NeverSynchronized As Boolean (read-only)	Indicates if the table is always excluded from synchronization.
PrimaryKey As ULIndexSchema (read-only)	The primary key for this table.
UploadUnchangedRows As Boolean (read-only)	Indicates if all rows in the table should be uploaded on synchronization, rather than just the rows changed since the last synchronization.

GetColumnName method

Prototype	GetColumnName(<i>id</i> As Integer) As String Member of UltraLiteAFLib.ULTableSchema
Description	Returns the name of the column that corresponds to the <i>id</i> value you supply. The ColumnCount property returns the number of columns in the table. Each column has a unique number from 1 to the ColumnCount value, where 1 is the first column in the table, 2 is the second column in the table, and so on.
Parameters	id The id of the column.
Returns	The name of a column.

GetIndex method

Prototype	GetIndex(<i>name</i> As String) As ULIndexSchema Member of UltraLiteAFLib.ULTableSchema
Description	Returns the ULIndexSchema object for the specified index.

☞ For information about the `ULIndexSchema` object, see [“ULIndexSchema class” on page 114](#).

Parameters **name** The name of the index.

Returns Returns a schema object for a given index on the table.

GetIndexName method

Prototype **GetIndexName(*id* As Integer) As String**
Member of **UltraLiteAFLib.ULTableSchema**

Description Returns the name of the index in the table that corresponds to the *id* value you supply. The `IndexCount` property returns the number of indexes in the table. Each index has a unique number from 1 to the `IndexCount` value, where 1 is the first index in the table, 2 is the second index in the table, and so on.

Parameters **name** The id of the index.

Returns Returns the name of the index.

InPublication method

Prototype **InPublication(*publicationName* As String) As Boolean**
Member of **UltraLiteAFLib.ULTableSchema**

Description Indicates whether this table is part of the specified publication.

Parameters **publicationName** The name of the publication you are checking.

Returns **True** if the table is part of the publication.
False if the table is not part of the publication.

Index

A

accessing schema information		
UltraLite for MobileVB	29	
AddAuthenticationParm method (ULSyncParms class)		
UltraLite for MobileVB API	143	
API reference		
UltraLite for MobileVB	81	
APIs		
UltraLite for MobileVB	81	
AppendByteChunk method (ULColumn class)		
UltraLite for MobileVB API	85	
AppendByteChunkParameter method (ULPreparedStatement class)		
UltraLite for MobileVB API	116	
AppendStringChunk method (ULColumn class)		
UltraLite for MobileVB API	86	
AppendStringChunkParameter method (ULColumn class)		
UltraLite for MobileVB API	117	
AppForge Booster		
MobileVB	2	
AppForge Crossfire		
adding references	7	
AppForge MobileVB		
adding references	6	
AppForge Booster	2	
UltraLite	2	
ApplyFile method		
UltraLite for MobileVB	9	
ApplyFile method (ULDatabaseSchema class)		
UltraLite for MobileVB API	112	
ApplyFileWithParms method		
UltraLite for MobileVB	9	
ApplyFileWithParms method (ULDatabaseSchema class)		
UltraLite for MobileVB API	112	
architecture		
UltraLite for MobileVB	3	
AuthStatus property (ULSyncResult class)		
UltraLite for MobileVB	145	
AutoCommit mode		
UltraLite for MobileVB	28	
AutoCommit property (ULConnection class)		
UltraLite for MobileVB API	91	
AutoIncrement property (ULColumnSchema class)		
UltraLite for MobileVB API	90	
AutoIncrement property (ULConnectionParms class)		
UltraLite for MobileVB API	102	

B

BLOBs		
GetByteChunk method in UltraLite for MobileVB	27	
UltraLite for MobileVB	27	
BOF property (ULTable class)		
UltraLite for MobileVB API	148	
BooleanValue property (ULColumn class)		
UltraLite for MobileVB API	84	
ByteValue property (ULColumn class)		
UltraLite for MobileVB API	84	

C

CancelSynchronize method (ULConnection class)		
UltraLite for MobileVB API	92	
casting		
data types in UltraLite for MobileVB	23	
ChangeEncryptionKey method (ULConnection class)		
UltraLite for MobileVB API	92	
CheckpointStore property (ULSyncParms class)		
UltraLite for MobileVB	141	
ClearAuthenticationParms method (ULSyncParms class)		
UltraLite for MobileVB API	144	
Close method (ULConnection class)		
UltraLite for MobileVB API	93	
Close method (ULPreparedStatement class)		
UltraLite for MobileVB API	117	
Close method (ULResultSet class)		
UltraLite for MobileVB API	122	
Close method (ULTable class)		
UltraLite for MobileVB API	148	

CodeXchange			
downloadable samples	79		
CollationName property (ULConnection class)			
UltraLite for MobileVB API	91		
Column method (ULTable class)			
UltraLite for MobileVB API	149		
ColumnCount property (ULIndexSchema class)			
UltraLite for MobileVB API	114		
ColumnCount property (ULTableSchema class)			
UltraLite for MobileVB API	157		
columns			
accessing schema information in UltraLite for MobileVB	29		
Columns collection			
UltraLite for MobileVB	21		
Commit method			
UltraLite for MobileVB	28		
Commit method (ULConnection class)			
UltraLite for MobileVB API	93		
commits			
UltraLite for MobileVB	28		
connecting			
UltraLite for MobileVB databases	11		
ContainsTable method (ULPublicationSchema class)			
UltraLite for MobileVB API	121		
conventions			
documentation	viii		
CountUploadRows method (ULConnection class)			
UltraLite for MobileVB API	93		
CreateDatabase method (ULDatabaseManager class)			
UltraLite for MobileVB API	105		
CreateDatabaseWithParms method (ULDatabaseManager class)			
UltraLite for MobileVB API	107		
D			
data manipulation			
dynamic SQL in UltraLite for MobileVB	15		
table API in UltraLite for MobileVB	21		
UltraLite for MobileVB	15		
data types			
accessing in UltraLite for MobileVB	22		
casting in UltraLite for MobileVB	23		
database schemas			
accessing in UltraLite for MobileVB	29		
UltraLite for MobileVB	9		
database state:			
maintaining on Palm OS with UltraLite for MobileVB	38		
DatabaseID property (ULConnection class)			
UltraLite for MobileVB API	91		
DatabaseNew property (ULConnection class)			
UltraLite for MobileVB API	91		
databases			
accessing schema information in UltraLite for MobileVB	29		
connecting in UltraLite for MobileVB	11		
DateFormat property (ULDatabaseSchema class)			
UltraLite for MobileVB API	111		
DateOrder property (ULDatabaseSchema class)			
UltraLite for MobileVB API	111		
DatetimeValue property (ULColumn class)			
UltraLite for MobileVB API	84		
DefaultValue property (ULColumnSchema class)			
UltraLite for MobileVB API	90		
Delete method (ULTable class)			
UltraLite for MobileVB API	149		
DeleteAllRows method (ULTable class)			
UltraLite for MobileVB API	149		
deleting			
rows in UltraLite for MobileVB	24		
deploying			
UltraLite applications	35		
UltraLite for MobileVB applications	35, 36		
development			
UltraLite for MobileVB	5		
development platforms			
UltraLite for MobileVB	2		
DML operations			
UltraLite for MobileVB	15		
documentation			
conventions	viii		
SQL Anywhere Studio	vi		
DoubleValue property (ULColumn class)			
UltraLite for MobileVB API	84		
DownloadOnly property (ULSyncParms class)			
UltraLite for MobileVB	141		
DropDatabase method (ULDatabaseManager class)			
UltraLite for MobileVB API	107		
DropDatabaseWithParms method (ULDatabaseManager class)			
UltraLite for MobileVB API	108		
dynamic SQL			
UltraLite for MobileVB development	15		

- E**
- encryption
 - UltraLite for MobileVB development 14
 - EOF property (ULTable class)
 - UltraLite for MobileVB API 148
 - error handling
 - UltraLite for MobileVB 30
 - errors
 - handling in UltraLite for MobileVB 30
 - ExecuteQuery method (ULPreparedStatement class)
 - UltraLite for MobileVB API 117
 - ExecuteStatement method (ULPreparedStatement class)
 - UltraLite for MobileVB API 117
- F**
- features
 - for MobileVB 2
 - feedback
 - documentation xii
 - providing xii
 - find methods
 - UltraLite for MobileVB 23
 - find mode
 - UltraLite for MobileVB 25
 - FindBegin method (ULTable class)
 - UltraLite for MobileVB API 149
 - FindFirst method (ULTable class)
 - UltraLite for MobileVB API 149
 - FindLast method (ULTable class)
 - UltraLite for MobileVB API 150
 - FindNext method (ULTable class)
 - UltraLite for MobileVB API 151
 - FindPrevious method (ULTable class)
 - UltraLite for MobileVB API 151
 - ForeignKey property (ULIndexSchema class)
 - UltraLite for MobileVB API 114
- G**
- GetByteChunk method
 - UltraLite for MobileVB 27
 - GetByteChunk method (ULColumn class)
 - UltraLite for MobileVB API 86
 - GetByteChunk method (ULResultSet class)
 - UltraLite for MobileVB API 122
 - GetColumnName method (ULIndexSchema class)
 - UltraLite for MobileVB API 115
 - GetColumnName method (ULTableSchema class)
 - UltraLite for MobileVB API 157
 - GetDatetime method (ULResultSet class)
 - UltraLite for MobileVB API 126
 - GetDouble method (ULResultSet class)
 - UltraLite for MobileVB API 126
 - GetIndex method (ULTableSchema class)
 - UltraLite for MobileVB API 157
 - GetIndexName method (ULTableSchema class)
 - UltraLite for MobileVB API 158
 - GetInteger method (ULResultSet class)
 - UltraLite for MobileVB API 126
 - GetLong method (ULResultSet class)
 - UltraLite for MobileVB API 126
 - GetNewUUID method (ULConnection class)
 - UltraLite for MobileVB API 94
 - GetPublicationName method (ULDatabaseSchema class)
 - UltraLite for MobileVB API 112
 - GetPublicationSchema method (ULDatabaseSchema class)
 - UltraLite for MobileVB API 113
 - GetReal method (ULResultSet class)
 - UltraLite for MobileVB API 126
 - GetString method (ULResultSet class)
 - UltraLite for MobileVB API 127
 - GetStringChunk method (ULColumn class)
 - UltraLite for MobileVB API 87
 - GetStringChunk method (ULResultSet class)
 - UltraLite for MobileVB API 123
 - GetTable function (ULConnection class)
 - UltraLite for MobileVB API 94
 - GetTableName method (ULDatabaseSchema class)
 - UltraLite for MobileVB API 113
 - GlobalAutoIncrement property (ULColumnSchema class)
 - UltraLite for MobileVB API 90
 - GlobalAutoIncrementUsage property (ULConnection class)
 - UltraLite for MobileVB API 91
 - grantConnectTo method
 - UltraLite for MobileVB 31
 - GrantConnectTo method (ULConnection class)
 - UltraLite for MobileVB API 94

I	
iAnywhere.UltraLiteForAppForge	
UltraLite development with Crossfire	7
icons	
used in manuals	x
ID property (ULColumnSchema class)	
UltraLite for MobileVB API	90
IgnoredRows property (ULSyncResult class)	
UltraLite for MobileVB	145
IndexCount property (ULTableSchema class)	
UltraLite for MobileVB API	157
indexes	
accessing schema information in UltraLite for MobileVB	29
InPublication method (ULTableSchema class)	
UltraLite for MobileVB API	158
Insert method (ULTable class)	
UltraLite for MobileVB API	152
insert mode	
UltraLite for MobileVB	25
InsertBegin method (ULTable class)	
UltraLite for MobileVB API	152
inserting	
rows in UltraLite for MobileVB	24
IntegerValue property (ULColumn class)	
UltraLite for MobileVB API	84
IsCaseSensitive property (ULConnection class)	
UltraLite for MobileVB API	91
IsColumnDescending method (ULIndexSchema class)	
UltraLite for MobileVB API	115
IsNull method (ULResultSet class)	
UltraLite for MobileVB API	125
IsNull property (ULColumn class)	
UltraLite for MobileVB API	84
IsOpen property (ULTable class)	
UltraLite for MobileVB API	148
K	
KeepPartialDownload property (ULSyncParams class)	
UltraLite for MobileVB	141
L	
LastDownloadTime method (ULConnection class)	
UltraLite for MobileVB API	94
LastIdentity property (ULConnection class)	
UltraLite for MobileVB API	91
library functions	
RollbackPartialDownload (UltraLite for MobileVB API)	98
LongValue property (ULColumn class)	
UltraLite for MobileVB API	84
lookup methods	
UltraLite for MobileVB	23
lookup mode	
UltraLite for MobileVB	25
LookupBackward method (ULTable class)	
UltraLite for MobileVB API	152
LookupBegin method (ULTable class)	
UltraLite for MobileVB API	153
LookupForward method (ULTable class)	
UltraLite for MobileVB API	153
M	
Mask property (ULPublicationSchema class)	
UltraLite for MobileVB	121
Mask property (ULResultSet class)	
UltraLite for MobileVB	122
Mask property (ULResultSetSchema class)	
UltraLite for MobileVB API	128
MobileVB	<i>see</i> AppForge MobileVB
modes	
UltraLite for MobileVB	25
MoveAfterLast method (ULResultSet class)	
UltraLite for MobileVB API	124
MoveAfterLast method (ULTable class)	
UltraLite for MobileVB API	153
MoveBeforeFirst method (ULResultSet class)	
UltraLite for MobileVB API	124
MoveBeforeFirst method (ULTable class)	
UltraLite for MobileVB API	154
MoveFirst method	
UltraLite for MobileVB	21
UltraLite for MobileVB development	17
MoveFirst method (ULResultSet class)	
UltraLite for MobileVB API	124
MoveFirst method (ULTable class)	
UltraLite for MobileVB API	154
MoveLast method (ULResultSet class)	
UltraLite for MobileVB API	124
MoveLast method (ULTable class)	
UltraLite for MobileVB API	154

-
- | | | | |
|--|-----|--|-----|
| MoveNext method | | UltraLite for MobileVB | 3 |
| UltraLite for MobileVB | 21 | OnReceive event (ULConnection class) | |
| UltraLite for MobileVB development | 17 | UltraLite for MobileVB API | 95 |
| MoveNext method (ULResultSet class) | | OnSchemaUpgradeProgress event (ULConnection class) | |
| UltraLite for MobileVB API | 125 | UltraLite for MobileVB API | 95 |
| MoveNext method (ULTable class) | | OnSchemaUpgradeStateChange event (ULConnection class) | |
| UltraLite for MobileVB API | 154 | UltraLite for MobileVB API | 96 |
| MovePrevious method (ULResultSet class) | | OnSend event (ULConnection class) | |
| UltraLite for MobileVB API | 125 | UltraLite for MobileVB API | 96 |
| MovePrevious method (ULTable class) | | OnStateChange event (ULConnection class) | |
| UltraLite for MobileVB API | 154 | UltraLite for MobileVB API | 97 |
| MoveRelative method (ULResultSet class) | | OnTableChange event (ULConnection class) | |
| UltraLite for MobileVB API | 125 | UltraLite for MobileVB API | 97 |
| MoveRelative method (ULTable class) | | Open method | |
| UltraLite for MobileVB API | 155 | ULTable object in MobileVB | 21 |
| | | ULTable object in UltraLite for MobileVB | 17 |
| N | | Open method (ULTable class) | |
| Name property (ULColumnSchema class) | | UltraLite for MobileVB API | 155 |
| UltraLite for MobileVB API | 90 | OpenByIndex method | |
| Name property (ULIndexSchema class) | | ULTable object in UltraLite for MobileVB | 17 |
| UltraLite for MobileVB API | 114 | OpenConnection method (ULDatabaseManager class) | |
| Name property (ULPublicationSchema class) | | UltraLite for MobileVB API | 108 |
| UltraLite for MobileVB | 121 | OpenConnectionWithparms method (ULDatabaseManager class) | |
| Name property (ULResultSet class) | | UltraLite for MobileVB API | 109 |
| UltraLite for MobileVB | 122 | OpenParms property (ULConnection class) | |
| Name property (ULResultSetSchema class) | | UltraLite for MobileVB API | 91 |
| UltraLite for MobileVB API | 128 | OptimalIndex property (ULColumnSchema class) | |
| Name property (ULTableSchema class) | | UltraLite for MobileVB API | 90 |
| UltraLite for MobileVB API | 157 | | |
| NearestCentury property (ULDatabaseSchema class) | | P | |
| UltraLite for MobileVB API | 111 | Palm Computing Platform | |
| network protocol options | | target platform in UltraLite for MobileVB | 2 |
| UltraLite for MobileVB | 141 | Palm OS | |
| NeverSynchronized property (ULTableSchema class) | | maintaining state with UltraLite for MobileVB | 38 |
| UltraLite for MobileVB API | 157 | UltraLite for MobileVB example | 39 |
| NewPassword property (ULSyncParms class) | | PartialDownloadRetained property (ULSyncResult class) | |
| UltraLite for MobileVB | 141 | UltraLite for MobileVB | 145 |
| newsgroups | | Password property (ULSyncParms class) | |
| technical support | xii | UltraLite for MobileVB | 141 |
| Nullable property (ULColumnSchema class) | | passwords | |
| UltraLite for MobileVB API | 90 | authentication in UltraLite for MobileVB | 31 |
| | | persistent name | |
| O | | | |
| obfuscation | | | |
| UltraLite for MobileVB | 14 | | |
| object hierarchy | | | |

UltraLite for MobileVB example	39	restartable downloads	
persistent name:		UltraLite for MobileVB API	98
maintaining	38	ResumePartialDownload property (ULSyncParms class)	
using	38	UltraLite for MobileVB	141
using with UltraLite for MobileVB on Palm OS38		RevokeConnectFrom method (ULConnection class)	
PingOnly property (ULSyncParms class)		UltraLite for MobileVB API	98
UltraLite for MobileVB	141	revokeConnectionFrom method	
platforms		UltraLite for MobileVB	31
supported in UltraLite for MobileVB	2	Rollback method	
Precision property (ULColumnSchema class)		UltraLite for MobileVB	28
UltraLite for MobileVB API	90	Rollback method (ULConnection class)	
Precision property (ULDatabaseSchema class)		UltraLite for MobileVB API	98
UltraLite for MobileVB API	111	RollbackPartialDownload method (ULConnection class)	
prepared statements		UltraLite for MobileVB API	98
UltraLite for MobileVB	15	rollbacks	
PrepareStatement method (ULConnection class)		UltraLite for MobileVB	28
UltraLite for MobileVB API	97	RowCount property (ULTable class)	
preparing to work with UltraLite for MobileVB		UltraLite for MobileVB API	148
about	6	rows	
PrimaryKey property (ULIndexSchema class)		accessing values in UltraLite for MobileVB	22
UltraLite for MobileVB API	114		
PrimaryKey property (ULTableSchema class)		S	
UltraLite for MobileVB API	157	samples	
projects		CodeXchange	79
creating in AppForge Crossfire	63	Scale property (ULColumnSchema class)	
creating in UltraLite for MobileVB	43	UltraLite for MobileVB API	90
PublicationCount property (ULDatabaseSchema class)		schema changes	
UltraLite for MobileVB API	111	UltraLite for MobileVB databases	9
PublicationMask property (ULSyncParms class)		schema files	
UltraLite for MobileVB	141	creating in UltraLite for MobileVB	9
publications		UltraLite for MobileVB	9
accessing schema information in UltraLite for MobileVB	29	Schema property (ULColumn class)	
		UltraLite for MobileVB API	84
R		Schema property (ULConnection class)	
RealValue property (ULColumn class)		UltraLite for MobileVB API	91
UltraLite for MobileVB API	84	Schema property (ULTable class)	
ReferencedIndexName property (ULIndexSchema class)		UltraLite for MobileVB API	148
UltraLite for MobileVB API	114	schema upgrades	
ReferencedTableName property (ULIndexSchema class)		UltraLite for MobileVB databases	9
UltraLite for MobileVB API	114	schemas	
ResetLastDownloadTime method (ULConnection class)		UltraLite for MobileVB	9, 29
UltraLite for MobileVB API	98	scrolling	
		UltraLite for MobileVB	21
		SELECT statement	
		UltraLite MobileVB development	17

SendColumnNames property (ULSyncParms class)		UltraLite for MobileVB API	91
UltraLite for MobileVB	141	SQLType property (ULColumnSchema class)	
SendDownloadAck property (ULSyncParms class)		UltraLite for MobileVB API	90
UltraLite for MobileVB	141	StartSynchronizationDelete method (ULConnection class)	
SetBooleanParameter method		UltraLite for MobileVB API	99
(ULPreparedStatement class)		StopSynchronizationDelete method (ULConnection class)	
UltraLite for MobileVB API	118	UltraLite for MobileVB API	99
SetByteChunk method (ULColumn class)		Stream property (ULSyncParms class)	
UltraLite for MobileVB API	88	UltraLite for MobileVB	141
SetByteChunkParameter method		StreamErrorContext property (ULSyncResult class)	
(ULPreparedStatement class)		UltraLite for MobileVB	145
UltraLite for MobileVB API	118	StreamErrorID property (ULSyncResult class)	
SetByteParameter method (ULPreparedStatement class)		UltraLite for MobileVB	145
UltraLite for MobileVB API	118	StreamErrorSystem property (ULSyncResult class)	
SetDateTimeParameter method		UltraLite for MobileVB	145
(ULPreparedStatement class)		StreamParms property (ULSyncParms class)	
UltraLite for MobileVB API	119	UltraLite for MobileVB	141
SetDoubleParameter method (ULPreparedStatement class)		StringToUUID method (ULConnection class)	
UltraLite for MobileVB API	119	UltraLite for MobileVB API	99
SetIntegerParameter method (ULPreparedStatement class)		StringValue property (ULColumn class)	
UltraLite for MobileVB API	119	UltraLite for MobileVB API	84
SetLongParameter method (ULPreparedStatement class)		support	
UltraLite for MobileVB API	119	newsgroups	xii
SetNull method (ULColumn class)		supported platforms	
UltraLite for MobileVB API	88	UltraLite for MobileVB	2
SetNullParameter method (ULPreparedStatement class)		synchronization	
UltraLite for MobileVB API	120	HTTP in UltraLite for MobileVB	32
SetRealParameter method (ULPreparedStatement class)		monitoring in UltraLite for MobileVB	32
UltraLite for MobileVB API	120	TCP/IP in UltraLite for MobileVB	32
SetStringParameter method (ULPreparedStatement class)		template in UltraLite for MobileVB	32
UltraLite for MobileVB API	120	UltraLite for MobileVB development	32
SetToDefault method (ULColumn class)		writing code in UltraLite for MobileVB	33
UltraLite for MobileVB API	88	Synchronize method (ULConnection class)	
Signature property (ULDatabaseSchema class)		UltraLite for MobileVB API	100
UltraLite for MobileVB API	111	synchronizing UltraLite applications	
Size property (ULColumnSchema class)		MobileVB development	32
UltraLite for MobileVB API	90	SyncParms property (ULConnection class)	
SQL Anywhere Studio		UltraLite for MobileVB API	91
documentation	vi	SyncResult property (ULConnection class)	
SQLErrorOffset property (ULConnection class)		UltraLite for MobileVB API	91
		T	
		TableCount property (ULDatabaseSchema class)	
		UltraLite for MobileVB API	111
		tables	

UltraLite databases		Close method (ULConnection class)	93
connecting in UltraLite for MobileVB	11	Close method (ULPreparedStatement class)	117
UltraLite for MobileVB		Close method (ULResultSet class)	122
about	1	Close method (ULTable class)	148
accessing schema information	29	Column method (ULTable class)	149
API reference	81	Commit method (ULConnection class)	93
architecture	3	ContainsTable method (ULPublicationSchema class)	121
connecting to UltraLite databases	11	CountUploadRows method (ULConnection class)	93
data manipulation using dynamic SQL	15	CreateDatabase method (ULDatabaseManager class)	105
data manipulation with Table API	21	CreateDatabaseWithParms method (ULDatabaseManager class)	107
deploying applications	35, 36	Delete method (ULTable class)	149
development	5	DeleteAllRows method (ULTable class)	149
encryption	14	DropDatabase method (ULDatabaseManager class)	107
Error handling	30	DropDatabaseWithParms method (ULDatabaseManager class)	108
features	2	ExecuteQuery method (ULPreparedStatement class)	117
FindFirst method (ULTable class)	149	ExecuteStatement method (ULPreparedStatement class)	117
GetNewUUID method (ULConnection class)	94	FindBegin method (ULTable class)	149
object hierarchy	3	FindLast method (ULTable class)	150
preparing to work with	6	FindNext method (ULTable class)	151
project architecture	43, 63	FindPrevious method (ULTable class)	151
supported platforms	2	GetByteChunk method (ULColumn class)	86
synchronizing UltraLite applications	32	GetByteChunk method (ULResultSet class)	122
tutorial	41	GetColumnName method (ULIndexSchema class)	115
tutorial (Crossfire)	61	GetColumnName method (ULTableSchema class)	157
ULStreamErrorCode enumeration	135	GetDatetime method (ULResultSet class)	126
User authentication	31	GetDouble method (ULResultSet class)	126
UltraLite for MobileVB API		GetIndex method (ULTableSchema class)	157
about	81	GetIndexName method (ULTableSchema class)	158
AddAuthenticationParm method (ULSyncParms class)	143	GetInteger method (ULResultSet class)	126
API reference	81	GetLong method (ULResultSet class)	126
AppendByteChunk method (ULColumn class)	85	GetPublicationName method (ULDatabaseSchema class)	112
AppendByteChunkParameter method (ULPreparedStatement class)	116	GetPublicationSchema method (ULDatabaseSchema class)	113
AppendStringChunk method (ULColumn class)	86	GetReal method (ULResultSet class)	126
AppendStringChunkParameter method (ULColumn class)	117	GetString method (ULResultSet class)	127
ApplyFile method (ULDatabaseSchema class)	112		
ApplyFileWithParms method (ULDatabaseSchema class)	112		
CancelSynchronize method (ULConnection class)	92		
ChangeEncryptionKey method (ULConnection class)	92		
ClearAuthenticationParms method (ULSyncParms class)	144		

GetStringChunk method (ULColumn class)	87	RollbackPartialDownload method	
GetStringChunk method (ULResultSet class)	123	(ULConnection class)	98
GetTable function (ULConnection class)	94	SetBooleanParameter method	
GetTableName method (ULDatabaseSchema class)	113	(ULPreparedStatement class)	118
GrantConnectTo method (ULConnection class)	94	SetByteChunk method (ULColumn class)	88
InPublication method (ULTableSchema class)	158	SetByteChunkParameter method	
Insert method (ULTable class)	152	(ULPreparedStatement class)	118
InsertBegin method (ULTable class)	152	SetByteParameter method (ULPreparedStatement class)	118
IsColumnDescending method (ULIndexSchema class)	115	SetDatetimeParameter method	
IsNull method (ULResultSet class)	125	(ULPreparedStatement class)	119
LookupBackward method (ULTable class)	152	SetDoubleParameter method	
LookupBegin method (ULTable class)	153	(ULPreparedStatement class)	119
LookupForward method (ULTable class)	153	SetIntegerParameter method	
MoveAfterLast method (ULResultSet class)	124	(ULPreparedStatement class)	119
MoveAfterLast method (ULTable class)	153	SetLongParameter method	
MoveBeforeFirst method (ULResultSet class)	124	(ULPreparedStatement class)	119
MoveBeforeFirst method (ULTable class)	154	SetNull method (ULColumn class)	88
MoveFirst method (ULResultSet class)	124	SetNullParameter method (ULPreparedStatement class)	120
MoveFirst method (ULTable class)	154	SetRealParameter method (ULPreparedStatement class)	120
MoveLast method (ULResultSet class)	124	SetToDefault method (ULColumn class)	88
MoveLast method (ULTable class)	154	StartSynchronizationDelete method	
MoveNext method (ULResultSet class)	125	(ULConnection class)	99
MoveNext method (ULTable class)	154	StopSynchronizationDelete method	
MovePrevious method (ULResultSet class)	125	(ULConnection class)	99
MovePrevious method (ULTable class)	154	StringToUUID method (ULConnection class)	99
MoveRelative method (ULResultSet class)	125	Synchronize method (ULConnection class)	100
MoveRelative method (ULTable class)	155	Truncate method (ULTable class)	155
OnReceive event (ULConnection class)	95	ULAuthStatusCode constants	83
OnSchemaUpgradeProgress event		ULColumnSchema class	90
(ULConnection class)	95	ULConnection class	91
OnSchemaUpgradeStateChange event		ULConnectionParms class	102
(ULConnection class)	96	ULDatabaseManager class	105
OnSend event (ULConnection class)	96	ULDatabaseSchema class	111
OnStateChange event (ULConnection class)	97	ULIndexSchema class	114
OnTableChange event (ULConnection class)	97	ULPreparedStatement class	116
Open method (ULTable class)	155	ULPublicationSchema class	121
OpenConnection method (ULDatabaseManager class)	108	ULResultSet class	122
OpenConnectionWithparms method		ULResultSetSchema class	128
(ULDatabaseManager class)	109	ULSchemaUpgradeState enumeration	129
PrepareStatement method (ULConnection class)	97	ULSQLCode constants	130
ResetLastDownloadTime method (ULConnection class)	98	ULSQLType constants	134
Rollback method (ULConnection class)	98	ULStreamErrorContext enumeration	138
		ULStreamErrorID constants	139
		ULStreamType enumeration	140

ULSyncParms class	141	UUIDValue property (ULColumn class)	
ULSyncResult class	145	UltraLite for MobileVB API	84
ULSyncState enumeration	146	V	
ULTable class	148	values	
ULTableSchema class	157	accessing in UltraLite for MobileVB	22
Update method (ULTable class)	156	Version property (ULDatabaseManager class)	
UpdateBegin method (ULTable class)	156	UltraLite for MobileVB API	105
UUIDToString method (ULConnection class)	100	Version property (ULSyncParms class)	
UltraLite for MobileVB API API		UltraLite for MobileVB	141
ULColumn class	84	Visual Basic	
UltraLite for MobileVB API reference		supported versions in UltraLite for MobileVB	2
alphabetic listing	81	Visual Basic programming language	
UltraLite for MobileVB APISetStringParameter		UltraLite for MobileVB	81
method (ULPreparedStatement class)		W	
SetStringParameter method		Windows CE	
(ULPreparedStatement class)	120	target platform in UltraLite for MobileVB	2
UniqueIndex property (ULIndexSchema class)			
UltraLite for MobileVB API	114		
UniqueKey property (ULIndexSchema class)			
UltraLite for MobileVB API	114		
Update method (ULTable class)			
UltraLite for MobileVB API	156		
update mode			
UltraLite for MobileVB	25		
UpdateBegin method (ULTable class)			
UltraLite for MobileVB API	156		
updating			
rows UltraLite for MobileVB	24		
UploadOK property (ULSyncResult class)			
UltraLite for MobileVB	145		
UploadOnly property (ULSyncParms class)			
UltraLite for MobileVB	141		
user authentication			
UltraLite for MobileVB	31		
UserName property (ULSyncParms class)			
UltraLite for MobileVB	141		
users			
authentication in UltraLite for MobileVB	31		
usm files			
creating in UltraLite for MobileVB	9		
UltraLite for MobileVB	9		
UUIDs			
getting as string in UltraLite for MobileVB API	94		
StringToUUID method	99		
UUIDToString method	100		
UUIDToString method (ULConnection class)			
UltraLite for MobileVB API	100		