



# UltraLite™ Database User's Guide

Part number: DC37121-01-0902-01  
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

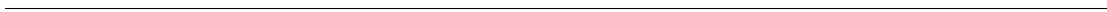
<b>About This Manual</b>	<b>vii</b>
SQL Anywhere Studio documentation . . . . .	viii
Documentation conventions . . . . .	xi
The CustDB sample database . . . . .	xiii
Finding out more and providing feedback . . . . .	xiv
<b>I UltraLite Databases</b>	<b>1</b>
<b>1 Welcome to UltraLite</b>	<b>3</b>
Introduction to UltraLite . . . . .	4
Choosing an UltraLite programming interface . . . . .	10
<b>2 Tutorial: The CustDB Sample UltraLite Application</b>	<b>15</b>
Introduction . . . . .	16
Lesson 1: Start the MobiLink synchronization server . . . . .	19
Lesson 2: Start the sample application and synchronize . . . . .	20
Lesson 3: Add an order . . . . .	21
Lesson 4: Approve or deny an existing order . . . . .	22
Lesson 5: Synchronize your changes . . . . .	23
Lesson 6: Browse the consolidated database . . . . .	25
<b>3 UltraLite Databases</b>	<b>27</b>
Creating UltraLite databases and schemas . . . . .	28
Setting UltraLite database properties . . . . .	33
User authentication in UltraLite . . . . .	40
Character sets in UltraLite . . . . .	43
UltraLite database internals . . . . .	47
UltraLite database limitations . . . . .	50
Upgrading UltraLite database schemas . . . . .	54
The UltraLite runtime . . . . .	58
<b>4 Connection Parameters</b>	<b>63</b>
Overview . . . . .	64
Database Identification parameters . . . . .	68
Open Connection parameters . . . . .	73
Database Schema parameters . . . . .	78
Additional connection parameters . . . . .	82

---

<b>5</b>	<b>UltraLite Utilities Reference</b>	<b>87</b>
	The UltraLite engine . . . . .	88
	The UltraLite Generator . . . . .	89
	The SQL Preprocessor . . . . .	95
	The HotSync Conduit Installer . . . . .	99
	The dbulstop utility . . . . .	100
	The ulconv utility . . . . .	101
	The ulcreate utility . . . . .	108
	The uldbsgen utility . . . . .	110
	The ulinit utility . . . . .	112
	The UltraLite Interactive SQL utility . . . . .	115
	The ulload utility . . . . .	117
	The ulsync utility . . . . .	119
	The ulunload utility . . . . .	121
	The ULUtil utility . . . . .	123
	The UltraLite Schema Painter . . . . .	124
	The ulxml utility . . . . .	126
<b>6</b>	<b>Tutorial: Working with UltraLite Databases</b>	<b>129</b>
	Lesson 1: Create an UltraLite database schema . . . . .	130
	Lesson 2: Define and create a consolidated database . . . . .	133
	Lesson 3: Enter data in your UltraLite database . . . . .	137
	Lesson 4: Synchronize your databases . . . . .	138
<b>II</b>	<b>UltraLite SQL</b>	<b>139</b>
<b>7</b>	<b>SQL Language Elements</b>	<b>141</b>
	Overview of SQL support in UltraLite . . . . .	142
	Data types in UltraLite . . . . .	145
	UltraLite SQL functions . . . . .	148
<b>8</b>	<b>Dynamic SQL</b>	<b>159</b>
	Introduction to dynamic SQL . . . . .	160
	Dynamic SQL expressions . . . . .	163
	Dynamic SQL operators . . . . .	166
	Dynamic SQL search conditions . . . . .	170
	Dynamic SQL statements . . . . .	172
	Query optimization . . . . .	185
<b>III</b>	<b>Application Development</b>	<b>187</b>
<b>9</b>	<b>Developing Applications for the Palm OS</b>	<b>189</b>

---

Choosing database storage on the Palm OS . . . . .	190
Understanding the Palm Creator ID . . . . .	191
<b>10 Using UltraLite Static Interfaces</b>	<b>193</b>
Overview . . . . .	194
Choosing an UltraLite static interface . . . . .	197
Preparing a reference database . . . . .	198
Defining SQL statements for your application . . . . .	202
Generating the UltraLite data access code . . . . .	207
Configuring development tools for static UltraLite development	208
<b>11 UltraLite Static Interfaces Reference</b>	<b>209</b>
Reference database stored procedures . . . . .	210
<b>Index</b>	<b>213</b>



---

# About This Manual

Subject	This manual introduces the UltraLite database system for small devices.
Audience	This manual is intended for all 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** } ]

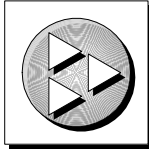
If the QUOTES option is used, one of ON or OFF must be provided. The brackets and braces should not be typed.

---

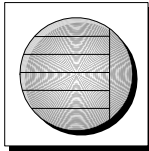
## Graphic icons

The following icons are used in this documentation.

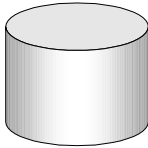
- ◆ A client application.



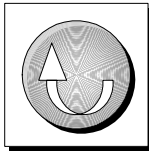
- ◆ A database server, such as Sybase Adaptive Server Anywhere.



- ◆ 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.



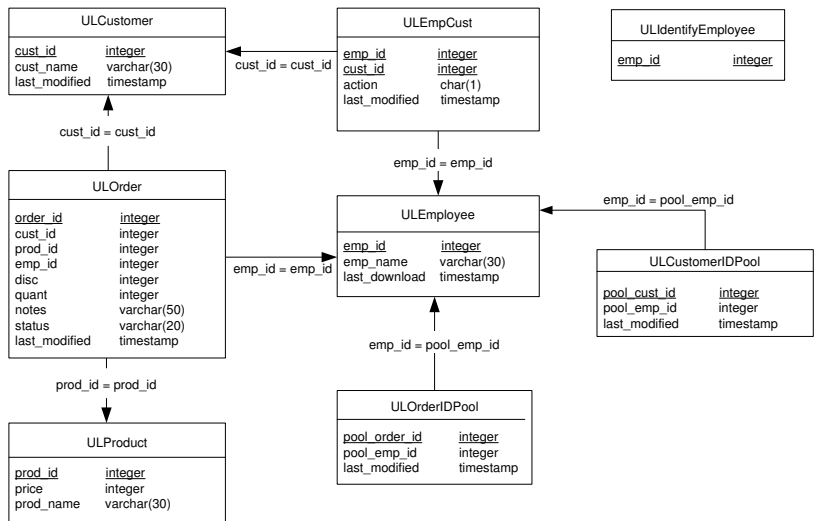
# The CustDB sample database

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](mailto: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.





PART I

# ULTRALITE DATABASES

This part introduces the UltraLite relational database system for small devices and describes general features of the UltraLite database.



---

## CHAPTER 1

# Welcome to UltraLite

About this chapter

This chapter introduces you to UltraLite features, platforms, architecture, and functionality.

Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Introduction to UltraLite</a>	4
<a href="#">Choosing an UltraLite programming interface</a>	10

---

# Introduction to UltraLite

UltraLite is a relational database management and synchronization system for small, mobile, and embedded devices. It provides the following benefits:

- ◆ **Robust data management** Data held on small devices is as important as data in enterprise databases. UltraLite brings transaction processing, referential integrity, and other benefits of relational database systems to small devices.

☞ For more information about UltraLite database features, see [“UltraLite database features” on page 6](#).

- ◆ **Powerful synchronization** UltraLite gives you the ability to synchronize data with a central database.

UltraLite uses MobiLink synchronization technology, included in SQL Anywhere Studio, to synchronize with industry-standard database-management systems. MobiLink synchronization works with Sybase Adaptive Server Anywhere, Sybase Adaptive Server Enterprise, IBM DB2, Microsoft SQL Server, and Oracle. It provides flexible, programmable, and scalable synchronization that can manage thousands of UltraLite databases.

☞ For more information, see [“UltraLite Clients” \[MobiLink Clients, page 277\]](#).

- ◆ **Your choice of programming interface** UltraLite components provide the option of using an object-based programming interface for straightforward access to data. Integration into popular development tools such as Visual Studio .NET, AppForge MobileVB and Crossfire, Borland JBuilder, and eMbedded Visual Basic improves developers’ productivity. Graphical tools enable you to design and modify UltraLite databases quickly.

☞ For more information, see [“UltraLite programming interfaces” on page 4](#) and [“Choosing an UltraLite programming interface” on page 10](#).

- ◆ **Multi-platform availability** You can develop and deploy UltraLite database applications for Windows CE, Palm OS, Windows XP, and Java-based devices.

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

## UltraLite programming interfaces

UltraLite provides a variety of programming interfaces and integrates into several popular programming tools. Each interface uses the same underlying UltraLite runtime library.

The interfaces fall into two categories: **components** and **static interfaces**. Each interface has its own strengths and has particular cases for which it is an appropriate choice. For tips on choosing a programming interface, see “[Choosing an UltraLite programming interface](#)” on page 10.

- ◆ **UltraLite components** UltraLite components provide users of development tools with a relational database and synchronization features. They provide a familiar interface for each supported development tool. UltraLite components provide a simple table-based data access interface and also dynamic SQL for more complex queries.

The following components are available:

- **UltraLite for MobileVB** Development using the AppForge MobileVB extension to Microsoft Visual Basic or the AppForge Crossfire extension to Microsoft Visual Studio .NET.  
☞ See [UltraLite for MobileVB User's Guide](#).
  - **UltraLite ActiveX** Development using eMbedded Visual Basic or JScript with Pocket IE.  
☞ See [UltraLite ActiveX User's Guide](#).
  - **Native UltraLite for Java** Development using a supported JDK. The UltraLite component itself accesses native (C++) methods for improved performance.  
☞ See [Native UltraLite for Java User's Guide](#).
  - **UltraLite.NET** Development using Visual Studio .NET, including an ADO.NET programming interface.  
☞ See [UltraLite.NET User's Guide](#).
  - **UltraLite C++ Component** Development using a C++ interface.  
☞ See [UltraLite C/C++ User's Guide](#).
  - **UltraLite for M-Business Anywhere** Development using M-Business Anywhere.  
☞ See [UltraLite for M-Business Anywhere User's Guide](#).
  - **ODBC** UltraLite supports a subset of the ODBC programming interface.  
☞ See “[Tutorial: Build an Application Using ODBC](#)” [[UltraLite C/C++ User's Guide](#), page 189] and “[UltraLite ODBC API Reference](#)” [[UltraLite C/C++ User's Guide](#), page 389].
- ◆ **Static interfaces** Static interfaces provide a rich SQL interface for C/C++ and Java developers comfortable with a preprocessor-based interface. All SQL statements used in the application must be defined at compile time.

The following static interfaces are available:

- 
- **Embedded SQL and Static C++ API** Development using C/C++ with embedded SQL statements.  
☞ See *UltraLite C/C++ User's Guide*.
  - **UltraLite Static Java** Development in pure Java using a JDBC interface. This interface uses a different runtime library to the other UltraLite interfaces.  
☞ See *UltraLite Static Java User's Guide*.

## UltraLite database features

UltraLite provides the following features:

- ◆ **Tables** A single UltraLite database file can hold many tables. The number and type of columns in a table is fixed at design time, but each table can have any number of rows (up to 64 K). Each row has a single entry for each column. The special NULL entry is used when there is no value for the entry.

When designing your database, each table should represent a separate type of item, such as customers or employees.

- ◆ **Data types** UltraLite databases can hold a full range of data types, as well as default values and NULL values.
- ◆ **Indexes** The rows in a relational database table are not ordered. You can create indexes to access the rows in order and to provide fast access to data. Indexes are commonly associated with a single column, but UltraLite also provides multi-column indexes.
- ◆ **Keys** Each table has a special index called the **primary key**. Entries in the primary key column or columns must be unique.

**Foreign keys** relate the data in one table to that in another. Each entry in the foreign key column must correspond to an entry in the primary key of another table.

Between them, primary keys and foreign keys ensure that the database has **referential integrity**. Referential integrity is enforced in UltraLite databases so that you cannot, for example, enter an order for a customer unless that customer exists in the database.

By enforcing referential integrity, UltraLite ensures that the data in your UltraLite database is correct, in the same manner that data elsewhere in the enterprise is correct. Referential integrity is checked when operations are committed, providing you with flexibility in the order with which you make changes to your data.

- ◆ **Publications** To synchronize the data in your UltraLite database with other databases you must have a valid SQL Anywhere Studio license. SQL Anywhere Studio includes MobiLink synchronization technology to synchronize UltraLite databases with desktop, workgroup or enterprise databases.

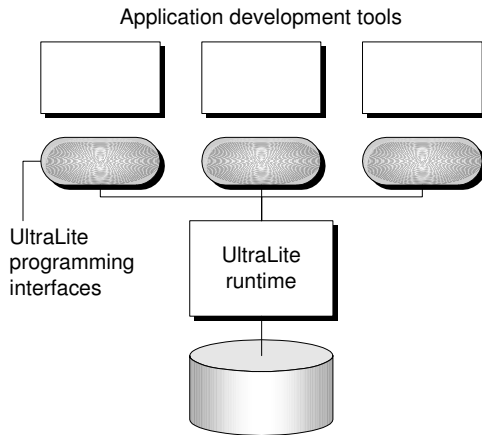
A publication defines a set of data to be synchronized. It is often desirable to synchronize all the data in an UltraLite database, but publications provide extra flexibility and control. They allow you to perform priority synchronizations, which means you can specify that only certain tables or groups of tables should be synchronized.

- ◆ **Transactions and recovery** UltraLite has commit and rollback features, together with automatic recovery in the event of device failure, to guarantee that transactions are executed completely or not at all.
- ◆ **Security** UltraLite provides user authentication and database encryption, as well as encryption during synchronization, to build secure applications.
- ◆ **Performance and small footprint** UltraLite target devices tend to have relatively slow processors. UltraLite employs algorithms and data structures that provide high performance and low memory use. For example, UltraLite provides a caching algorithm designed specifically for small devices.
- ◆ **Multi-threaded applications** For platforms that support multi-threaded applications, UltraLite supports both multi-threaded applications and multiple applications connecting to a single database.

☞ For information about working with UltraLite databases, see [“UltraLite Databases” on page 27](#).

## UltraLite application architecture

All UltraLite applications are built on the same underlying database management code. UltraLite components include this code in the component itself, while static interfaces use the code in the form of a separate UltraLite runtime library.



UltraLite applications consist of the following:

- ◆ Your application code
- ◆ The UltraLite component or runtime library
- ◆ An UltraLite database or schema file

## Using the UltraLite documentation

Once you have selected an UltraLite programming interface, you can find the information you need in the following books, all of which are included in the SQL Anywhere online books.

☞ For tips on choosing a programming interface, see [“Choosing an UltraLite programming interface”](#) on page 10.

- ◆ **UltraLite Database User’s Guide (this book)** This book presents information that is useful for all UltraLite interfaces, including information about UltraLite database management, SQL, and synchronization.
- ◆ **Interface books** Each UltraLite interface has a separate book, which contains all the information you need for developing applications using that interface.
  - [Native UltraLite for Java User’s Guide](#)
  - [UltraLite ActiveX User’s Guide](#)
  - [UltraLite C/C++ User’s Guide](#)
  - [UltraLite for MobileVB User’s Guide](#)
  - [UltraLite Static Java User’s Guide](#)
  - [UltraLite.NET User’s Guide](#)



- *UltraLite for M-Business Anywhere User's Guide*
- ◆ **MobiLink books** If your application includes synchronization, the *MobiLink Administration Guide* and the *MobiLink Clients* provide a complete guide to the synchronization system.

---

## Choosing an UltraLite programming interface

Choosing which UltraLite programming interface to use depends primarily on your answers to the following questions:

- ◆ What is your target platform or platforms?
- ◆ Which programming language do you wish to use?

The availability of more than one interface for C/C++ developers, and for Java developers, provides further flexibility.

Each interface is described in a separate book. For more information, see [“Using the UltraLite documentation” on page 8](#).

Cross platform  
development for Palm  
OS, Windows XP, and  
Windows CE

Your options are as follows:

- ◆ **C/C++** You can choose from the following:
  - ◆ UltraLite C++ Component
  - ◆ Static C++ API
  - ◆ Embedded SQL (static interface)

For information about choosing between these interfaces, see [“Choosing between components and static interfaces” on page 11](#).

- ◆ **Visual Basic .NET and Visual Basic** You can use UltraLite for MobileVB together with AppForge MobileVB or AppForge Crossfire to develop cross-platform applications from a Microsoft development environment.
- ◆ **Web development** You can use UltraLite for M-Business Anywhere to develop cross-platform web applications.

Development for Palm  
OS only

Your options are as follows:

- ◆ **C/C++** You can choose from the following:
  - ◆ UltraLite C++ Component
  - ◆ Static C++ API
  - ◆ Embedded SQL (static interface)

For information about choosing between these interfaces, see [“Choosing between components and static interfaces” on page 11](#).

- ◆ **Visual Basic** You can use UltraLite for MobileVB to develop Visual Basic applications for the Palm OS.
- ◆ **Web development** You can use UltraLite for M-Business Anywhere to develop browser-based applications for the Palm OS.

Development for  
Windows CE and  
Windows XP

Your options are as follows:

- ◆ **C/C++** You can choose from the following:
  - ◆ UltraLite C++ Component
  - ◆ ODBC
  - ◆ Static C++ API
  - ◆ Embedded SQL (static interface)

For information about choosing between these interfaces, see [“Choosing between components and static interfaces” on page 11](#).

- ◆ **Java** You can choose from the following:
  - ◆ Native UltraLite for Java (component)
  - ◆ Static Java API

For information about choosing between these interfaces, see [“Choosing between components and static interfaces” on page 11](#).

- ◆ **Visual Basic** You can choose from the following:
  - UltraLite.NET, which provides an interface from Visual Basic .NET.
  - UltraLite for MobileVB, which provides an interface from Visual Basic, using the AppForge MobileVB extension.
  - UltraLite ActiveX, which provides an interface from eMbedded Visual Basic.
- ◆ **C#** You can use UltraLite.NET to develop C# applications for Windows CE or Windows XP.
- ◆ **Web development** You can build JavaScript applications using the following:
  - ◆ UltraLite for M-Business Anywhere
  - ◆ UltraLite ActiveX

Development for other  
platforms

UltraLite Static Java provides a pure Java solution for any platform supporting JDK 1.1.8 or later.

## Choosing between components and static interfaces

UltraLite applications are built using either an UltraLite **component** or a **static interface**.

The choice of which to use depends partly on the language you wish to use. If you are a C#, Visual Basic, or JavaScript programmer, you should choose an UltraLite component. If you are a C/C++ or Java programmer, you have a choice between using a component or a static interface. This section compares components and static interfaces.

---

## Data access features

UltraLite components can use either dynamic SQL or a table-based API to access data.

- ◆ Dynamic SQL provides many common SQL features, including multi-table joins. In contrast to the static interfaces, dynamic SQL permits SQL statements to be constructed at runtime.

☞ For a full description, see [“Dynamic SQL” on page 159](#).

- ◆ The UltraLite component table-based API accesses rows one at a time. It is simple, but if you require logic such as joining tables, then you must implement it yourself within your application.

You can combine dynamic SQL and the table-based API in a single application.

Static interfaces (embedded SQL, the Static C++ API, and the static Java API) use SQL to access data. A wider range of SQL is supported in the static interfaces than in dynamic SQL, but all queries must be specified at compile time. For example, UNION and FULL OUTER JOIN queries are currently supported only from the static interfaces. In both static and dynamic SQL, queries can contain parameters, for which you can supply values at runtime.

The Static C++ API also provides a table-based API, which has some of the benefits and limitations of the component table-based API. For a description, see [“Developing Applications Using the Static C++ API” \[UltraLite C/C++ User’s Guide, page 41\]](#).

## Application size

The UltraLite components include code to parse, optimize, and execute arbitrary queries. In contrast, the static interfaces generate code that executes specified queries, but do not need to generate code to parse or optimize queries. For this reason, applications built using the static interfaces are typically much smaller than those built with UltraLite components.

As the number of queries and tables in the database increases, the size advantage of the static interfaces is lost. For complex applications using many queries and addressing databases that contain many tables, the components can be smaller as they do not need to contain code for each separate query.

## Development model

Each UltraLite component exposes an object-oriented API designed to be familiar to users of the language supported by that component. The development model is similar to that for many other kinds of applications.

The UltraLite static interfaces require a more complicated development model, in which a preprocessing step generates application code from a reference database.

For many users, the UltraLite components are easier to learn than the static interfaces.

For more information, see the following sections:

- ◆ “[Developing applications with UltraLite components](#)” on page 13
- ◆ “[Developing Static C++ applications](#)” [*UltraLite C/C++ User’s Guide*, page 6]
- ◆ “[Developing embedded SQL applications](#)” [*UltraLite C/C++ User’s Guide*, page 7]
- ◆ “[Data Access Using the Static Java API](#)” [*UltraLite Static Java User’s Guide*, page 19]

#### Performance

Queries included as part of an application that uses a static interface are already parsed and optimized. Therefore, they may perform better than queries in UltraLite components. The optimization of queries in the static interface depends on the distribution of data in the reference database. The closer the data in the reference database is to that in the UltraLite database, the better the performance will be.

☞ For more information about Adaptive Server Anywhere SQL support, see “[SQL Statements](#)” [*ASA SQL Reference*, page 253].

#### Compatibility with Adaptive Server Anywhere

Embedded SQL provides a common static programming interface for UltraLite and Adaptive Server Anywhere databases. ADO.NET and ODBC provide programming models that are shared between UltraLite components and Adaptive Server Anywhere.

Maintaining a common interface may be particularly useful on platforms such as Windows CE, where both databases are available. If you need to move from UltraLite to the more powerful and full-featured Adaptive Server Anywhere database, using embedded SQL, ODBC, or ADO.NET makes migrating your application easier.

## Developing applications with UltraLite components

### ❖ To develop an UltraLite component application

1. Design your database schema file.

You create a database schema using the UltraLite Schema Painter or by writing an XML file. Users of SQL Anywhere Studio can generate an UltraLite database schema from an Adaptive Server Anywhere database.

☞ For more information, see “[The UltraLite Schema Painter](#)” on page 124, “[The ulxml utility](#)” on page 126, and “[The ulinit utility](#)” on page 112.

---

2. Set up your development environment for UltraLite projects.

☞ For more information, see the following:

- ◆ UltraLite for MobileVB: “Lesson 1: Create a project architecture” [*UltraLite for MobileVB User's Guide*, page 43].
- ◆ UltraLite.NET: “Lesson 1: Create a Visual Studio project” [*UltraLite.NET User's Guide*, page 7].
- ◆ UltraLite C++ Component: “Tutorial: Build an Application Using the C++ Component” [*UltraLite C/C++ User's Guide*, page 147].
- ◆ UltraLite ActiveX: “Adding UltraLite ActiveX to the eMbedded Visual Basic design environment” [*UltraLite ActiveX User's Guide*, page 6].
- ◆ Native UltraLite for Java: “Developing applications with Borland JBuilder” [*Native UltraLite for Java User's Guide*, page 60].
- ◆ UltraLite for M-Business Anywhere: “UltraLite for M-Business Anywhere Quick Start” [*UltraLite for M-Business Anywhere User's Guide*, page 6].

3. Write your application code using the UltraLite API for a particular language.

4. Deploy your application and database schema.

The steps depend on the target device and on the component used.

---

## CHAPTER 2

# Tutorial: The CustDB Sample UltraLite Application

### About this chapter

This chapter uses the CustDB sample application to illustrate some key features of UltraLite.

These techniques are illustrated using a desktop version of CustDB. Additionally, a version of CustDB is provided for each of the supported interfaces. For more information, see [“What Next?” on page 26](#).

Much of the material in this chapter is explained in a more general manner elsewhere in this book.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Introduction</a>	16
<a href="#">Lesson 1: Start the MobiLink synchronization server</a>	19
<a href="#">Lesson 2: Start the sample application and synchronize</a>	20
<a href="#">Lesson 3: Add an order</a>	21
<a href="#">Lesson 4: Approve or deny an existing order</a>	22
<a href="#">Lesson 5: Synchronize your changes</a>	23
<a href="#">Lesson 6: Browse the consolidated database</a>	25

---

# Introduction

This chapter introduces you to the CustDB (Customer Database) UltraLite sample application. CustDB is a sales-status application.

The CustDB sample application provides you with examples of how to implement many of the techniques you will need to develop UltraLite applications.

This chapter uses the compiled version of the application for Windows NT/2000/XP.

A separate version of the CustDB application is provided for each UltraLite programming interface. Each version has similar features, with some variation to conform to the conventions of each platform.

## UltraLite features

This chapter illustrates the following UltraLite features:

- ◆ UltraLite database applications run on small devices using very limited resources.
- ◆ UltraLite applications include a relational database engine.
- ◆ UltraLite applications share data with a central, consolidated database in a two-way synchronization scheme. UltraLite databases are also called **remote** databases.
- ◆ Each remote database contains a subset of the data in the consolidated database.
- ◆ The MobiLink synchronization server, included with SQL Anywhere Studio, carries out data synchronization between the consolidated database and each UltraLite installation.
- ◆ SQL scripts stored in the consolidated database implement the synchronization logic.
- ◆ You can use Sybase Central to browse and edit the synchronization scripts.

## Scenario

The CustDB scenario is as follows:

A consolidated database is stored at the head office.

There are two types of remote databases, sales representatives and mobile managers. Each sales representative's UltraLite remote database contains all products but only those orders assigned to that sales representative. A mobile manager's UltraLite remote database contains all products and orders.

You can carry out the following tasks with the sample application.



- ◆ View lists of customers and products.
- ◆ Add new customers.
- ◆ Add or delete orders.
- ◆ Scroll through the list of outstanding orders.
- ◆ Accept or deny orders.
- ◆ Synchronize changes with the consolidated database.

When you run the CustDB UltraLite application, you are working on a single remote database, and synchronizing your changes with a consolidated database.

In a typical UltraLite installation, there will be many remote databases, each running on a handheld device, and each containing a small subset of the data from the consolidated database.

## File locations

The CustDB sample application is included in the following locations within your UltraLite installation.

### Runtime file locations

To run the CustDB sample application, you need the following components:

- ◆ **The consolidated database** An Adaptive Server Anywhere version of the CustDB database is located in `Samples\UltraLite\Custdb\custdb.db`. For information about the schema, and information about using one of the other supported consolidated database types, see “[The CustDB Sample Application](#)” [*MobiLink Tutorials*, page 99].

This database serves as the consolidated database. It contains the following information.

- MobiLink system tables that hold the synchronization metadata.
- The CustDB data, stored in rows in base tables.
- The synchronization scripts.

During installation, an ODBC data source named UltraLite 9.0 Sample is created for this database.

- ◆ **The UltraLite application executable and source code** A version of the sample is supplied for each interface.

The executable and source code are located in the following subdirectories of your SQL Anywhere Studio installation:

---




<b>Component</b>	<b>Location</b>
UltraLite for MobileVB	<i>Samples\UltraLiteForMobileVB and Samples\UltraLiteForCrossfire</i>
UltraLite ActiveX	<i>Samples\UltraLiteActiveX</i>
UltraLite.NET	<i>Samples\UltraLite.NET</i>
Native UltraLite for Java	<i>Samples\NativeUltraLiteForJava</i>
C++ Component	<i>Samples\UltraLite</i>
Embedded SQL	<i>Samples\UltraLite</i>
Static C++ API	<i>Samples\UltraLite</i>
Static Java API	<i>Samples\UltraLite</i>
UltraLite for M-Business Anywhere	<i>Samples\UltraLiteForMBusinessAnywhere</i>

To browse the samples from the Start menu, choose Programs ► SQL Anywhere 9 ► Sample Applications and Projects.

## Synchronization techniques in the sample application

Synchronization with the CustDB sample database is carried out by the MobiLink synchronization server running on your desktop computer.

The CustDB sample application demonstrates several useful synchronization techniques. For a more detailed explanation of how synchronization works, refer to the following sections in the MobiLink documentation.

- ◆  For an overview of the synchronization process, see “[The synchronization process](#)” [*MobiLink Administration Guide*, page 15].
- ◆  For a description of how to write the synchronization scripts that control synchronization, see “[Writing Synchronization Scripts](#)” [*MobiLink Administration Guide*, page 227].
- ◆  For information about the synchronization techniques used in the CustDB sample application, see “[The CustDB Sample Application](#)” [*MobiLink Tutorials*, page 99].

## Lesson 1: Start the MobiLink synchronization server

When you start the sample UltraLite application for the first time, the remote database contains no data. The application carries out synchronization to download an initial copy of the data from a consolidated database.

Before you can carry out synchronization, you must start the database server, and start the MobiLink synchronization server running against the UltraLite sample database.

The following procedure uses some of the shortcuts that the SQL Anywhere Studio installation adds to the Start menu.

### ❖ To start the MobiLink synchronization server

1. Start the Adaptive Server Anywhere CustDB sample database.

From the Start menu, choose Programs ► Sybase SQL Anywhere 9 ► UltraLite ► Personal Server Sample for UltraLite.

The CustDB sample database is the consolidated database in this tutorial. You will synchronize data between an UltraLite database and this consolidated database. You can also use database servers other than Adaptive Server Anywhere as consolidated databases. For more information, see [“The CustDB Sample Application”](#) [*MobiLink Tutorials*, page 99].

2. Start the MobiLink synchronization server, running against the CustDB database.

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

The MobiLink synchronization server uses an ODBC driver to connect to the consolidated database server. It can run on a separate computer from the database server, but in this example it runs on the same computer.

---

## Lesson 2: Start the sample application and synchronize

The following procedure starts the sample application, and synchronizes with the consolidated database to obtain an initial set of data. The data you download depends on the user ID you enter when you start the application.

### ❖ To start and synchronize the sample application

1. Launch the sample application.

From the Start menu, choose Programs ► SQL Anywhere 9 ► UltraLite ► Windows Sample Application.

In this walkthrough the sample application is running on the same desktop machine as the MobiLink synchronization server. In a production environment, UltraLite applications more commonly run on handheld devices.

2. Enter an employee ID.

Enter a value of 50 and click Enter. The application also allows values of 51, 52, or 53, but behaves slightly differently in these cases.

After you enter the employee ID, the application synchronizes, and a set of customers, products, and orders is downloaded to the application. The MobiLink synchronization server window displays messages showing the synchronization taking place.

3. Confirm that the application contains data.

A company name and a sample order should appear on the application window.

## Lesson 3: Add an order

In this section, you add a new order to the database. Adding an order is carried out in a similar way in each version of the application.

The database contains a table in which each row holds information about a particular order. For each order, this data includes the customer, the product, the quantity, the price, and any applicable discount. Also included are a status field and a notes field, which you can modify from this application.

The remote database does not receive all the orders listed in the ULOrder table in the consolidated database. Only orders that have not yet been approved are downloaded. Synchronization scripts allow you to control which information is sent to your application.

### Add an order

#### ❖ To add an order

1. Scroll through the outstanding orders.

Click Next to display the next customer.

2. Enter a new order.

From the Order menu, choose New.

The Add New Order screen is displayed.

3. Choose a customer.

Choose **Basements R Us** from the dropdown list. This list exposes the complete list of customers from the consolidated database.

The current list of orders does not have any from this customer.

4. Choose a product.

The UltraLite application holds the complete list of products from the consolidated database. To see this list, open the Product drop-down list box.

Choose **Screwmaster Drill** from the list. The price of this item is automatically entered in the Price field.

5. Enter the quantity and discount.

Enter a value of 20 for the quantity, and a value of 5 for the discount.

6. Click Enter to add the new order.

By adding an order, you have modified the data in your local UltraLite database. This data is not shared with the consolidated database until you synchronize.

---

## Lesson 4: Approve or deny an existing order

In this step, you approve one order and deny another. Approving or denying orders updates two columns in the local database. The data in the consolidated database is unchanged until you synchronize.

### ❖ To approve, deny, and delete orders

1. Approve the order from Apple Street Builders.
  - ◆ Go to the first order in the list, which is from Apple Street Builders.
  - ◆ Click Approve to approve the order.
  - ◆ Add a note to your approval, saying Good Work.
  - ◆ The order appears with a status of Approved.
2. Deny the order from Art's Renovations.
  - ◆ Go to the next order in the list, which is from Art's Renovations.
  - ◆ Click Deny to deny this order.
  - ◆ Add a note stating Discount too high.
3. Delete the order from Awnings R Us.
  - ◆ Go to the next order in the list, which is from Awnings R Us.
  - ◆ Delete this order by choosing the menu item Order ► Delete. It disappears from your local copy of the data.

## Lesson 5: Synchronize your changes

In this section, you synchronize the changes you made to the remote database with the consolidated database.

For synchronization to take place, the MobiLink synchronization server must be running. If you have shut down your MobiLink synchronization server, restart it as described in “[Lesson 1: Start the MobiLink synchronization server](#)” on page 19.

### ❖ To synchronize your changes

1. Choose File ► Synchronize to synchronize your data.
2. Confirm that synchronization took place.

The synchronization process for the sample application removes approved orders from your database. Confirm that the approved order for Apple Street Builders is no longer in your application.

## Confirm the synchronization at the consolidated database

In this section, you use Interactive SQL to connect to the consolidated database to confirm that your changes were synchronized.

### ❖ To confirm the synchronization at the consolidated database

1. Connect to the consolidated database from Interactive SQL.
  - ◆ Choose Start ► Programs ► SQL Anywhere 9 ► Adaptive Server Anywhere ► Interactive SQL.  
The Interactive SQL Connect dialog appears.
  - ◆ Select ODBC Data Source Name and choose UltraLite 9.0 Sample from the dropdown list.

2. Confirm the status change of the approved and denied orders.

To confirm that the approval and denial have been synchronized, issue the following statement.

```
SELECT order_id, status
FROM ULOrder
WHERE status IS NOT NULL
```

The results show that order 5100 is approved, and 5101 is denied.

3. Confirm that the deleted order has been removed.

The deleted order has an order\_id of 5102. The following query returns no rows, demonstrating that the order has been removed from the system.

---

```
SELECT *  
FROM ULOrder  
WHERE order_id = 5102
```



## Lesson 6: Browse the consolidated database

You can use Sybase Central to manage MobiLink synchronization. The synchronization logic is held in the consolidated database.

This section describes how to use Sybase Central to browse the scripts in the CustDB consolidated database.

### Connect to the CustDB database from Sybase Central

1. Start the CustDB database.
  - ◆ Select Programs ► Sybase SQL Anywhere 9 ► UltraLite ► Personal Server Sample for UltraLite.
2. Start Sybase Central.
  - ◆ From the Start menu, select Programs ► SQL Anywhere 9 ► Sybase Central.
3. Connect to the sample database.
  - ◆ In Sybase Central, select Tools ► Connect. If there is a choice of connection types, select MobiLink.  
The MobiLink Connect dialog appears.
  - ◆ Select ODBC and enter UltraLite 9.0 Sample in the Data Source box. Click OK to connect.

### Browse the synchronization scripts

From Sybase Central, you can browse the tables, users, synchronized tables, and synchronization scripts that are stored in the consolidated database. Sybase Central is the primary tool for adding these scripts to the database.

#### ❖ To browse the synchronization scripts

1. Open the Connection Scripts folder.

The right hand pane lists a set of synchronization scripts and a set of events with which these scripts are associated. As the MobiLink synchronization server carries out the synchronization process, it triggers a sequence of events. Any synchronization script associated with an event is run at that time. By writing synchronization scripts and assigning them to the synchronization events, you can control the actions that are carried out during synchronization.
2. Open the Synchronized Tables folder, and open the ULCustomer table folder.

---

The right hand pane lists a pair of scripts that are specific to this table, and their corresponding events. These scripts control the way that data in the ULCustomer table is synchronized with the remote databases.

The content of the synchronization scripts is discussed in detail in [“Writing Synchronization Scripts”](#) [*MobiLink Administration Guide*, page 227] and [“The CustDB Sample Application”](#) [*MobiLink Tutorials*, page 99].

## What Next?

In addition to the CustDB application, tutorials are provided for each of the supported interfaces. For more information, see the following sections:

- ◆ **Native UltraLite for Java** [“Tutorial: The CustDB Sample Application”](#) [*Native UltraLite for Java User’s Guide*, page 23].
- ◆ **UltraLite ActiveX** [“Tutorial: A Sample UltraLite ActiveX Application”](#) [*UltraLite ActiveX User’s Guide*, page 43] or [“Tutorial: An UltraLite ActiveX Application for Pocket IE”](#) [*UltraLite ActiveX User’s Guide*, page 63].
- ◆ **UltraLite C++ Component** [“Tutorial: Build an Application Using the C++ Component”](#) [*UltraLite C/C++ User’s Guide*, page 147].
- ◆ **UltraLite Embedded SQL** [“Tutorial: Build an Application Using Embedded SQL”](#) [*UltraLite C/C++ User’s Guide*, page 177].
- ◆ **UltraLite for MobileVB** [“Tutorial: A Sample UltraLite for MobileVB Application”](#) [*UltraLite for MobileVB User’s Guide*, page 41].
- ◆ **UltraLite.NET** [“Tutorial: Build an UltraLite.NET Application”](#) [*UltraLite.NET User’s Guide*, page 5].
- ◆ **UltraLite for M-Business Anywhere** [“UltraLite for M-Business Anywhere Quick Start”](#) [*UltraLite for M-Business Anywhere User’s Guide*, page 6]

---

## CHAPTER 3

# UltraLite Databases

About this chapter

This chapter provides basic information about data storage, user authentication, and character set issues in UltraLite databases.

Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Creating UltraLite databases and schemas</a>	28
<a href="#">Setting UltraLite database properties</a>	33
<a href="#">User authentication in UltraLite</a>	40
<a href="#">Character sets in UltraLite</a>	43
<a href="#">UltraLite database internals</a>	47
<a href="#">UltraLite database limitations</a>	50
<a href="#">Upgrading UltraLite database schemas</a>	54
<a href="#">The UltraLite runtime</a>	58

---

## Creating UltraLite databases and schemas

UltraLite databases are held in a single file (or, in the case of Palm OS, in the Palm persistent store). The database file contains tables, indexes, and also contains additional information required for synchronization.

Every database contains a **schema**: information about the tables, indexes, and so on that make up the database. This metadata includes column names and data types, primary and foreign key definitions, and so on. Most relational databases store the schema in a special set of tables called the **system tables**, or **catalog**. UltraLite stores its schema in a more compact form.

### Creating an UltraLite schema

To create an UltraLite database, you create an UltraLite schema externally, separately from the database itself.

If you are developing applications using an UltraLite component, you create the schema in a schema file. You then apply the schema to an UltraLite database file.

If you are developing applications using a static interface, you create the schema in a reference database; when you generate your application code from the reference database, the schema is added to the application itself.

☞ For more information, see [“Creating UltraLite schema files” on page 29](#).

### Applying a schema to a database

It is common not to deploy UltraLite database files with your application, but instead to let UltraLite create the database file and to use synchronization to fill up the database with the appropriate data.

UltraLite component applications require that you deploy a database schema file. You can write your application so that it creates a database file on the first connection attempt, and applies the schema file to this database file. Sample code is provided in the tutorials for each programming interface.

UltraLite static interface applications contain a database schema definition in the generated code. They automatically create a database and apply the schema on the first connection attempt.

☞ For more information, see [“Creating UltraLite databases” on page 30](#).

### Altering a database schema

You can alter the schema of an UltraLite database by applying a new schema to the database file. As with the original schema, a new schema may be held in a schema file (UltraLite components) or in a new version of the application (static interfaces).

If you are using an UltraLite component, you can also alter the schema of an UltraLite database by executing SQL statements that modify tables and indexes (data definition statements).

☞ For more information, see [“Upgrading UltraLite database schemas” on page 54](#).

See also

☞ For an introduction to UltraLite database features, see [“UltraLite database features” on page 6](#).

## Creating UltraLite schema files

UltraLite schema files are used with UltraLite components. You can create an UltraLite schema file in the following ways:

- ◆ **UltraLite Schema Painter** The simplest way to create a schema file is to use the graphical UltraLite Schema Painter.

To start the Schema Painter, choose Start ► Programs ► SQL Anywhere 9 ► UltraLite ► UltraLite Schema Painter, or double-click a schema file (with extension `.usm`) in Windows Explorer. For more information, see [“The UltraLite Schema Painter” on page 124](#).

- ◆ **Generate the schema from an Adaptive Server Anywhere database** If you already have an Adaptive Server Anywhere database with the schema needed for your UltraLite database, or a superset of that schema, you can generate an UltraLite schema file using the `ulinit` command line utility.

☞ For more information, see [“The ulinit utility” on page 112](#).

You can use the Adaptive Server Anywhere migration wizard to migrate table definitions, indexes, and data from other databases into an Adaptive Server Anywhere database. You can use this wizard to assist in creating UltraLite schema definitions from Adaptive Server Enterprise, Oracle, SQL Server, or DB2 databases.

☞ For more information, see [“Migrating databases to Adaptive Server Anywhere” \[ASA SQL User’s Guide, page 591\]](#).

- ◆ **Convert an XML file to a schema file** The `ulxml` command line utility allows you to convert XML files to schema files. It can also carry out the conversion of schema files to XML files.

The utility is useful, for example, if you wish to keep your database definition under source control. It also integrates well into automated build processes.

For more information, see [“The ulxml utility” on page 126](#).

- ◆ **Unload the schema from an existing UltraLite database** The `ulconv` command line utility allows you to carry out numerous operations on UltraLite databases, including unloading a schema from an existing database.

---

☞ For more information, see [“The ulconv utility” on page 101](#).

When you create an UltraLite schema file, you set some database-wide characteristics. These include the following:

- ◆ **Case sensitivity** The case sensitivity of a database affects all string comparisons. It must be specified when the database is created because indexes are stored in sorted order, and the order depends on whether the database is case sensitive or not.
- ◆ **Character set** Each database has a well-defined collation sequence (character set and sort order). The collation sequence is defined when the database is created.

☞ For more information, see [“Character sets in UltraLite” on page 43](#).

## Creating UltraLite databases

The way you create an UltraLite database depends on the development model you use.

### ❖ To create an UltraLite database (UltraLite components)

1. Create an UltraLite schema file.

☞ For more information, see [“Creating UltraLite schema files” on page 29](#).

2. Write your application to use the schema file when a database is not found.

Your code that connects to the database should carry out the following steps:

```
Open Connection( database identification parameters )
If ( database not found ) Then
    Create Database( database schema parameters )
End If
```

For a list of database identification parameters, see [“Database Identification parameters” on page 68](#). For a list of database schema parameters, see [“Database Schema parameters” on page 78](#).

The specifics depend on the component you are using. For more information, see the following:

- ◆ UltraLite for MobileVB: [“Connecting to an UltraLite database” \[UltraLite for MobileVB User’s Guide, page 11\]](#)
- ◆ UltraLite ActiveX: [“Connecting to an UltraLite database” \[UltraLite ActiveX User’s Guide, page 11\]](#)

- ◆ Native UltraLite for Java: “Connecting to a database” [*Native UltraLite for Java User’s Guide*, page 36]
  - ◆ UltraLite.NET: “Connecting to a database” [*UltraLite.NET User’s Guide*, page 24]
  - ◆ UltraLite C++ Component: “Connecting to a database” [*UltraLite C/C++ User’s Guide*, page 17]
  - ◆ UltraLite for M-Business Anywhere: “Connecting to an UltraLite database” [*UltraLite for M-Business Anywhere User’s Guide*, page 13]
3. Add data to the UltraLite database from the application or by synchronizing.

#### ❖ To create an UltraLite database (static interface)

1. Create an Adaptive Server Anywhere **reference database**. A reference database holds the same tables and indexes as your UltraLite database.
  - ☞ For more information, see “Preparing a reference database” on page 198.
2. Generate an UltraLite application from the reference database.
  - ☞ For more information, see “Generating the UltraLite data access code” on page 207.
3. When your application first runs, it creates an UltraLite database file with the information from the reference database.
4. Add data to the UltraLite database from the application or by synchronizing.

## Creating UltraLite database files

UltraLite applications require a schema definition in order to create a database. If you are using an UltraLite component, you deploy the database schema in a separate schema file. If you are using a static development model, your application contains the schema definition.

The physical storage of the UltraLite database depends on the target platform.

- ◆ **Palm Computing Platform** The database is stored in the Palm persistent (static) memory using the Data Manager API. For devices operating Palm OS version 4.0, you can store UltraLite databases in the file-based storage of expansion cards.

☞ For more information, see “Choosing database storage on the Palm OS” on page 190 and “Database On Palm connection parameter” on page 71.

- 
- ◆ **Windows and Windows CE** The database is stored in the file system. On Windows CE the default file is `\UltraLiteDB\ul.udb`. On other versions of Windows the default file is `ul_<project>.udb` in the working directory of the application, where `<project>` is the UltraLite project name used during the development process.

You can choose to explicitly specify a database file name, or you can choose to use the default file name.

☞ For more information, see “[Database On CE connection parameter](#)” on page 69.

- ◆ **Static Java** The database is either transient, or is stored as a file in the file system. By default, it is transient.

For static APIs, you can supply parameters that control features such as database encryption on your first connection attempt (which is when the database is created). For UltraLite components, set the schema parameters of the Create Database method.

For more information

- ◆ **UltraLite for MobileVB** See “[CreateDatabaseWithParms method](#)” [*UltraLite ActiveX User’s Guide*, page 102].
- ◆ **UltraLite ActiveX** See “[CreateDatabaseWithParms method](#)” [*UltraLite ActiveX User’s Guide*, page 102].
- ◆ **Native UltraLite for Java** See `ianywhere.native_ultralite.DatabaseManager` in the Native UltraLite for Java API Reference.
- ◆ **UltraLite.NET** See “[ULDatabaseManager class](#)” [*UltraLite.NET User’s Guide*, page 137] (`iAnywhere.Data.UltraLite` namespace) or “[DatabaseManager class](#)” [*UltraLite.NET User’s Guide*, page 399] (`iAnywhere.UltraLite` namespace).
- ◆ **UltraLite C++ Component** See “[Class UltraLite\\_DatabaseManager](#)” [*UltraLite C/C++ User’s Guide*, page 249].
- ◆ **UltraLite for embedded SQL** See “[Macros and compiler directives for UltraLite C/C++ applications](#)” [*UltraLite C/C++ User’s Guide*, page 221].
- ◆ **UltraLite Static C++** See “[Macros and compiler directives for UltraLite C/C++ applications](#)” [*UltraLite C/C++ User’s Guide*, page 221].
- ◆ **UltraLite Static Java** See “[UltraLite API reference](#)” [*UltraLite Static Java User’s Guide*, page 54].
- ◆ **UltraLite for M-Business Anywhere** See “[Method createDatabaseWithParms](#)” [*UltraLite for M-Business Anywhere User’s Guide*, page 76].



## Setting UltraLite database properties

You can set global characteristics of UltraLite databases when the database is first created, either in the schema or on the first connection attempt.

- ◆ Character set.
  - ☞ See “Creating UltraLite schema files” on page 29.
- ◆ Case sensitivity.
  - ☞ See “Creating UltraLite schema files” on page 29.
- ◆ Data encryption.
  - ☞ See “Encrypting UltraLite databases” on page 36 and “Obfuscate connection parameter” on page 83.
- ◆ Database page size.
  - ☞ See “UltraLite database files” on page 47 and “Page Size connection parameter ” on page 83.
- ◆ Date and time formats.
  - ☞ These must be set in the schema file as database options. See “UltraLite database options” on page 33.
- ◆ Precision and scale for arithmetic operations.
  - ☞ These must be set in the schema file as database options. See “UltraLite database options” on page 33.
- ◆ The amount of memory used as a cache by the UltraLite runtime.
  - ☞ See “Cache Size connection parameter ” on page 73.
- ◆ Preallocation of file-system space.
  - ☞ See “Reserve Size connection parameter ” on page 84.

### UltraLite database options

UltraLite databases support the following set of database options, which must be set in the schema file. One set of options controls the handling of dates and times. A second set controls the default handling of arithmetic operations.

**Setting database options** For UltraLite components, the database options must be set in the schema file.

- ◆ If you are using the Schema Painter, set the options in the database property sheet.

- 
- ◆ If you are using the ulinit utility, set the options in your Adaptive Server Anywhere reference database.
  - ◆ If you are using the ulxml utility, set the options in the XML document that describes the database schema.

For UltraLite static interfaces, set the options in your Adaptive Server Anywhere reference database.

☞ For information about setting Adaptive Server Anywhere database options, see “[Setting options](#)” [*ASA Database Administration Guide*, page 614].

## Date and time options

The following database options control the default handling of dates and times. These settings can be changed within SQL operations by using functions such as the DATEFORMAT function.

- ◆ **DateFormat** Sets the default string format in which dates are retrieved from the database.

Allowed values are constructed from the symbols listed in the table that follows this list. The default value is YYYY-MM-DD.

The corresponding Adaptive Server Anywhere database option is DATE\_FORMAT. See “[DATE\\_FORMAT option \[compatibility\]](#)” [*ASA Database Administration Guide*, page 646].

- ◆ **DateOrder** Sets the default interpretation of dates when submitted to the database.

Allowed values are MDY, YMD, DMY. The default value is YMD.

The corresponding Adaptive Server Anywhere database option is DATE\_ORDER. See “[DATE\\_ORDER option \[compatibility\]](#)” [*ASA Database Administration Guide*, page 648].

- ◆ **NearestCentury** Sets the interpretation of two-integer year values when submitted to the database.

Allowed values are integers from 0 to 100 inclusive. The default value is 50. Two digit years YY less than the value are converted to 20YY, while years greater than or equal to the value are converted to 19YY.

The corresponding Adaptive Server Anywhere database option is NEAREST\_CENTURY. See “[NEAREST\\_CENTURY option \[compatibility\]](#)” [*ASA Database Administration Guide*, page 671].

- ◆ **TimeFormat** Sets the default format for times retrieved from the database.

Allowed values are constructed from the symbols listed in the table that follows this list. The default value is HH:NN:SS.SSS.

The corresponding Adaptive Server Anywhere database option is `TIME_FORMAT`. See [“`TIME\_FORMAT` option \[compatibility\]” \[ASA Database Administration Guide, page 694\]](#).

- ◆ **TimestampFormat** Sets the default format for timestamp values retrieved from the database.

Allowed values are constructed from the symbols listed in the table that follows this list. The default value is `YYYY-MM-DD HH:NN:ss.SSS`.

The corresponding Adaptive Server Anywhere database option is `TIMESTAMP_FORMAT`. See [“`TIMESTAMP\_FORMAT` option \[compatibility\]” \[ASA Database Administration Guide, page 695\]](#).

- ◆ **TimestampIncrement** As timestamps are inserted into the database, UltraLite truncates them to match this increment. This value is useful when a `DEFAULT TIMESTAMP` column is being used as a primary key or row identifier. In particular, during synchronization it can be important that timestamps match, and different supported consolidated databases maintain timestamps to different resolution. Setting the `TimestampIncrement` to match that of the consolidated database can help to avoid spurious inequalities.

Allowed values are integers greater than zero. The default value is 1.

The corresponding Adaptive Server Anywhere database option is `TRUNCATE_TIMESTAMP_VALUES`. See [“`TRUNCATE\_TIMESTAMP\_VALUES` option \[database\]” \[ASA Database Administration Guide, page 697\]](#).

The symbols used in `DateFormat`, `TimeFormat`, and `TimestampFormat` values are taken from the following table:

Symbol	Description
<code>yy</code>	Two digit year.
<code>yyyy</code>	Four digit year.
<code>mm</code>	Two digit month, or two digit minutes if following a colon (as in <code>hh:mm</code> ).
<code>mmm[m...]</code>	Character short form for months—as many characters as there are “m”s. An upper case M causes the output to be made upper case.
<code>d</code>	Single digit day of week, (0 = Sunday, 6 = Saturday).
<code>dd</code>	Two digit day of month. A leading zero is not required.

Symbol	Description
<i>ddd</i> [ <i>d...</i> ]	Character short form for day of the week. An upper case D causes the output to be made upper case.
<i>hh</i>	Two digit hours. A leading zero is not required.
<i>nn</i>	Two digit minutes. A leading zero is not required.
<i>ss</i> [ <i>.ss..</i> ]	Seconds and parts of a second.
<i>aa</i>	Indicate AM or PM (12 hour clock).
<i>pp</i>	Indicate times after noon by PM (12 hour clock).
<i>jjj</i>	Day of the year, from 1 to 366.

Arithmetical operations      The Precision and Scale options control the handling of arithmetical operations.

- ◆ **Precision**    Sets the maximum number of digits in the result of any decimal arithmetic.

Allowed values are integers between 0 and 127 inclusive. The default value is 30.

Precision is the total number of digits to the left and right of the decimal point. The Scale option specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum Precision.

- ◆ **Scale**        Sets the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum Precision.

Allowed values are integers between 0 and 127 inclusive. The default value is 6.

For example, when a DECIMAL(8,2) value is multiplied by a DECIMAL(9,2) value, the result could require a DECIMAL(17,4). If Precision is 15, only 15 digits are kept in the result. If Scale is 4, the result is a DECIMAL(15,4). If Scale is 2, the result is a DECIMAL(15,2). In both cases, there is a possibility of overflow.

## Encrypting UltraLite databases

By default, UltraLite databases are unencrypted on disk and in permanent memory. Text and binary columns are plainly readable within the database file when using a viewing tool such as a hex editor. Two options are provided for greater security:

- ◆ **Obfuscation** This option provides protection against casual attempts to access data in the database. It does not provide as much security as strong encryption. Obfuscation has minimal performance impact.
- ◆ **Strong encryption** UltraLite database files can be strongly encrypted using the AES 128-bit algorithm, which is the same algorithm used to encrypt Adaptive Server Anywhere databases. Use of strong encryption provides security against skilled and determined attempts to gain access to the data, but has a significant performance impact.

**Caution**

*If the encryption key for a strongly encrypted database is lost or forgotten, there is no way to access the database. Under these circumstances, technical support cannot gain access to the database for you. It must be discarded and you must create a new database.*

Encrypting UltraLite databases

To encrypt an UltraLite database, you supply an encryption key when you create the database file. The supplied key is used to encrypt the database. On subsequent attempts, the supplied key is checked against the encryption key, and connection fails unless the key matches.

Changing the encryption key

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

**Caution**

*When the key is changed, every row in the database is decrypted using the old key and re-encrypted using the new key. This operation is unrecoverable. If the application is interrupted part-way through, the database is invalid and cannot be accessed. A new one must be created.*

For more information

- ◆ **UltraLite for MobileVB** See “Encryption and obfuscation” [*UltraLite for MobileVB User’s Guide*, page 14].
- ◆ **UltraLite ActiveX** See “Encryption and obfuscation” [*UltraLite ActiveX User’s Guide*, page 16].
- ◆ **Native UltraLite for Java** See `ianywhere.native_ultralite.ConnectionParms` in the API Reference.
- ◆ **UltraLite.NET** See “ULConnectionParms class” [*UltraLite.NET User’s Guide*, page 100] (`Anywhere.Data.UltraLite` namespace) or “ConnectionParms class” [*UltraLite.NET User’s Guide*, page 362] (`Anywhere.UltraLite` namespace).
- ◆ **UltraLite C++ Component** See “Class `UltraLite_Connection_iface`” [*UltraLite C/C++ User’s Guide*, page 236].

- ◆ **Embedded SQL and Static C++ API** See “Encrypting data” [*UltraLite C/C++ User’s Guide*, page 49].
- ◆ **UltraLite Static Java** See “Encrypting UltraLite databases” [*UltraLite Static Java User’s Guide*, page 35].
- ◆ **UltraLite for M-Business Anywhere** See “Encryption and obfuscation” [*UltraLite for M-Business Anywhere User’s Guide*, page 17].

## Palm OS considerations

If you encrypt an UltraLite database on the Palm Computing Platform, the end user is prompted to re-enter the key each time they launch the application. This section describes how to add code that circumvents the re-entering of the key.

You can save the encryption key in dynamic memory as a Palm **feature**, and retrieve the key when you launch the application rather than prompting the user. Features are indexed by creator and a feature number. Users can pass in their creator ID or NULL, along with the feature number or NULL, to save and retrieve the encryption key.

The encryption key is not backed up and is cleared on any reset of the device. The retrieval of the key then fails, and the user is prompted to re-enter the key.

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

```
#define UL_STORE_PARMS StoreParms
static ul_char StoreParms[STORE_PARMS_MAX];
...
startupRoutine() {
    ul_char buffer[MAX_PWD];

    if( !ULRetrieveEncryptionKey(
        buffer, MAX_PWD, NULL, NULL ) ){
        // prompt user for key
        userPrompt( buffer, MAX_PWD );
        if( !ULSaveEncryptionKey( buffer, NULL, NULL ) ) {
            // inform user save failed
        }
    }
    // build store parms
    StrCopy( StoreParms, "key=" );
    StrCat( StoreParms, buffer );
    ULPalmLaunch( &sqlca, UL_NULL );
}
```

The following sample code illustrates how to use a menu item to secure the device by clearing the encryption key:

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

---

## User authentication in UltraLite

UltraLite provides optional database user IDs and passwords for user authentication. Unlike Adaptive Server Anywhere and other multi-user database systems, UltraLite user IDs are used for authentication only, not for permission checking or object ownership within a database. By default, UltraLite databases have no user authentication. Once connected to the database, each user has full access to the database.

When an UltraLite database is created, it has an initial user ID of `DBA`, with a password of `SQL`. These are also the default values when you connect, so that you can avoid user authentication by not supplying user ID or password connection parameters.

UltraLite permits up to four different user IDs to be defined at a time, with both user ID and password being less than 16 characters long.

If the database is case insensitive (the default) then the user ID and password are case insensitive. If the database is case sensitive, then the password is case sensitive.

UltraLite users have to be added from an existing connection. This means that if you want to add user authentication to your application by changing the user ID or password, you must do so once you have connected using the default user ID and password. What is more, you cannot change a user ID: you add a user and delete an existing user. The function used for changing a password is the same as for changing a user ID.

The general scheme for adding user authentication is as follows:

### ❖ To add user authentication to your application

1. Connect to the database using the default **uid** and **pwd** parameters.

New users have to be added from an existing connection, so the first connection to a database must be made using the default user ID and password of `DBA` and `SQL`.

2. Prompt for a user ID and password.

The way in which you prompt the user depends on your application.

3. Grant access to this user ID and password combination.

The method for granting access depends on the interface you are using.

4. Revoke access from the original user ID.

In a production environment, it is a security problem if you leave the default user ID and password with access to the database.



UltraLite user IDs are separate from MobiLink user names and from user IDs in any reference database or consolidated databases you use during development and after deployment. In many cases you may wish to provide code so that the values used for each are the same, but they do remain distinct concepts. For example, in the CustDB sample application, you are prompted for an employee number when starting the application. This employee number identifies the database for the purposes of MobiLink synchronization, and is not an UltraLite user ID for connection or data access purposes.

For more information

- ◆ **UltraLite for MobileVB** See “Authenticating users” [*UltraLite ActiveX User’s Guide*, page 38].
- ◆ **UltraLite.NET** See “User authentication” [*UltraLite.NET User’s Guide*, page 46].
- ◆ **UltraLite C++ Component** See “Authenticating users” [*UltraLite C/C++ User’s Guide*, page 35].
- ◆ **UltraLite ActiveX** See “Authenticating users” [*UltraLite ActiveX User’s Guide*, page 38].
- ◆ **Native UltraLite for Java** See “User authentication” [*Native UltraLite for Java User’s Guide*, page 54].
- ◆ **Embedded SQL** See “Authenticating users” [*UltraLite C/C++ User’s Guide*, page 85].
- ◆ **Static C++ API** See “Authenticating users” [*UltraLite C/C++ User’s Guide*, page 47].
- ◆ **UltraLite Static Java** See “Adding user authentication to your application” [*UltraLite Static Java User’s Guide*, page 33].
- ◆ **UltraLite for M-Business Anywhere** See “Authenticating users” [*UltraLite for M-Business Anywhere User’s Guide*, page 32].

## Sharing MobiLink and UltraLite user IDs

Although UltraLite and MobiLink user authentication mechanisms are separate, you may wish to provide your end users with a single user ID and password that provides both MobiLink and UltraLite user authentication. To share user IDs and passwords, store them in variables and use the same variable in the UltraLite user authentication calls and the synchronization call.

You can design your application so that, if passwords are reset at a MobiLink consolidated site, your application prompts for the new password.

---

❖ **To prompt for a new MobiLink or UltraLite password**

1. Save the user ID and password in variables.
2. Synchronize.
3. If synchronization fails because the user was not authenticated, prompt the user for a new password.
4. Update the UltraLite user's password using the appropriate function or method.
5. Update the synchronization information and synchronize again.

☞ For information on MobiLink user authentication, see [“Authenticating MobiLink Users”](#) [*MobiLink Clients*, page 9].

## Character sets in UltraLite

There are several places in UltraLite applications where character set issues can arise:

- ◆ **The UltraLite schema** The database itself has a single collating sequence (character set and sort order), which is specified when the database is created. The collating sequence determines the order of character data in indexes, the results of string comparisons, and so on.

☞ For more information, see [“UltraLite database character sets”](#) on page 43.

- ◆ **The UltraLite runtime library or component** The UltraLite component or runtime library that accesses the database uses a character set for messages and other interactions with the environment.

The runtime character sets may be Unicode or ANSI. The character set used determines how the data is stored in the database file. To manage a database, you must use a runtime of the same character set as the one used to create it.

☞ For more information, see [“UltraLite runtime character sets”](#) on page 44.

- ◆ **Synchronization** When data in the UltraLite database is synchronized with a MobiLink synchronization server, the character set used in the UltraLite database and that in the consolidated database must be consistent.

☞ For more information, see [“Synchronization and character sets”](#) on page 46.

### UltraLite database character sets

If you are using an UltraLite component, and create an UltraLite database schema using the Schema Painter, you specify the character set and collating sequence as you create the database schema.

UltraLite applications use the native multi-byte character encoding of the target platform for reasons of efficiency. When the reference database uses a different character encoding, the UltraLite application uses the default collation of the target device.

UltraLite applications use the collating sequence of the reference database if either of the following conditions is met.

- ◆ The reference database uses a single-byte character set.

- 
- ◆ The native character encoding of the target device is multi-byte, the reference database uses the same multi-byte character encoding, and the UltraLite analyzer can find a compact representation for the collation sequence used by the reference database.

For example, if you use a 932JPN reference database to build an application for the Japanese Palm Computing Platform, then the UltraLite application can inherit the collation information because the native character encoding is the same as that of the reference database. However, if you use a 932JPN reference database to build an UltraLite application for the Windows CE platform, the application will use Unicode and the default Unicode collation information.

#### Sort orders

If the character set is single byte, or the native character set of the target device is the same as the character set of the reference database, columns that are CHAR(*n*) or VARCHAR(*n*) compare and sort according to the collation sequence of the reference database.

☞ For information about creating databases, see [“Creating UltraLite databases and schemas” on page 28](#).

#### Data storage

The way that character data is stored depends not only on the collation sequence used when creating the schema, but also on the character set (ANSI or Unicode) of the UltraLite runtime library that manages the database.

☞ For more information, see [“UltraLite runtime character sets” on page 44](#).

## UltraLite runtime character sets

The character set of the UltraLite runtime library is different depending on the target operating system. The character set determines how data is exchanged with the application and also affects data storage. This section provides background information on character sets used on platforms supported by UltraLite.

#### Palm Computing Platform

Single-byte Palm Computing Platform devices use a character set based on code page 1252 (the Windows US code page). The 1252Latin1 code page is appropriate for developing applications for the Palm Computing Platform. The 1252Latin1 code page is the default Adaptive Server Anywhere collation sequence. Japanese Palm Computing Platform devices use the 932JPN character set.

#### Windows CE

The Windows CE operating system uses Unicode. UltraLite running on Windows CE also uses Unicode to store CHAR(*n*) and VARCHAR(*n*) columns. Adaptive Server Anywhere collating sequences define behavior for 8-bit ASCII character sets.

UltraLite for Windows CE uses the Adaptive Server Anywhere collating

sequence when comparing Unicode characters that have a corresponding 8-bit ASCII character in the collating sequence being used, allowing accented characters to be considered equal to and to sort with unaccented characters. Unicode characters that have no corresponding 8-bit ASCII character use a comparison of two Unicode values.

Windows desktop operating systems

The UltraLite components are all Unicode based.

The runtime library used by UltraLite embedded SQL and Static C++ applications on Windows NT/2000/XP and Windows 98 is provided in both ANSI and UNICODE versions. UltraLite versions before version 9 included only an ANSI version of the runtime library.

Compatibility of database files

The runtime or component that is used to create the database determines how characters are stored within the database. You cannot create an UltraLite database using an ANSI component or runtime library and then use that database file with a Unicode component or runtime library.

The following table lists the character set in use by UltraLite components and runtime libraries. These character sets dictate whether or not you can use a database file created by one version of UltraLite in another. In particular, note that you cannot open a database created by version 8.0.2 UltraLite for MobileVB or UltraLite ActiveX on a Windows operating system (other than Windows CE) in version 9.0 or later software.

Environment	8.0.2	9.0 and later
Windows components	ANSI	Unicode
Windows CE components	Unicode	Unicode
Windows (Native UltraLite for Java)	Unicode	Unicode
Windows 8.0.2 DLL	ANSI	N/A
Windows ( <i>ulrt9.dll</i> )	N/A	ANSI
Windows ( <i>ulrtw9.dll</i> )	N/A	Unicode
Windows ( <i>ulrtcw9.dll</i> (engine))	N/A	Unicode
Windows CE	Unicode	Unicode

Static Java

The error-handling objects **SQLException** and **SQLWarning** provide the capability for Java applications to obtain error or warning messages. By

---

default, these messages are supplied in English.

Localized error and warning messages may be obtained in a non-English language by setting the Java Locale to the appropriate language.

For example, to obtain French messages, the following code fragment might be used:

```
java.util.Locale locale = new java.util.Locale( "fr", " " );  
java.util.Locale.setDefault( locale );
```

The default Locale should be set at the start of the program. Once a message is placed in an error-handling object, the language to be used for the message is established for that execution of the program. For more information, see your Java documentation.

## Synchronization and character sets

When you synchronize, the MobiLink synchronization server always translates characters uploaded from your application database to Unicode and passes them to your consolidated database server using the Unicode ODBC API. The consolidated database server, or its ODBC driver, then performs any translation that may be required to convert them to the character encoding of your consolidated database. This second translation will always occur unless your consolidated database uses Unicode.

When information is downloaded, the consolidated database server converts the characters to Unicode. The MobiLink synchronization server then automatically translates the characters, if necessary, to suit the requirements of your UltraLite application.

When both UltraLite application and consolidated database use the same character encoding, no translation is necessary. If translation is necessary, problems can arise when multiple character codes in your UltraLite application map to a single Unicode value, or vice versa. In this event, the MobiLink synchronization server translates in a consistent manner, but the behavior is influenced by the translation mechanism within the consolidated database server.

## UltraLite database internals

In addition to storing the rows of data in each table, UltraLite stores state information about each row, and stores indexes to efficiently access the rows.

### UltraLite database files

UltraLite databases are stored in a single file that holds the tables, indexes, and all other information in the database.

The tables and indexes are stored in fixed-size pages. Database I/O operations are carried out a page at a time. The default page size is 4K. For more information, see [“Page Size connection parameter”](#) on page 83.

#### The temporary file

UltraLite maintains a temporary file to hold information while processing and to hold application state information. This file can be deleted without loss of data when UltraLite is not running.

On Windows and Windows CE, UltraLite database files typically have a name of the form *dbname.udb*. In this case, the temporary file is *dbname.~db* (with the same path). If the database name has a different form, a ‘~’ is appended to the end of the name for the temporary file. On Palm OS the temporary file is *ul\_tmp\_<creator-id>*.

#### Palm OS

UltraLite databases on the Palm OS can be stored in the Palm virtual file system for expansion cards, or they can be stored in the Palm record-based data store. For more information, see [“Choosing database storage on the Palm OS”](#) on page 190.

### How UltraLite tracks row states

Each row in an UltraLite database has a one-byte marker to keep track of the state of the row. The row states are used to control transaction processing, recovery, and synchronization.

When a delete is issued, the state of each affected row is changed to reflect the fact that it was deleted. Rolling back a delete is as simple as restoring the original state of the row.

When a delete is committed, the affected rows are not always removed from memory. If the row has never been synchronized, then it is removed. If the row has been synchronized, then it is not removed until the next synchronization confirms the delete with the consolidated database. After the next synchronization, the row is removed from memory.

Similarly, when a row is updated in an UltraLite database, a new version of the row is created. The states of the old and new rows are set so the old row is no longer visible and the new row is visible. When an update is

---

synchronized, both the old and new versions of the row are needed to allow conflict detection and resolution.

The old version of the row is deleted after synchronization. If a row is updated many times between synchronizations, only the oldest version of the row and the most recent version of the row are kept.

## UltraLite tables must have primary keys

Tables in UltraLite applications must include a primary key.

Primary keys are also required during MobiLink synchronization, to associate rows in the UltraLite database with rows in the consolidated database.

For static APIs, the UltraLite generator uses primary key columns from your reference database to generate primary key columns in the UltraLite database. If the primary key columns for any table are not included in the data required in the UltraLite database, the UltraLite generator looks for a uniqueness constraint on the table, and promotes the columns with such a constraint to a primary key in the UltraLite database. If there are no unique columns, the generator reports an error.

## Indexes in UltraLite databases

UltraLite indexes are similar to B+ trees with very small index entries.

Except for pure Java databases, each index entry is exactly two bytes, and each index page contains 256 entries. Since index pages are rarely 100% full and each index has some fixed overhead, the memory used by an UltraLite index is more than two bytes per row in the table. The overhead for each index is just over 1 K per index. Typically, UltraLite index pages on larger tables will be least 85% full.

No similar consistent rule can be given for the memory requirements of UltraLite Java databases.

## Transaction processing, recovery, and backup

UltraLite provides transaction processing. A transaction is a logical set of operations that are executed atomically; that is, either all operations in the transaction are stored in the database or none are. If a transaction is **committed**, all operations are stored in the database; if a transaction is **rolled back**, none are.

Some transactions consist of a single operation. Many programming interfaces use an **autocommit** setting to commit a transaction after each



operation. If you are using one of these interfaces, you must set autocommit to off in order to exploit multi-operation transactions. The way of setting autocommit off depends on the programming interface you are using; in most interfaces it is a property of the connection object.

#### Recovery from system failure

If an application using an UltraLite database stops running unexpectedly, the UltraLite database automatically recovers to a consistent state when the application is restarted. All transactions committed prior to the unexpected failure are present in the UltraLite database. All transactions not committed at the time of the failure are rolled back.

UltraLite does not use a transaction log to perform recovery. Instead, UltraLite uses the state byte for every row to determine the fate of a row when recovering. When a row is inserted, updated, or deleted in an UltraLite database, the state of the row is modified to reflect the operation and the connection that performed the operations. When a transaction is committed, the states of all rows affected by the transaction are modified to reflect the commit. If an unexpected failure occurs during a commit, the entire transaction is rolled back on recovery.

☞ For information about state bytes, see [“How UltraLite tracks row states” on page 47](#).

#### Backups

UltraLite provides protection against system failures, but not against media failures. If the UltraLite data store itself is corrupted, the only way to protect is through synchronization.

The best way of making a backup of an UltraLite application is to synchronize with a consolidated database. To restore an UltraLite database, start with an empty database and populate it from the consolidated database through synchronization.

---

## UltraLite database limitations

The following table lists the absolute limitations imposed by data structures in the software on the size and number of objects in an UltraLite database. In most cases, the memory, CPU, and storage device of the computer impose stricter limits.

Item	Limitation
Number of connections per database	14
Number of columns per table	65535 but limited by row size. Row size is limited to about 4 KB, so the practical limit on the number of columns per table is much smaller than this: much less than 4000 in most situations.
Number of indexes	Approximately 1000. If you set the page size to 2 KB, the maximum number of indexes per table is reduced to approximately 500.
Number of rows per database	Limited by persistent store.
Number of rows per table	65534.
Number of tables per database	Approximately 1000. If you set the page size to 2 KB, the maximum number of tables is reduced to approximately 500.
Number of tables referenced per transaction	No limit
Row size	Approximately 4 KB (compressed). LONG VARCHAR and LONG BINARY values are stored separately, and are in addition to the 4 KB limit.
File-based persistent store	2 GB file or OS limit on file size

Item	Limitation
Palm Computing Platform database size	128 Mb (Primary storage) 2 GB (expansion card file system)

☞ For other limitations, see [“Overview of SQL support in UltraLite” on page 142](#).

## Adaptive Server Anywhere features not available in UltraLite

The following Adaptive Server Anywhere features are not available in UltraLite databases:

- ◆ **Cascading updates and deletes** Some applications rely on declarative referential integrity to implement business rules. These features are not available in UltraLite databases except during synchronization downloads, when updates and deletes are cascaded automatically.  
  
Any attempt to delete a primary key that has a corresponding value in a foreign key fails with an error. Any attempt to update a primary key value when foreign keys reference the original value also fails.
- ◆ **Check constraints** You cannot include table or column check constraints in an UltraLite database.
- ◆ **Computed columns** You cannot include computed columns in an UltraLite database.
- ◆ **Global temporary tables** The temporary aspect of global temporary tables is not recognized by UltraLite. They are treated as if they were permanent base tables, which you should use instead.
- ◆ **Declared temporary tables** You cannot declare a temporary table within an UltraLite application.
- ◆ **Stored procedures** You cannot call stored procedures or user-defined functions in an UltraLite application.
- ◆ **Functions** Not all SQL functions are available for use in UltraLite applications.

Use of an unsupported function gives a `Feature not available in UltraLite` error.

☞ For a list of supported functions, see [“UltraLite SQL functions” on page 148](#).

- 
- ◆ **Triggers** Triggers are not supported in UltraLite databases.
  - ◆ **System table access** There are no system tables in an UltraLite database.
  - ◆ **System functions** You cannot use Adaptive Server Anywhere system functions, including property functions, in UltraLite applications.
  - ◆ **Java in the database** You cannot include Java methods in your queries or make any other use of Java in the database.
  - ◆ **Timestamp columns** You cannot use Transact-SQL timestamp columns in UltraLite databases. Transact-SQL timestamp columns are created with the following default:

`DEFAULT TIMESTAMP`

You can use columns created as follows:

`DEFAULT CURRENT TIMESTAMP`

There is a behavior difference between the two: a `DEFAULT CURRENT TIMESTAMP` column is not automatically updated when the row is updated, while a `DEFAULT TIMESTAMP` column is automatically updated. You must explicitly update columns created with `DEFAULT CURRENT TIMESTAMP` if you wish the column to reflect the latest update time.

## SQL limitations

If you are using dynamic SQL in your UltraLite application, the range of SQL available is less than in Adaptive Server Anywhere.

☞ For more information, see [“Dynamic SQL” on page 159](#).

UltraLite applications using static interfaces can use a wider range of SQL, but the following SQL features are not supported.

- ◆ **Dynamic SQL** Dynamic SQL is not available to applications using static interfaces.
- ◆ **SAVEPOINT statement** UltraLite databases support transactions, but not savepoints within transactions.
- ◆ **SET OPTION statement** UltraLite databases do support a set of options, but you cannot use the `SET OPTION` statement in an UltraLite application to change option settings.

For UltraLite components, you can set options in the database schema file, either by setting them in the reference database and using `ulinit`, or from the UltraLite Schema Painter, or in an XML document holding the schema.

☞ For more information about UltraLite database options, see [“UltraLite database options” on page 33](#).

- ◆ **Schema modification** To modify the schema of a UltraLite database from a static interface application, you must build a new version of your application.

To modify the schema of an UltraLite database from an UltraLite component, you can use schema modification statements or you can deploy an updated schema file.

☞ For more information, see [“Upgrading UltraLite database schemas” on page 54](#).

---

## Upgrading UltraLite database schemas

If you develop a new version of an UltraLite application, you may wish to alter the database schema. You can deploy an upgraded UltraLite schema and maintain the data in existing end-user databases subject to some restrictions. This feature is not available to UltraLite applications built using the Static Java API.

### **Caution**

*Schema upgrade is not a reversible process. If an error occurs during schema upgrade, it is possible to be left with an unusable database. Ensure that your application has backed up all changes (by synchronizing or by copying the database file) before upgrading the schema of databases containing important data.*

The mechanism for deploying an upgraded schema depends on whether you use an UltraLite component (which requires a new schema file) or whether you use a static interface (in which case the schema definition is held in the generated application code).

### ❖ **To deploy a new schema (UltraLite components)**

1. Create a new schema file.

You can create your new schema using the Schema Painter or using `ulinit`.

If you use the schema painter, prepare the schema for deployment by defining the mapping between old and new names. This mapping is used to avoid losing data during the migration.

- a. In the schema painter, choose **File** ► **Prepare Schema for Deployment**.
- b. Specify the mappings between old table or column names and their equivalent in the new database schema.

If you use `ulinit` to create the new schema, you lose any data held in renamed columns—the schema is interpreted as dropping the old column and creating a new column.

2. Apply the schema file to the existing database.

The components expose a database schema as a `DatabaseSchema` or `ULDatabaseSchema` object. You obtain a schema object using the `DatabaseSchema` property on the `Connection` or `ULConnection` object.

Use the `ApplyFile` method on the `ULDatabaseSchema` object to apply the new schema.

❖ **To deploy a new schema (embedded SQL and Static C++ API)**

1. Modify the schema in your reference database.
2. Create the new version of your application.
3. Ensure that your application calls `ULEnableGenericSchema()`.

When a new UltraLite application built with a static interface is deployed to a device, UltraLite by default recreates an empty database, losing any data that was in the database before the new application was deployed. If you call `ULRegisterSchemaUpgradeObserver`, the existing database is instead upgraded to the schema of the new application. Applications that upgrade a database schema must call this function.

☞ See “[ULRegisterSchemaUpgradeObserver function](#)” [*UltraLite C/C++ User’s Guide*, page 216].

4. The schema is upgraded automatically when the new application is applied.

☞ For information on how the schema upgrade happens, see “[How schema upgrade works](#)” on page 56.

## Monitoring schema upgrades

Database schema upgrades can be a time-consuming process. An upgrade observer lets you display progress information to the user, and allows the user to cancel the upgrade during the initial step of the process.

The mechanism depends on the interface you are using.

- ◆ **UltraLite for MobileVB** Implement handlers for the `OnSchemaUpgradeStateChange` and `OnSchemaUpgradeProgress` events.

☞ See “[OnSchemaUpgradeProgress event](#)” [*UltraLite for MobileVB User’s Guide*, page 95] and “[OnSchemaUpgradeStateChange event](#)” [*UltraLite for MobileVB User’s Guide*, page 96].

- ◆ **UltraLite.NET** Implement the `SchemaUpgradeListener` interface and supply the object to `DatabaseSchema.ApplyFile()`. During the schema upgrade, your `SchemaUpgradeListener.SchemaUpgrading` method is invoked.

☞ See `SchemaUpgradeListener` in the API Reference.

- ◆ **Native UltraLite for Java** Implement the `SchemaUpgradeListener` interface and supply the object to `DatabaseSchema.applyFile()`. During the schema upgrade, your `SchemaUpgradeListener.schemaUpgrading` method is invoked.

☞ See `ianywhere.native_ultralite.SchemaUpgradeListener` in the API Reference.

- 
- ◆ **UltraLite C/C++ interfaces** Implement and register a callback function that handles the schema upgrade events.

☞ See “ULRegisterSchemaUpgradeObserver function” [*UltraLite C/C++ User’s Guide*, page 216] and “Callback function for ULRegisterSchemaUpgradeObserver” [*UltraLite C/C++ User’s Guide*, page 206].

- ◆ **UltraLite ActiveX** This feature is not available for UltraLite ActiveX.
- ◆ **UltraLite for M-Business Anywhere** This feature is not available for UltraLite for M-Business Anywhere.

## How schema upgrade works

### Back up before upgrading

It is strongly recommended that you back up your data before attempting a schema upgrade, either by copying the database file or by synchronizing.

The schema upgrading process relies on matching names in the old and new schema. If a row in the database is incompatible with the new schema, that row is deleted from the database. In general, adding constraints to tables that have data in them or carrying out unpredictable column conversions may result in lost rows.

The schema upgrade proceeds as follows:

1. Any old tables not in the new schema are dropped.
2. Any new tables are created.
3. For any table that exists in both the old and new schema, columns are added and dropped as needed. If a new column is not nullable and has no default value, it is filled with zeros (numeric data types), the empty string (character data types), and an empty binary value.
4. Columns whose properties have changed are then modified.

### Caution

*If an error occurs during conversion for any row, that row is dropped and the SQL warning `SQLLE_ROW_DROPPED_DURING_SCHEMA_UPGRADE` is generated.*

5. Indexes and constraints are rebuilt. This step may also result in rows being dropped if, for example, an index is redefined as UNIQUE but has duplicate values.



## Upgrading UltraLite software

On Windows operating systems other than Windows CE, if you are using UltraLite for MobileVB or UltraLite ActiveX, you cannot open an UltraLite database file created using 8.0.2 software. For more information, see “UltraLite behavior changes” [*What’s New in SQL Anywhere Studio*, page 96].

☞ For compatibility of UltraLite database files with previous releases, see “UltraLite runtime character sets” on page 44.

---

## The UltraLite runtime

The **UltraLite runtime** manages UltraLite databases and synchronization. With the exception of the Static Java API, the UltraLite runtime for each target platform is based on a single code base. The UltraLite runtime is provided in several different forms:

- ◆ **Components** The UltraLite runtime is linked into each component.
- ◆ **Static library** For static interfaces, the UltraLite runtime is provided as a static library that you link into your application. There is then no separate file to deploy.
- ◆ **Dynamic library** For C/C++ Windows and Windows CE applications, the UltraLite runtime is provided as a dynamically linked library (DLL).
- ◆ **Separate executable** For Windows and Windows CE a separate executable called the **UltraLite engine** is provided. The UltraLite engine is the only version of the runtime to support connections from multiple applications. Each such application must link against a client library when using the UltraLite engine.

☞ For more information, see [“Using the UltraLite engine” on page 61](#).

## Understanding concurrency in UltraLite

UltraLite databases may receive multiple concurrent requests. In order to design applications that handle concurrent requests properly you should understand how UltraLite manages concurrency in the database.

### Concepts

It is helpful to separate several concepts when thinking about concurrent database access. These concepts are ordered from high-level to low-level:

- ◆ **Applications** The UltraLite engine can respond to requests from multiple separate applications. Other versions of the UltraLite runtime permit only a single application to connect to a database at a time.  
☞ For more information, see [“The UltraLite runtime” on page 58](#).
- ◆ **Threads** The UltraLite runtime supports multi-threaded applications. A single application may be written to use multiple threads, each of which may connect to the database.  
☞ For more information, see [“Threading in UltraLite applications” on page 60](#).
- ◆ **Connections** Even a single-threaded application may open multiple connections to a database. In any case, individual connections can employ only a single thread.

- ◆ **Transactions** Each connection can have a single transaction in progress at any one time. Transactions may consist of a single request or multiple requests. Data modifications made during a transaction are not permanent in the database until the transaction is committed. Either all data modifications made in a transaction are committed, or all are rolled back.
- ◆ **Requests** A transaction consists of one or more requests. A request may be a query that reads data, or an insert, update, or delete that modifies data, or a synchronization.
- ◆ **The current row** When an application is working with the result set of a query, UltraLite maintains a pointer to the **current row** within the result set. In some interfaces, this current row is tracked explicitly using a cursor (a pointer to a position in a result set). In others, the application uses methods on a result set object or table object to identify and change the current row. Such methods use a cursor “under the covers”.

**Multiple databases**

The UltraLite runtime can manage a maximum of four databases at any one time. A single UltraLite application may open multiple connections to separate databases. No concurrency issues arise from such applications, as the data in each database is independent.

**Locking**

When a transaction changes a row, UltraLite locks that row until the transaction is committed or rolled back. The lock prevents other transactions from changing the row, although they can still read the row. An attempt to change a locked row sets error SQLCODE SQLE\_LOCKED, while an attempt to change a deleted row sets the error SQLE\_NOTFOUND. Your applications should check the SQLCODE value after attempting to modify data.

**Re-reading rows**

To understand how locking and concurrency is managed, it helps to consider two connections, A and B, each with their own transaction.

As connection A works with the result set of a query, UltraLite **fetches** a copy of the current row into a buffer. If A modifies the current row, it changes the copy in the buffer. The copy in the buffer is written back into the database when connection A calls an Update method or closes the result set. At that time, a write lock is placed on the row to prevent other transactions from modifying it. The change to the database is not permanent until connection A commits the transaction.

Reading or fetching a row does not lock the row. If connection A fetches but does not modify a row, connection B can still modify the row.

If connection B does modify the current row, the row becomes locked. Connection A can then not modify the current row. If connection A fetches

---

the current row again, and the row has been deleted, connection A gets the next row in the result set. If the row has been modified, connection A gets the latest copy of the row. If the columns of the index used by connection A have been modified, connection A sees the change as a delete followed by an insert, and so gets the next row in the result set.

## Synchronization

Synchronization behaves as a separate connection. During the upload phase, UltraLite applications can access UltraLite databases in a read-only fashion. During the download phase, read-write access is permitted but if an application changes a row that the download then attempts to change, the download will fail and roll back. You can disable access to data during synchronization by setting the Disable Concurrency synchronization parameter.

☞ For more information, see “[Disable Concurrency synchronization parameter](#)” [*MobiLink Clients*, page 319].

## Threading in UltraLite applications

The UltraLite runtime is thread safe. You can develop multi-threaded applications as long as the development tool and the target platform both support multi-threaded applications. The exceptions are as follows:

- ◆ You cannot develop multi-threaded applications using UltraLite for MobileVB, because of limitations in the underlying development tools.
- ◆ You cannot develop multi-threaded applications using UltraLite ActiveX or UltraLite for M-Business Anywhere, because of limitations in eMbedded Visual Basic, JScript, and JavaScript.
- ◆ You cannot develop multi-threaded applications for the Palm OS, because of limitations in the operating system.
- ◆ From static interfaces, the same query cannot be executed more than once at a time. As a result, you cannot access more than one instance at a time of a result set for a given query.

Each thread must manage its own set of objects, including the Database Manager object or its equivalents in the static interfaces (ULData, SQLCA). The database manager object can be passed to a thread rather than being initialized on the thread, but all management of data must be carried out on an individual thread.

## Static Java API

The UltraLite runtime used by the Static Java API permits access to a single database by a single application.

The UltraLite runtime is thread safe. Users of the Sun Java VM must use version 1.2 or later to run multi-threaded UltraLite applications. Users of the

Jeode VM on Pocket PC and the IBM Java VM can run multi-threaded UltraLite applications even though these VMs are based on JDK 1.1.8.

The entire UltraLite runtime is treated as a single critical section, only allowing one thread to enter it at a time.

☞ For more information, see “Using the UltraLite JdbcDatabase.connect method” [*UltraLite Static Java User’s Guide*, page 26].

## Using the UltraLite engine

You can deploy the UltraLite runtime in the form of the UltraLite engine as an alternative to deploying the UltraLite runtime as a library or in a component.

The UltraLite engine acts as a database server for UltraLite databases. It is provided for Windows and Windows CE platforms, as a separate executable. The advantage of the UltraLite engine is that it provides same-machine access to the database from multiple applications. The disadvantage is that the UltraLite engine requires more system resources than other versions of the UltraLite runtime and may yield lower performance.

### ❖ To use the UltraLite engine

1. Deploy the client version of your component.

The client versions are as follows. Paths are relative to your SQL Anywhere installation:

- ◆ **UltraLite.NET (Windows CE)**  
*UltraLite\UltraLite.NET\ce\arm\ulnetclient9.dll*
- ◆ **UltraLite.NET (Windows XP)** *win32\ulnetclient9.dll*
- ◆ **Native UltraLite for Java (Windows CE)**  
*UltraLite\NativeUltraLiteForJava\ce\arm\julclient9.dll*
- ◆ **Native UltraLite for Java (Windows XP)**  
*UltraLite\NativeUltraLiteForJava\win32\julclient9.dll*
- ◆ **C++ Component (Windows CE)** *UltraLite\ce\arm\lib\ulrtc.lib*  
(static library)
- ◆ **C++ Component (Windows XP)**  
*UltraLite\win32\386\lib\ulimpcw.lib* (import library) and  
*UltraLite\win32\386\ulrtcw9.dll* (dynamic library)
- ◆ **UltraLite for M-Business Anywhere (Windows CE)**  
*UltraLite\UltraLiteForMBusinessAnywhere\ce\arm\ulpodclient9.dll*.  
The client library must be deployed under the *AvantGo\Pods* directory.

---

- ◆ **UltraLite for M-Business Anywhere (Windows XP)**

*UltraLite\UltraLiteForMBusinessAnywhere\win32\386\ulpodclient9.-dll*. The client library must be deployed under the *AvantGo\Pods* directory.

- ◆ **UltraLite ActiveX** Not available.

- ◆ **UltraLite for MobileVB** Not available.

2. Specify the runtime type in the DatabaseManager constructor.

This step is required for Native UltraLite for Java and for UltraLite.NET only.

## Starting the engine

You can start the UltraLite engine in the following ways.

- ◆ **Manually starting the UltraLite engine** Enter the following command at a command prompt.

```
dbuleng9
```

- ◆ **Let your application start the UltraLite engine** When you deploy your UltraLite application using the client version of your UltraLite component, it starts the UltraLite engine on the first connection attempt. If an engine is already running, it uses that engine to connect to the database.

To specify the location of *dbuleng9.exe*, use the *StartLine* connection parameter when connecting to the database for the first time. If the *StartLine* parameter is not provided, the client looks for *dbuleng9.exe* in the *\Windows*, root (*\*), and *\UltraLiteDB* directories in that order on Windows CE and in the *win32* subdirectory of your SQL Anywhere installation on other Windows operating systems.

To connect to a database using the UltraLite engine, you must supply the user ID (*uid*) and password (*pwd*) connection parameters.

You can stop the UltraLite engine manually using the *dbulstop* utility. Alternatively, you can let the application stop the engine. The last application to disconnect from the database stops the UltraLite engine automatically.

---

## CHAPTER 4

# Connection Parameters

About this chapter

This chapter provides a reference for the parameters that establish and describe connections from client applications to a database.

Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Overview</a>	64
<a href="#">Database Identification parameters</a>	68
<a href="#">Open Connection parameters</a>	73
<a href="#">Database Schema parameters</a>	78
<a href="#">Additional connection parameters</a>	82

---

## Overview

You must supply connection parameters for an application to connect to an UltraLite database. The connection parameters must specify the database to which the connection is to be established, usually by providing a database filename and path.

As an application may be compiled for more than one platform, a separate parameter for each target platform is available to identify the database. If user authentication is enabled, the connection parameters must also specify a user name and password.

☞ For more information, see [“Database Identification parameters” on page 68](#), and [“Open Connection parameters” on page 73](#).

UltraLite databases are often created by the application itself, and characteristics of the database are defined by connection parameters. The connection parameters must include a schema file (for UltraLite components) as well as optional parameters to adjust database features. If you are using embedded SQL, the static C++ API, or the Static Java API, the database is created from information already stored in the application and no schema file is needed.

☞ For more information, see [“Database Schema parameters” on page 78](#), and [“Additional connection parameters” on page 82](#).

All connection parameters are case insensitive.

The following table lists the available connection parameters. Database Schema parameters, used only when creating a database are marked with an asterisk (\*).

Parameter	Description
Additional Parm*s	A placeholder for additional connection parameters. See <a href="#">“Additional Parm*s connection parameter” on page 68</a>
Cache Size	Defines the size of the cache. See <a href="#">“Cache Size connection parameter ” on page 73</a> .
Connection Name	Specifies a connection name. See <a href="#">“Connection Name connection parameter” on page 74</a> .
Database On CE	The path and filename of the UltraLite database file to which you want to connect on Windows CE. See <a href="#">“Database On CE connection parameter” on page 69</a> .



Parameter	Description
Database On Desktop	The path and filename of the UltraLite database file to which you want to connect. See <a href="#">“Database On Desktop connection parameter”</a> on page 70.
Database On Palm	An identifier for the UltraLite database on Palm OS. See <a href="#">“Database On Palm connection parameter”</a> on page 71.
Encryption Key	An encryption key for the database. See <a href="#">“Encryption Key connection parameter”</a> on page 75.
Obfuscate*	Apply a simple encryption scheme to the database. See <a href="#">“Obfuscate connection parameter”</a> on page 83.
Page Size	The database page size. See <a href="#">“Page Size connection parameter”</a> on page 83.
Palm Allow Backup	Controls HotSync backup behavior on Palm OS devices. See <a href="#">“Palm Allow Backup parameter”</a> on page 84.
Password	A password for the user. See <a href="#">“Password connection parameter”</a> on page 76.
Reserve Size	Defines the reserve size. See <a href="#">“Reserve Size connection parameter”</a> on page 84.
Schema On CE*	The path and filename of the UltraLite schema on Windows CE. See <a href="#">“Schema On CE connection parameter”</a> on page 78.
Schema On Desktop*	The path and filename of the UltraLite schema. See <a href="#">“Schema On Desktop connection parameter”</a> on page 79.
Schema On Palm*	The UltraLite schema for the Palm OS. See <a href="#">“Schema On Palm connection parameter”</a> on page 80.
User ID	The user ID with which you connect to the database. See <a href="#">“User ID connection parameter”</a> on page 76.
VFS On Palm*	Identifies the Palm card as using the virtual file system. See <a href="#">“VFS On Palm parameter”</a> on page 81.

## Specifying file paths

Filenames and paths in connection parameters are subject to the following requirements, depending on the UltraLite Component you are using:

Target platform	Requirement
Java	All backslashes must be escaped. For example, "file_name=\\UltraLite\\MyFile.udb".
Windows CE	Paths are absolute.
Windows	Paths may be absolute or relative.

## Specifying connection parameters

Each connection parameter can be specified in the following ways:

- ◆ **As a property of a Connection Parameters object** The following interfaces provide a connection parameters object. This object has properties that are individual connection parameters.
  - UltraLite for MobileVB. You can specify connection parameters by adding an UltraLite control to a form and specifying its properties in the Properties window.
  - UltraLite ActiveX
  - Native UltraLite for Java
  - UltraLite.NET. You can specify connection parameters by adding an UltraLite.NET control to a form and specifying its properties in the Properties window.
- ◆ **In the AdditionalParms property** The interfaces that supply a connection parameters object include an Additional Parms property. This property takes a connection string as its value.
- ◆ **As a keyword in a connection string** UltraLite interfaces can supply a connection string when connecting. The keywords in the connection string are individual connection parameters.
- ◆ **In the UL\_STORE\_PARMS macro** Embedded SQL and the static C++ API both use the UL\_STORE\_PARMS macro to hold connection parameters that affect database file. The parameters are used only on the initial connection attempt, when the database is created. The UL\_STORE\_PARMS macro takes a connection string.

For example, the following definition sets a connection parameter.

```
#define UL_STORE_PARMS    UL_TEXT( "reserve_size=2m" )
```

Where possible, it is recommended that you use the Connection Parameters object. It provides easier checking and a more systematic interface than using a connection string.

Connection strings and connection parameters

Some less commonly used parameters can be specified only in a connection string. Depending on the interface, the connection string can be supplied in the `AdditionalParams` property, in the `UL_STORE_PARMS` macro, or in an `Open` method that takes a connection string as argument.

If a parameter is specified both in a property and in a connection string, the value in the property takes precedence.

**Troubleshooting tip**

UltraLite ignores unrecognized connection string parameters. As a result, spelling mistakes may not be immediately obvious.

---

## Database Identification parameters

The connection parameters in this section are used to identify the UltraLite database. At least one of these parameters must be specified on each connection attempt.

Database Identification parameters are also used when dropping (deleting) a database.

For more information

- ◆ **UltraLite for MobileVB** See “OpenConnection method” [*UltraLite for MobileVB User’s Guide*, page 108], and “OpenConnectionWithParms method” [*UltraLite for MobileVB User’s Guide*, page 109].
- ◆ **UltraLite ActiveX** See “OpenConnection method” [*UltraLite ActiveX User’s Guide*, page 107], and “OpenConnectionWithParms method” [*UltraLite ActiveX User’s Guide*, page 107].
- ◆ **Native UltraLite for Java** See **ianywhere.native\_ultralite.DatabaseManager** in the Native UltraLite for Java API Reference.
- ◆ **UltraLite.NET** See “ULDatabaseManager class” [*UltraLite.NET User’s Guide*, page 137] (iAnywhere.Data.UltraLite namespace) or “DatabaseManager class” [*UltraLite.NET User’s Guide*, page 399] (iAnywhere.UltraLite namespace).
- ◆ **UltraLite C++ Component** See “Class UltraLite\_DatabaseManager” [*UltraLite C/C++ User’s Guide*, page 249].
- ◆ **UltraLite for M-Business Anywhere** See “Method openConnection” [*UltraLite for M-Business Anywhere User’s Guide*, page 77], and “Method openConnectionWithParms” [*UltraLite for M-Business Anywhere User’s Guide*, page 77].
- ◆ **UltraLite for Embedded SQL** See “Initializing the SQL Communications Area” [*UltraLite C/C++ User’s Guide*, page 64].
- ◆ **UltraLite Static C++ API** See “Connecting to a database” [*UltraLite C/C++ User’s Guide*, page 45].
- ◆ **UltraLite Static Java** See “Connecting to the database using JDBC” [*UltraLite Static Java User’s Guide*, page 27].

### Additional Parms connection parameter

Function	Permits additional connection parameters to be specified.
Syntax	

Interface	Connection parameter
UltraLite for MobileVB	AdditionalParms
UltraLite ActiveX	AdditionalParms
UltraLite.NET	AdditionalParms
Native UltraLite for Java	additionalParms
UltraLite for M-Business Anywhere	additionalParms

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#) and [“Additional connection parameters” on page 82](#).

Usage	Some less commonly used connection parameters do not have properties associated with them in the UltraLite components. These parameters can be specified as a connection string in AdditionalParms.
Values	A connection string.
Default	None.
See also	<a href="#">“Specifying connection parameters” on page 66</a>

## Database On CE connection parameter

Function The path and filename of the UltraLite database file to which you want to connect on Windows CE.

Syntax

Interface	Connection parameter
UltraLite for MobileVB	DatabaseOnCE
UltraLite ActiveX	DatabaseOnCE
UltraLite.NET	DatabaseOnCE
Native UltraLite for Java	databaseOnCE
UltraLite for M-Business Anywhere	databaseOnCE
Connection string	<b>ce_file</b>

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Values	<i>String</i>
Default	If Database On CE is not specified then the value for Database On Desktop is

---

used.

If neither is specified then the default value of `\UltraLiteDB\ulstore.udb` is used. It is recommended that you explicitly specify the parameter, and not rely on default behavior.

Description

When creating a database, this parameter names the new database file.

When opening a connection to an existing database, the parameter identifies the database.

- ◆ If the filename does not include an extension, the file of extension `.udb` is presumed.
- ◆ The full path of the file must be specified. No substitutions are performed on this value.
- ◆ The schema file is not required if a `.udb` file already exists.
- ◆ Database On CE is required if you use a database with any name other than the default.
- ◆ You must ensure that this directory exists when the connection parameter is used. UltraLite does not create the directory automatically.

Example

To create and connect to the sample database, `udemo.udb`:

```
"schema_file=MyOrders.usm;CE_FILE=udemo.udb"
```

See also

[“Specifying file paths” on page 65](#)

## Database On Desktop connection parameter

Function

The database file to which you want to connect in the desktop development environment.

Syntax

Interface	Connection parameter
UltraLite for MobileVB	DatabaseOnDesktop
UltraLite ActiveX	DatabaseOnDesktop
UltraLite.NET	DatabaseOnDesktop
Native UltraLite for Java	databaseOnDesktop
UltraLite for M-Business Anywhere	databaseOnDesktop
Connection string	{ <b>file_name</b>   <b>DBF</b> }

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Values

*String*

Default

The default value is *ulstore.udb*.

It is recommended that you explicitly specify the parameter, and not rely on default behavior.

Description

When creating a database, this parameter names the new database file.

When opening a connection to an existing database, the parameter identifies the database.

If the filename does not include an extension, the file of extension *.udb* is assumed.

Example

◆ To create and connect to the sample database, *udemo.udb*, installed in the directory *c:\mydb*, use the following connection string:

```
"schema_file=MyOrders.usm;DBF=c:\mydb\udemo.udb"
```

See also

[“Specifying file paths” on page 65](#)

## Database On Palm connection parameter

Function

The Palm creator ID of the database to which you want to connect.

Syntax

Interface	Connection parameter
UltraLite for MobileVB	DatabaseOnPalm
UltraLite for M-Business Anywhere	databaseOnPalm
Connection string	<b>palm_db</b>

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Values

*String*

Default

For C++, the creator ID of the application. For MobileVB applications, there is no default.

Description

For many applications, there is no need to supply a Database On Palm connection parameter. From UltraLite for MobileVB, supply a DatabaseOnPalm connection parameter that matches the creator ID of your application.

---

On first connecting to a database, UltraLite creates a database which is named `ul_udb_creator-id` and which has a creator ID of `creator-id`, where `creator-id` is the value of the Database On Palm connection parameter, or (if that parameter is not supplied) is the creator ID of the application.

See also

- ◆ [“Specifying file paths” on page 65](#)
- ◆ [“Understanding the Palm Creator ID” on page 191](#)



## Open Connection parameters

Open Connection parameters are used, together with Database Identification parameters, by `OpenConnection` methods and `OpenConnectionWithParms` methods.

For more information

- ◆ **UltraLite for MobileVB** See “[OpenConnection method](#)” [*UltraLite for MobileVB User’s Guide*, page 108], and “[OpenConnectionWithParms method](#)” [*UltraLite for MobileVB User’s Guide*, page 109].
- ◆ **UltraLite ActiveX** See “[OpenConnection method](#)” [*UltraLite ActiveX User’s Guide*, page 107], and “[OpenConnectionWithParms method](#)” [*UltraLite ActiveX User’s Guide*, page 107].
- ◆ **Native UltraLite for Java** See `ianywhere.native_ultralite.DatabaseManager` in the Native UltraLite for Java API Reference.
- ◆ **UltraLite.NET** See “[ULConnectionParms class](#)” [*UltraLite.NET User’s Guide*, page 100] (`Anywhere.Data.UltraLite` namespace) or “[ConnectionParms class](#)” [*UltraLite.NET User’s Guide*, page 362] (`Anywhere.UltraLite` namespace).
- ◆ **UltraLite C++ Component** See “[OpenConnection Function](#)” [*UltraLite C/C++ User’s Guide*, page 251].
- ◆ **UltraLite for M-Business Anywhere** See “[Method openConnection](#)” [*UltraLite for M-Business Anywhere User’s Guide*, page 77], and “[Method openConnectionWithParms](#)” [*UltraLite for M-Business Anywhere User’s Guide*, page 77].
- ◆ **UltraLite for Embedded SQL** See “[Authenticating users](#)” [*UltraLite C/C++ User’s Guide*, page 85].
- ◆ **UltraLite Static C++ API** See “[Authenticating users](#)” [*UltraLite C/C++ User’s Guide*, page 47].
- ◆ **UltraLite Static Java** See “[Adding user authentication to your application](#)” [*UltraLite Static Java User’s Guide*, page 33].

## Cache Size connection parameter

Function

Defines the size of the database cache.

Syntax

Interface	Connection parameter
Connection string	<code>cache_size</code>

---

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Usage	Used when you configure a database. Use k or K, m or M to denote kilobytes or megabytes, respectively.
Values	The minimum cache size is 4K.
Default	The default is 16 x page_size. Actual value used is rounded down to the nearest multiple of page_size.
Description	<p>Defines the size of the cache. You can specify the size in units of bytes. Use the suffix k or K to indicate units of kilobytes and use the suffix M or m to indicate megabytes</p> <p>The default cache size is sixteen pages. Using the default page size of 4 K, the default cache size is therefore 64 K. The minimum cache size is platform dependent.</p> <p>The default cache size is conservative. If your testing shows the need for better performance, you should increase the cache size.</p> <p>Increasing the cache size beyond the size of the database itself provides no performance improvement. Also, large cache sizes may interfere with the number of other applications you can use.</p> <p>On the Palm Computing Platform, the parameter applies only to virtual file system (VFS) databases. The cache itself resides in record storage, not VFS storage.</p>
Example	For example, the following string sets the cache size to 128 K.

```
"cache_size=128k"
```

## Connection Name **connection parameter**

Function	Specifies a name for the connection. This is only needed if you create more than one connection to the database.
Syntax	

Interface	Connection parameter
UltraLite for MobileVB	ConnectionString
UltraLite ActiveX	ConnectionString
UltraLite.NET	ConnectionString
Native UltraLite for Java	connectionName
UltraLite for M-Business Anywhere	connectionName
Connection string	<b>con</b>

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Usage

The Connection Name parameter is global to the application.

## Encryption Key connection parameter

Function

An encryption key for the database. You can define an encryption key for your UltraLite database when CreateDatabase is called.

Syntax

Interface	Connection parameter
UltraLite for MobileVB	EncryptionKey
UltraLite ActiveX	EncryptionKey
UltraLite.NET	EncryptionKey
Native UltraLite for Java	encryptionKey
UltraLite for M-Business Anywhere	encryptionKey
Connection string	{ <b>key</b>   <b>dbkey</b> }

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Values

*String*

Default

No key is provided.

Description

Used only when you create a database.

If a database is created using an encryption key, the database file is strongly encrypted using the AES 128-bit algorithm, which is the same algorithm used to encrypt Adaptive Server Anywhere databases. Use of strong encryption provides security against skilled and determined attempts to gain

---

access to the data, but may have a significant performance impact.

Example

```
"schema_file=MyOrders.usm;KEY=MyKey"
```

See also

[“Encrypting UltraLite databases” on page 36](#)

## Password connection parameter

Function

A password for the user. Passwords are case insensitive if the database is case insensitive and case sensitive if the database is case sensitive.

Syntax

Interface	Connection parameter
UltraLite for MobileVB	Password
UltraLite ActiveX	Password
UltraLite.NET	Password
Native UltraLite for Java	password
UltraLite for M-Business Anywhere	password
Connection string	{ <b>password</b>   <b>PWD</b> }

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Usage

Anywhere

Values

*String*

Default

SQL

Description

Every user of a database has a password. The password must be supplied for the user to be allowed to connect to the database.

The Password (PWD) connection parameter is not encrypted.

Example

◆ The following connection string fragment supplies the user ID DBA and password SQL.

```
"UID=DBA;PWD=SQL;schema_file=MyOrders.usm"
```

## User ID connection parameter

Function

The user ID with which you log on to the database. The value must be an authenticated user for the database. User ID's are case-insensitive if the database is case-insensitive and case sensitive if the database is case sensitive.

Databases are created with a single authenticated user DBA whose initial password is SQL. By default, connections are opened using the UID=DBA and the PWD=SQL. To disable the default user, use

```
connection.revokeConnectionFrom.
```

To add a user or change a user's password, use

```
connection.grantConnectTo.
```

## Syntax

Interface	Connection parameter
UltraLite for MobileVB	UserID
UltraLite ActiveX	UserID
UltraLite.NET	UserID
Native UltraLite for Java	userID
UltraLite for M-Business Anywhere	userID
Connection string	{ <b>userid</b>   <b>UID</b> }

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Usage	Anywhere
Values	<i>String</i>
Default	DBA
Description	You must always supply a user ID when connecting to a database, unless you leave the database using the default user ID and password of DBA and SQL.
Example	<ul style="list-style-type: none"> <li>◆ The following connection string fragment supplies the user ID DBA and password SQL:           <pre>"schema_file=MyOrders.usm;uid=DBA;pwd=SQL"</pre> </li> </ul>

---

## Database Schema parameters

The keywords in this section are used to specify a schema for an UltraLite database.

For more information

- ◆ **UltraLite for MobileVB** See “CreateDatabase method” [*UltraLite for MobileVB User’s Guide*, page 105], and “CreateDatabaseWithParms method” [*UltraLite for MobileVB User’s Guide*, page 107].
- ◆ **UltraLite ActiveX** See “CreateDatabase method” [*UltraLite ActiveX User’s Guide*, page 101], and “CreateDatabaseWithParms method” [*UltraLite ActiveX User’s Guide*, page 102].
- ◆ **Native UltraLite for Java** See **ianywhere.native\_ultralite.DatabaseManager** in the Native UltraLite for Java API Reference.
- ◆ **UltraLite.NET** See “OpenWithCreate method” [*UltraLite.NET User’s Guide*, page 93] (iAnywhere.Data.UltraLite namespace) or “CreateDatabase method” [*UltraLite.NET User’s Guide*, page 401] (iAnywhere.UltraLite namespace).
- ◆ **UltraLite C++ Component** See “CreateAndOpenDatabase Function” [*UltraLite C/C++ User’s Guide*, page 250].
- ◆ **UltraLite for M-Business Anywhere** See “Method createDatabase” [*UltraLite for M-Business Anywhere User’s Guide*, page 75], and “Method createDatabaseWithParms” [*UltraLite for M-Business Anywhere User’s Guide*, page 76].
- ◆ **UltraLite for embedded SQL** See “Macros and compiler directives for UltraLite C/C++ applications” [*UltraLite C/C++ User’s Guide*, page 221].
- ◆ **UltraLite static C++** See “Macros and compiler directives for UltraLite C/C++ applications” [*UltraLite C/C++ User’s Guide*, page 221].

### Schema On CE connection parameter

Function

To identify the schema filename deployed to Windows CE.

Syntax

Interface	Connection parameter
UltraLite for MobileVB	SchemaOnCE
UltraLite ActiveX	SchemaOnCE
UltraLite.NET	SchemaOnCE
Native UltraLite for Java	schemaOnCE
UltraLite for M-Business Anywhere	schemaOnCE
Connection string	<b>ce_schema</b>

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Values

*String*

Default

The recommended file extension is *.usm*.

Description

Used only when you create a database.

The path and filename of the UltraLite schema file on Windows CE. The default extension for UltraLite schema files is *.usm*. This is a required parameter when using CreateDatabase for CE.

Example

◆ The following connection string fragment supplies the *ce\_schema* and *schema\_file* parameters. When run on the desktop, *MyOrders.usm* is used; when run on a CE device (including the emulator), *orders.usm* is used.

```
"CE_SCHEMA=orders.usm;SCHEMA_FILE=MyOrders.usm"
```

## Schema On Desktop connection parameter

Function

To identify the schema file in the desktop development environment.

Syntax

Interface	Connection parameter
UltraLite for MobileVB	SchemaOnDesktop
UltraLite ActiveX	SchemaOnDesktop
UltraLite.NET	SchemaOnDesktop
Native UltraLite for Java	schemaOnDesktop
UltraLite for M-Business Anywhere	schemaOnDesktop
Connection string	<b>schema_file</b>

---

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Values

*String*

Default

The recommended file extension is *.usm*.

Description

Used only when you create a database.

The path and filename of the UltraLite schema in the development environment.

Example

- ◆ The following connection string fragment supplies the *ce\_schema* and *schema\_file* parameters. When run on the desktop, *MyOrders.usm* is used; when run on a CE device (including the emulator), *orders.usm* is used.

```
"CE_SCHEMA=orders.usm;SCHEMA_FILE=MyOrders.usm"
```

## Schema On Palm connection parameter

Function

To identify the schema file deployed to a Palm OS device.

Syntax

Interface	Connection parameter
UltraLite for MobileVB	SchemaOnPalm
UltraLite for M-Business Anywhere	schemaOnPalm
Connection string	<b>palm_schema</b>

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Values

*String*

Default

There is no default value.

Description

This parameter specifies the UltraLite schema for a Palm OS device. It is required only when you create a database.

Although *.pdb* is the extension on the desktop, do not supply *.pdb* in your connection parameter string.

Example

- ◆ The following connection string fragment supplies the *palm\_schema* and *schema\_file* parameters.

```
"PALM_SCHEMA=orders;SCHEMA_FILE=MyOrders.usm"
```



## VFS On Palm parameter

### Function

Identifies the Palm card as using the virtual file system.

This parameter is available only in UltraLite for MobileVB. To use the virtual file system from an embedded SQL or static C++ API application, use the `EnablePalmFileDB` function.

### Syntax

Interface	Connection parameter
UltraLite for MobileVB	VFSOnPalm
Connection string (UltraLite for MobileVB)	PALM_FS=VFS

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

### Values

As a parameter, VFSOnPalm is a boolean value.

In the connection string, the parameter must be specified as follows:

```
PALM_FS=VFS
```

### Description

The `palm_fs=vfs` parameter needs to be specified both for `CreateDatabase` and `OpenConnection` if you are using the VFS card for Palm devices and you want the database stored on the card.

The default database filename is `ul_udb_YYYY.udb`, where `YYYY` is the creator ID of the application. You can control the filename by specifying a different creator ID in the `DatabaseOnPalm` parameter. For example, the following connection string references a database on the card with filename `ul_udb_XXXX.udb`:

```
palm_db=XXXX;palm_fs=vfs
```

Even when the VFSOnPalm parameter is specified, the `palm_db` parameter (Database on Palm) must be set to a valid creator ID.

If the VFS On Palm parameter is not specified, the database is created (or dropped from or to which you are connecting) on the device and not the card.

---

## Additional connection parameters

These are optional parameters to configure a database when it is created. Some of these parameters can influence performance, so it is suggested that you test these parameters to find the optimal performance for your application.

For more information

- ◆ **UltraLite for MobileVB** See “CreateDatabase method” [*UltraLite for MobileVB User’s Guide*, page 105], and “CreateDatabaseWithParms method” [*UltraLite for MobileVB User’s Guide*, page 107].
- ◆ **UltraLite ActiveX** See “CreateDatabase method” [*UltraLite ActiveX User’s Guide*, page 101], and “CreateDatabaseWithParms method” [*UltraLite ActiveX User’s Guide*, page 102].
- ◆ **Native UltraLite for Java** See **ianywhere.native\_ultralite.DatabaseManager** in the Native UltraLite for Java API Reference.
- ◆ **UltraLite.NET** See “OpenWithCreate method” [*UltraLite.NET User’s Guide*, page 93] (iAnywhere.Data.UltraLite namespace) or “CreateDatabase method” [*UltraLite.NET User’s Guide*, page 401] (iAnywhere.UltraLite namespace).
- ◆ **UltraLite C++ Component** See “CreateAndOpenDatabase Function” [*UltraLite C/C++ User’s Guide*, page 250].
- ◆ **UltraLite for M-Business Anywhere** See “Method createDatabase” [*UltraLite for M-Business Anywhere User’s Guide*, page 75], and “Method createDatabaseWithParms” [*UltraLite for M-Business Anywhere User’s Guide*, page 76].
- ◆ **UltraLite for embedded SQL** See “Macros and compiler directives for UltraLite C/C++ applications” [*UltraLite C/C++ User’s Guide*, page 221].
- ◆ **UltraLite static C++** See “Macros and compiler directives for UltraLite C/C++ applications” [*UltraLite C/C++ User’s Guide*, page 221].

### Database Name connection parameter

Function

For applications that connect to more than one database, this parameter can be used to distinguish the databases.

Syntax

Interface	Connection parameter
Connection string	dbn

**Usage** Specify this parameter either in connection strings or in the AdditionalParms string.

Once a database has been started, UltraLite sets the database name. You can then make new connections using the database name parameter instead of specifying the database file.

**Default** The default value is derived from the database file name by removing the path and extension. On Palm OS, the default value is the creator ID.

## Obfuscate connection parameter

**Function** Obfuscates the database. Obfuscation is a form of simple encryption.

**Syntax**

Interface	Connection parameter
Connection string	<b>obfuscate</b>

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

**Values** **0** or **1**. A value of 1 indicates that the database should be obfuscated.

**Usage** Used only when you create a database.

Embedded SQL and static C++ API developers can also use the `UL_ENABLE_OBFUSCATION` macro to obfuscate a database. See [“UL\\_ENABLE\\_OBFUSCATION macro” \[UltraLite C/C++ User’s Guide, page 221\]](#).

**Default** By default, databases are not obfuscated.

**See also** [“Encrypting UltraLite databases” on page 36](#)

## Page Size connection parameter

**Function** Defines the database page size.

**Syntax**

Interface	Connection parameter
Connection string	<b>page_size</b>

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

**Usage** Used only when you create a database.

Used when you create a database. Use k or K to denote kilobytes.

---

Default	The default page size for UltraLite databases is 4 K. The range of size is 2 K to 4 K.
Description	UltraLite databases are stored in pages. I/O operations are carried out a page at a time. It can be used on any target platform. Setting a page size of 2 K reduces the maximum number of tables to approximately 500.  This parameter is ignored when starting an existing database.
Example	You can specify 2 kb pages using the following storage parameters string:  <code>"schema_file=MyOrders.usm;PAGE_SIZE=2K"</code>

## Palm Allow Backup parameter

Function Control backup behavior over HotSync on Palm devices.

Syntax

Interface	Connection parameter
Connection string	<b>palm_allow_backup</b>

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Usage Used when you configure a database.

Values **yes** or **no**.

Description If the backup bit is set on the UltraLite database, and if this parameter is set to **yes**, the entire Palm database is backed up every time the device is synchronized using HotSync. If this parameter is not set, UltraLite ensures that the backup bit is cleared. In most applications, data is backed up by synchronization, so there is no need to set this parameter.

The backup bit is set when a database file is deployed by HotSync, and can also be set by the ULUtil utility. For more information, see [“The ULUtil utility” on page 123](#).

Example The following string sets the parameter.

```
#define UL_STORE_PARMS UL_TEXT("palm_allow_backup=yes")
```

## Reserve Size connection parameter

Function Reserves file system space for storage of UltraLite persistent data.

Syntax

Interface	Connection parameter
Connection string	<code>reserve_size</code>

☞ For information about using the connection parameter, see [“Specifying connection parameters” on page 66](#).

Usage	Use k or K, m or M to denote kilobytes or megabytes, respectively.
Values	Values can be expressed in KB or MB.
Description	<p>The <code>reserve_size</code> parameter allows you to pre-allocate the file system space required for your UltraLite database without actually inserting any data. Reserving file system space can improve performance slightly and also prevent out of memory failures. By default, the persistent storage file only grows when required as the application updates the database.</p> <p><code>Reserve_size</code> reserves file system space, which includes the metadata in the persistent store file, and not just the raw data. The metadata overhead as well as data compression must be considered when deriving the required file system space from the amount of database data. Running the database with test data and observing the persistent store file size is recommended.</p> <p>The <code>reserve_size</code> parameter reserves space by growing the persistent store file to the given reserve size on startup, regardless of whether the file previously existed. The file is never truncated.</p> <p>This parameter does not apply to the Palm Computing Platform unless the application uses the Virtual File System (VFS).</p>
Example	<p>Use the <code>reserve_size</code> parameter to pre-allocate space as follows:</p> <pre>"CE_SCHEMA=orders;RESERVE_SIZE=128K"</pre> <p>This example ensures that the persistent store file is at least 128 K upon startup.</p>



---

## CHAPTER 5

# UltraLite Utilities Reference

About this chapter

This chapter provides reference information about UltraLite utility programs.

Contents

<b>Topic:</b>	<b>page</b>
The UltraLite engine	88
The UltraLite Generator	89
The SQL Preprocessor	95
The HotSync Conduit Installer	99
The dbulstop utility	100
The ulconv utility	101
The ulcreate utility	108
The uldbsgen utility	110
The ulinit utility	112
The UltraLite Interactive SQL utility	115
The ulload utility	117
The ulsync utility	119
The ulunload utility	121
The ULUtil utility	123
The UltraLite Schema Painter	124
The ulxml utility	126

---

## The UltraLite engine

Function	Manages UltraLite databases. In contrast to the UltraLite runtime, which hosts a single application, the UltraLite engine permits multiple applications to access UltraLite databases concurrently.
Syntax	<b>dbuleng9</b>
Applies to	Windows XP and Windows CE.
Remarks	There are no command line options for the UltraLite engine.
See also	<ul style="list-style-type: none"><li>◆ <a href="#">“Using the UltraLite engine” on page 61</a></li><li>◆ <a href="#">“The dbulstop utility” on page 100</a></li></ul>



## The UltraLite Generator

Applies to	Static interfaces only.
Function	The UltraLite generator implements your application database and generates additional C/C++ or Java source files, which must be compiled and linked into your application.
Syntax	<b>ulgen</b> [ <i>options</i> ] [ <i>project</i> [ <i>output-filename</i> ] ]

Option	Description
<b>-a</b>	Uppercase SQL string names [ Java ]
<b>-c</b> “ <i>key-word=value;...</i> ”	Supply database connection parameters for your reference database
<b>-e</b>	Replace SQL strings with generated constants [ Java ]
<b>-f</b> <i>filename</i>	Specify output file name
<b>-g</b>	Do not display warnings
<b>-I</b>	Generate inner classes [ Java ]
<b>-j</b> <i>project-name</i>	Project name
<b>-l</b> <i>type</i>	Log the execution plan for each statement to a file. The type must be one of the following: <ul style="list-style-type: none"> <li>◆ xml</li> <li>◆ short</li> <li>◆ long</li> </ul>
<b>-m</b> <i>version</i>	Specify the version name for generated synchronization scripts
<b>-o</b> <i>table-name,...</i>	Specify the order in which tables are uploaded during synchronization
<b>-p</b> <i>package-name</i>	Package name for generated classes [ Java ]
<b>-q</b>	Do not print the banner
<b>-r</b> <i>filename</i>	The file containing the trusted root certificates
<b>-s</b> <i>filename</i>	Generate a list of SQL strings in an interface definition [ Java ]

Option	Description
<b>-t</b> <i>target</i>	Target language. Must be one of the following: <ul style="list-style-type: none"> <li>◆ c</li> <li>◆ c++</li> <li>◆ java</li> </ul>
<b>-u</b> <i>pub-name</i>	The publication to use (C++ API only)
<b>-v</b> <i>pub-name</i>	The publication to use for synchronization
<b>-x</b>	Generate more and smaller C/C++ files.

## Remarks

The UltraLite generator creates code that you compile and make part of an UltraLite application. Its output is based on the schema of the Adaptive Server Anywhere reference database and the specific SQL statements or tables that you use in your embedded SQL source files.

You must ensure that all your statements and tables are defined in the `dbo.ul_statement` table before running the generator. You do this as follows:

- ◆ In embedded SQL, run the SQL preprocessor on each file.
- ◆ In the C/C++ API and Java, add statements to the database using `ul_add_statement`, and/or define publications in the database.

In this table, statements are associated with projects. By specifying a project name on the generator command line, you determine which statements are included in your generated database.

You can include multiple projects, and also mix projects with a publication, on the generator command line. You must run the generator only once for each generated database.

If you do not specify an output file name, the generated code is written to a file with a name of *project*. It is recommended that you specify an output file name using the `-f` command line option.

**Customizing UltraLite generator operations** The UltraLite analyzer provides hooks that you can use to customize the code generation process. These hooks are stored procedure names. If you supply stored procedures with the following names, the UltraLite analyzer invokes them before and after the analysis process:

- ◆ `sp_hook_ulgen_begin()`

◆ **sp\_hook\_ulgen\_end()**

These hooks are defined in the reference database and are used only during the analyzer analysis phase. The hooks can be created as follows:

```
CREATE PROCEDURE sp_hook_ulgen_begin ()
BEGIN
// actions here
END
CREATE PROCEDURE sp_hook_ulgen_end ()
BEGIN
// actions here
END
```

## Options

**project** The project name determines the set of statements that are to be included in the generated database. For a more precise specification of the filename, use the `-j` option.

**output-filename** The name for the generated file, without extension. For a more precise specification of the filename, use the `-f` option.

In Java, this name is also the database name, which you must supply on connection.

**-a** If you are developing a Java application, the names of the SQL statements in the project are used as constants in your application. By convention, constants are upper case, with underscore characters between words. The `-a` option makes the names of SQL statements fit this convention by making the characters upper case and inserting an underscore whenever an uppercase character in the original name is found if not already preceded by an underscore or an uppercase character. For example, a statement named `MyStatement` becomes `MY_STATEMENT`, and a statement named `AStatement` becomes `ASTATEMENT`.

The generated names have spaces and non-alphanumeric characters replaced with an underscore, regardless of whether `-a` is used.

**-c connection-string** The connection string must give the generator permission to read and modify your reference database. This parameter is required.

**-e** The SQL strings in the generated database are replaced by smaller, generated strings. This option is useful when you are trying to reduce the footprint of a database with a lot of statements.

**-f filename** This is the recommended way to specify the output file. Do not specify an extension.

**-g** Suppress the display of warning messages. Error messages are still displayed.

---

The UltraLite generator provides warnings to indicate that some generated code may, under some circumstances, cause problems. For example, it generates a warning for SQL statements that include temporary tables.

**-l** By default, generated classes are written as top-level non-public classes except for the main database class. If you use `-I`, the generated classes are written as inner classes. If you use this option, you must use a Java compiler that can correctly compile inner classes.

**-j project-name** This is the recommended way to specify the project. You can specify multiple projects using this option as follows:

```
ulgen -j project1 -j project2 ...
```

**-l type** Log the execution plan for queries in the application. These plans can be viewed in Interactive SQL. The types available are:

- ◆ **xml** Description in XML format. Use the Interactive SQL File ► Open command to display the plan.
- ◆ **short** Brief description of the plan in a file named `<statement>.txt`. The content is that generated by the EXPLANATION function
- ◆ **long** Detailed description of the plan in a file named `<statement>.txt`. The content is that generated by the PLAN function.

**-m version** Specify the version name for generated synchronization scripts. The generated synchronization scripts can be used in a MobiLink consolidated database for simple synchronization. The default value is `ul_default`.

**-o table-name,...** Specify the order in which tables are uploaded during synchronization. This option can be used to avoid referential integrity errors during upload. Each table to be uploaded must be specified exactly once.

**-p package-name** A package name for generated files when generating Java output.

**-q** Do not display output messages.

**-r filename** The file containing the trusted root certificates used for secure synchronization using Certicom security software.

The generator embeds these trusted roots into the UltraLite application. When the application receives a certificate chain from a MobiLink synchronization server, it checks if its root is among the trusted roots, and only accepts a connection if it is.

The generator checks the expiry dates of all the certificates in the trusted root certificate file and issues the following warning for any certificate that

expires in less than 6 months (180 days):

```
Warning: Certificate will expire in %1 days"
```

The generator issues a `Certificate has expired` error for any certificate that has already expired.

On platforms other than Palm OS, an alternative to using `-r` is to supply the certificate separately and address it with the `trusted_certificates` security parameter. For more information, see “[trusted\\_certificates](#)” [*MobiLink Clients*, page 53].

☞ For more information, see “[UltraLite Synchronization Parameters](#)” [*MobiLink Clients*, page 315], and “[MobiLink Transport-Layer Security](#)” [*MobiLink Administration Guide*, page 165].

**-s filename** Generate an interface that contains the SQL statements as constants. This option is for use with Java only. The interface file has a format similar to the following example:

```
package com.sybase.test;
public interface EmpTestSQL {
    String EMPLOYEE = "select emp_fname, emp_lname
        from employee where emp_id = ?";
    String UPDATE_EMPLOYEE = "update employee
        set emp_fname = ?, emp_lname = ?
        where emp_id = ?";
}
```

Do not supply the `.java` extension in `filename`. The `-a` option controls the case of the statement names.

**-t target** Specifies the kind and extension of the generated file.

- ◆ If you are using Java, you must use a *target* of **java**. If you are using embedded SQL or the C++ API, you can use a *target* of either **c** or **c++**. Which one you choose decides the extension of the file name, and has nothing to do with whether you are using the C++ API or embedded SQL.
- ◆ If you specify **c++**, the following files are generated:
  - **filename.cpp** The code for the generated API.
  - **filename.h** A header file. You do not need to look at this file.
  - **filename.hpp** The C++ API definition for your application.
- ◆ If you specify a *target* of **c**, `filename.c` is generated.

**-u pub-name** If you are generating a C++ API for a publication, specify the publication name with the `-u` option.

---

**-v pub-name** Specifies a publication to synchronize. If you do not use publications to define which changes are to be synchronized, all changes are synchronized.

If columns or tables specified in publications are not referenced by SQL statements in your application, they are not included in the UltraLite database.

To specify multiple publications, repeat the `-v` option. For example:

```
ulgen -v pub1 -v pub2 ...
```

The maximum number of publications is 32.

☞ For more information, see “[UltraLite Clients](#)” [*MobiLink Clients*, page 277].

**-x** This option is intended for use in situations where the file containing the generated code is too large for the C/C++ compiler to compile.

This option causes the UltraLite generator to produce more and smaller files. When `-x` is used, the UltraLite generator writes out one C/C++ file for the database and one for each SQL statement.

This option has no effect when generating Java code.

## The SQL Preprocessor

Applies to	Embedded SQL static development model only.
Function	The SQL preprocessor processes a C or C++ program containing embedded SQL, before the compiler is run.
Syntax	<b>sqlpp</b> [ <i>options</i> ] <i>sql-filename</i> [ <i>output-filename</i> ]

Option	Description
<b>-c</b> “ <i>key-word=value;...</i> ”	Supply database connection parameters for your reference database
<b>-d</b>	Generate code that favors small data size
<b>-e</b> <i>level</i>	Flag non-conforming SQL syntax as an error
<b>-g</b>	Do not display UltraLite warnings
<b>-h</b> <i>line-width</i>	Limit the maximum line length of output
<b>-k</b>	Include user declaration of SQLCODE
<b>-m</b> <i>version</i>	Specify the version name for generated synchronization scripts
<b>-n</b>	Line numbers
<b>-o</b> <i>operating-sys</i>	Target operating system: WIN32, WINNT, NETWARE, or UNIX
<b>-p</b> <i>project-name</i>	UltraLite project name
<b>-q</b>	Quiet mode—do not print banner
<b>-s</b> <i>string-len</i>	Maximum string length for the compiler
<b>-w</b> <i>level</i>	Flag non-conforming SQL syntax as a warning
<b>-x</b>	Change multi-byte SQL strings to escape sequences.
<b>-z</b> <i>sequence</i>	Specify collation sequence

See also [“Introduction” \[ASA Programming Guide, page 136\]](#)

Remarks The SQL preprocessor processes a C or C++ source file that contains embedded SQL, before the compiler is run. This preprocessor translates the SQL statements in the *input-file* into C/C++. It writes the result to the *output-file*. The normal extension for source files containing embedded SQL

---

is *sql*. The default output filename is the *SQL-filename* base name with an extension of *c*. However, if the *SQL-filename* already has the *.c* extension, the default output extension is *.cc*.

When preprocessing files that are part of an UltraLite application, the SQL preprocessor requires access to an Adaptive Server Anywhere reference database. You must supply the connection parameters for the reference database using the **-c** option.

If you specify *no* project name, the SQL preprocessor also runs the UltraLite generator and appends additional code to the generated C/C++ source file. This code contains a C/C++ language description of your database schema as well as the implementation of the SQL statements in the application.

**Customizing UltraLite generator operations** The UltraLite analyzer provides hooks that you can use to customize the code generation process. These hooks are stored procedure names. If you supply stored procedures with the following names, the UltraLite analyzer invokes them before and after the analysis process:

◆ **sp\_hook\_ulgen\_begin()**

◆ **sp\_hook\_ulgen\_end()**

These hooks are defined in the reference database and are used only during the analyzer analysis phase. The hooks can be created as follows:

```
CREATE PROCEDURE sp_hook_ulgen_begin ()
BEGIN
// actions here
END
CREATE PROCEDURE sp_hook_ulgen_end ()
BEGIN
// actions here
END
```

## Options

**-c** Required when preprocessing files that are part of an UltraLite application. The connection string must give the SQL preprocessor access to read and modify your reference database.

**-d** Generate code that reduces data space size. Data structures are reused and initialized at execution time before use. This increases code size.

**-e** This option flags any embedded SQL that is not part of a specified set of SQL/92 as an error. The option is not applicable to UltraLite.

The allowed values of *level* and their meanings are as follows:

◆ **e** flag syntax that is not entry-level SQL/92 syntax

◆ **i** flag syntax that is not intermediate-level SQL/92 syntax



- ◆ **f** flag syntax that is not full-SQL/92 syntax
- ◆ **t** flag non-standard host variable types
- ◆ **u** flag features not supported by UltraLite
- ◆ **w** allow all supported syntax
- g** Do not display warning specific to UltraLite code generation.
- h num** Limits the maximum length of lines output by *sqlpp* to NUM characters. The continuation character is a backslash (\), and the minimum value of NUM is ten.
- k** Notifies the preprocessor that the program to be compiled includes a user declaration of SQLCODE.
- m version** Specify the version name for generated synchronization scripts. The generated synchronization scripts can be used in a MobiLink consolidated database for simple synchronization. The default value is **ul\_default**.
- n** Generate line number information in the C file. This consists of *#line* directives in the appropriate places in the generated C code. If your compiler supports the *#line* directive, this option will make the compiler report errors on line numbers in the **SQL-filename**, as opposed to reporting errors on line numbers in the C/C++ output file. Also, the *#line* directives will indirectly be used by the source-level debugger so that you can debug while viewing the **SQL-filename**.
- o** Specify the target operating system. Note that this option must match the operating system where you will run the program. A reference to a special symbol will be generated in your program. This symbol is defined in the interface library. If you use the wrong operating system specification or the wrong library, an error will be detected by the linker. The supported operating systems are:
  - ◆ **WIN32** Microsoft Windows 95/98/Me and Windows CE
  - ◆ **WINNT** Microsoft Windows NT/2000/XP
- p project-name** Identifies the UltraLite project to which the embedded SQL files belong. Applies only when processing files that are part of an UltraLite application.
- q** Operate quietly. Do not print the banner.
- s string-len** Set the maximum size string that the preprocessor will put into the C file. Strings longer than this value will be initialized using a list of characters ('a', 'b', 'c', and so on). Most C compilers have a limit on the size

---

of string literal they can handle. This option is used to set that upper limit. The default value is 500.

**-w level** This option flags any embedded SQL that is not part of a specified set of SQL/92 as a warning. The option is not applicable to UltraLite.

The allowed values of *level* and their meanings are as follows:

- ◆ **e** flag syntax that is not entry-level SQL/92 syntax
- ◆ **i** flag syntax that is not intermediate-level SQL/92 syntax
- ◆ **f** flag syntax that is not full-SQL/92 syntax
- ◆ **t** flag non-standard host variable types
- ◆ **u** flag features not supported by UltraLite
- ◆ **w** allow all supported syntax

**-x** Change multi-byte strings to escape sequences so that they can pass through compilers.

**-z sequence** This option specifies the collation sequence or filename. For a listing of recommended collation sequences, type **dbinit -l** at the command prompt.

## The HotSync Conduit Installer

**Function** The utility installs or removes a HotSync conduit onto the current machine.

**Syntax** `dbcond9 [ options ] id`

Option	Description
<i>id</i>	The creator ID of the application to use the conduit
<b>-n name</b>	The name displayed by the HotSync manager.
<b>-k key</b>	Default encryption key
<b>-p parms</b>	Default client synchronization parameters.
<b>-x</b>	Remove the conduit for the specified creator ID

**Remarks** The utility installs a HotSync conduit onto the current machine. HotSync Manager must be present in order for the HotSync Conduit Installer to run.

**Options** **id** The application user ID who is to use the conduit. If a conduit already exists for the specified *creatorID*, it is replaced by the new conduit. This is a required option.

**-k key** The default encryption key. Supply an empty string (`-k ""`) to clear an existing encryption key.

**-n name** The name displayed by the HotSync manager. This is also the name of the subdirectory where the conduit stores data. Do not use this option together with `-x`. The default value is **MobiLink conduit**.

**-p parms** The default client synchronization parameters. Supply an empty string (`-p ""`) to clear any existing parameters.

**-x** Remove the conduit for the named *creatorID*. If `-x` is not specified, a conduit is installed.

**Examples** The following command line installs the conduit for the CustDB sample application, which has a creator ID of Syb2:

```
dbcond9 -n CustDB Syb2
```

---

## The dbulstop utility

Function	Stops the UltraLite engine.
Syntax	<b>dbulstop</b>
Applies to	Windows XP and Windows CE.
Remarks	There are no command line options for the UltraLite engine stop utility.
See also	<ul style="list-style-type: none"><li>◆ <a href="#">“Using the UltraLite engine” on page 61</a></li><li>◆ <a href="#">“The UltraLite engine” on page 88</a></li></ul>

## The ulconv utility

Function

This utility converts UltraLite databases and schema files among formats.

Syntax


**ulconv create** [ *options* ]

**ulconv** { **load** | **unload** } [ *options* ] *xml-file*

**ulconv sync** [ *options* ] [ *sync-parms* ]

The options depend on the syntax used. Command line options for this utility are case sensitive.

- ◆ **create** Create a new, empty UltraLite database from a specified schema file. The following options are supported:

Option	Description
<b>-c</b> “ <i>keyword=value;...</i> ”	UltraLite database connection parameters. Required.
<b>-m</b>	Create a multibyte UltraLite database.
<b>-p</b> <i>creator-id</i>	Create a Palm OS database, using the supplied creator ID. The database file is given a Palm OS PDB name of <i>ul_udb_creator-id</i> .  For more information about creator IDs, see “ <a href="#">Understanding the Palm Creator ID</a> ” on <a href="#">page 191</a> .
<b>-q</b>	Do not print the banner or messages.
<b>-S</b> <i>schema-file</i>	Use the specified schema file to define the database schema.
<b>-u</b>	Create a Unicode UltraLite database. Default behavior.
<b>-v</b>	Print verbose messages.
<b>-y</b>	Overwrite the database file if it exists.

- ◆ **load** Read an XML file and load the data into a new or existing database. The following options are supported:

Option	Description
<i>xml-file</i>	The XML file that holds the data to be loaded.

Option	Description
<b>-c</b> “ <i>keyword=value;...</i> ”	UltraLite database connection parameters. Required.
<b>-d</b>	Create a new database only if an existing database cannot be opened. If an existing database can be opened, ignore the schema definition in the XML file.  If <b>-d</b> is not supplied, a new database is created.
<b>-D</b> <i>directory</i>	Specify a directory for files holding external data.  The <i>xml-file</i> may contain column data for long binary and long character columns in an external file, as follows:  <pre data-bbox="673 670 1076 895"> &lt;uldata&gt;   &lt;table name="table-name"&gt;     &lt;row       longData.File="column.dat "     .../&gt;     ...   &lt;/table&gt;   ... &lt;/uldata&gt; </pre> If <b>-D</b> is supplied, the path of <i>column.dat</i> is interpreted as relative to the supplied directory. If no <b>-D</b> is supplied, the path is relative to the <i>xml-file</i> .
<b>-i</b>	Include inserted rows in the next upload synchronization. By default, rows inserted by this utility are not uploaded during synchronization.
<b>-m</b>	Create a multibyte UltraLite database.
<b>-n</b>	Load schema only. Ignore data.
<b>-p</b> <i>creator-id</i>	Create a Palm OS database, using the supplied creator ID.
<b>-q</b>	Do not print the banner or messages.
<b>-S</b> <i>schema-file</i>	Save the schema to the specified schema file.
<b>-u</b>	Create a Unicode UltraLite database. Default behavior.
<b>-v</b>	Print verbose messages.

Option	Description
<b>-y</b>	Overwrite the database file if it exists.

- ◆ **sync** Open and synchronize an UltraLite database, using the supplied synchronization parameter. The following options are supported:

Option	Description
<i>sync-parms</i>	<p>A semicolon-separated list of <i>keyword=value</i> pairs that control synchronization.</p> <p>The keywords must be taken from the following case-insensitive list:</p> <ul style="list-style-type: none"> <li>downloadOnly (boolean)</li> <li>newpasswd (string)</li> <li>passwd (string)</li> <li>publicationMask (integer)</li> <li>sendColumnNames (boolean)</li> <li>uploadOnly (boolean)</li> <li>username (string)</li> <li>version (string)</li> </ul> <p>For more information about synchronization parameters, see “UltraLite Synchronization Parameters” [<i>MobiLink Clients</i>, page 315].</p>
<b>-a</b> <i>auth-param</i>	<p>MobiLink authentication parameters. Several authentication parameters can be specified using multiple <b>-a</b> <i>auth-param</i> options.</p> <p>The parameters are sent in the order they appear on the command line.</p> <p>☞ For information on MobiLink authentication parameters, see “authenticate_parameters connection event” [<i>MobiLink Administration Guide</i>, page 334].</p>
<b>-c</b> " <i>keyword=value;...</i> "	UltraLite database connection parameters. Required.
<b>-k</b> <i>stream-type</i>	<p>The synchronization stream to use.</p> <p><i>stream-type</i> must be one of the following case-insensitive values:</p> <ul style="list-style-type: none"> <li>tcpip (the default)</li> <li>http</li> </ul>

Option	Description
<b>-q</b>	Do not print the banner or messages.
<b>-s stream-Parms=keyword=value;...</b>	<p>The stream parameters to use when synchronizing.</p> <p>Stream parameters are <i>stream-type</i> specific. The default streamParms is “host=localhost”, which corresponds to the -k tcpip stream-type.</p> <p>For a list of available stream parameters, see “<a href="#">Network protocol options for UltraLite synchronization clients</a>” [<i>MobiLink Clients</i>, page 341].</p> <p>Boolean false can be specified by <b>0</b>, <b>no</b>, <b>off</b> or <b>false</b>; true with the values <b>1</b>, <b>yes</b>, <b>on</b>, or <b>true</b>.</p> <p>Integer values can be given in octal, hex, or decimal.</p> <p>String values can be quoted (quotation marks are removed).</p>
<b>-v</b>	Print verbose messages.

- ◆ **unload** Save an UltraLite database file to an XML document. The following options are supported:

Option	Description
<i>xml-file</i>	The XML file to which data is to be unloaded.
<b>-B max-blob-size</b>	<p>The maximum size of long binary or long character data to write to the XML file. The default value is 10 KB.</p> <p>A value of -1 corresponds to no maximum size (all data is stored in the XML file).</p>
<b>-c “keyword=value;...”</b>	UltraLite database connection parameters. Required.
<b>-d</b>	Unload data only. Do not unload the schema.



Option	Description
<b>-D</b> <i>blob-directory</i>	The directory in which to store long binary data that exceeds the <b>-B</b> <i>max-blob-size</i> limit. Values are stored as files named <i>tableName-columnName-rowNumber.bin</i> . The default directory is the same directory as the XML file.
<b>-e</b> <i>table,...</i>	Exclude data for the named tables.
<b>-n</b>	Unload schema only. Do not unload data.
<b>-q</b>	Quiet mode—do not print messages.
<b>-S</b> <i>schema-file</i>	Save the schema to the named UltraLite schema file.
<b>-t</b> <i>table,...</i>	Only output data for the named tables.
<b>-v</b>	Verbose messages.
<b>-y</b>	Overwrite the XML file if it exists.

## Remarks

The XML file that is a central format for unloading and loading databases is the file `win32\usm.xsd` in your SQL Anywhere installation. This file is the same as that for the UltraLite XML utility.

If your application is built using embedded SQL or the Static C++ API, if you are deploying it to the Palm OS, and if you wish to use a database created using the UltraLite database converter, you must make a special call in your application. Before initializing the database APIs, call `ULEnableGenericSchemaEx ( &sqlca, false )`.

The connection string must include a database file (DBF) parameter.

When used for unload, the utility makes a temporary copy of the UltraLite database file before unloading data.

When used for load or create, you can specify a user ID and password combination in the connection string. The utility adds this user ID and password combination to the list of authenticated users, in addition to the default user ID and password combination of DBA and SQL.

## Examples

The following examples illustrate uses of the UltraLite database converter.

- ◆ Create an UltraLite database for distribution on the Palm OS with an application.
  1. Start MobiLink against the consolidated database:

---

```
start dbmlsrv9 -c dsn=...
```

2. Create an UltraLite schema file:

```
ulinit -f app.usm -c dsn=...
```

3. Create a database in Palm OS format with creator ID **SYB2**:

```
ulconv create -p SYB2 -c DBF=app.udb;schema_file=app.usm
```

This command creates a desktop file named `app.udb` which, when synchronized to a Palm OS device using HotSync, appears as the PDF `ul_udb_SYB2`. If you export the file back to the desktop, it appears as `ul_udb_SYB2`.

4. Synchronize with MobiLink, downloading application data. The following command must be entered all on one line.

```
ulconv sync -c "DBF=app.pdb"
username=appuid;version=app;downloadOnly=true
```

The `username`, `version`, and `downloadOnly` values are the synchronization parameters that control the synchronization.

5. Copy *palmos.pdb* to Palm devices with the UltraLite application.
- ◆ Defragment an existing Windows CE UltraLite database, already copied to the desktop.

1. Unload the schema and data into an XML file *cedatabase.xml*.

```
ulconv unload -c DBF=cedatabase.udb cedatabase.xml
```

2. Load the new database from the XML file (-u forces a new Unicode database)

```
ulconv load -u -c DBF=newdb.udb cedatabase.xml
```

3. Copy the new database file to the Windows CE device.

- ◆ Convert an existing UltraLite database from desktop to Palm OS format.

1. Unload data to *database.xml*:

```
ulconv unload -c DBF=database.udb database.xml
```

2. Create and load a new UltraLite database in Palm OS format, with creator ID **CREA**.

```
ulconv load -p CREA -c dbf=palm.pdb database.xml
```

3. Copy *palm.pdb* to the Palm OS device.

- ◆ Convert from a UNICODE UltraLite database into a multibyte character set database:

```
ulconv unload -c DBF=CEDatabase.udb newdb.xml  
ulconv load -m -c DBF=MBDatabase.udb newdb.xml
```

- ◆ Save the indicated tables as data only:

```
ulconv unload -c dbf=existingDB.udb -d -t table1,table2
```


---

# The ulcreate utility

Function                      Creates UltraLite databases.

Syntax                        **ulcreate** *options*

Command line options for this utility are case sensitive.

Option	Description
<b>-c</b> “ <i>key-word=value;...</i> ”	UltraLite database connection string. Required. Use the DBF parameter to set the database file name. For information about UltraLite connection strings, see <a href="#">“Connection Parameters” on page 63</a> .
<b>-C</b>	Create the database as case-sensitive for all string comparisons.
<b>-I</b> <i>database-id</i>	Set the initial database ID for global autoincrement columns.
<b>-l</b>	List the available collation sequences and exit. For a list of collation sequences, see <a href="#">“Supplied and recommended collations” [ASA Database Administration Guide, page 336]</a> .
<b>-m</b>	Create a multibyte UltraLite database.
<b>-p</b> <i>creator-id</i>	Create a Palm OS database, using the supplied creator ID. The database file is given a Palm OS PDB name of <code>ul_udb_creator-id</code> .  For more information about creator IDs, see <a href="#">“Understanding the Palm Creator ID” on page 191</a> .
<b>-q</b>	Quiet mode—do not print messages.
<b>-S</b> <i>schema-file</i>	Use the specified schema file to define the database schema.
<b>-u</b>	Create a Unicode database. (default)
<b>-v</b>	Print verbose messages.
<b>-y</b>	Overwrite the database file if it exists.
<b>-z</b> <i>collation-sequence</i>	Specify the collation sequence.

**Remarks** It the database file does not exist, the database is created as case insensitive, non-Unicode database with a collation sequence that depends on the current locale.

**Examples** Create an UltraLite database in the file with all default settings:

```
ulcreate -c "dbf=test.udb"
```

Create a case-sensitive Unicode database, overwriting the database file if it exists:

```
ulcreate -c "dbf=test.udb" -C -u -y
```

Create a database and apply the schema supplied in the file *myschema.usm*:

```
ulcreate -c "dbf=test.udb" -S myschema.usm
```

Create an encrypted database with encryption key **afvc\_1835**:

```
ulcreate -c "dbf=test.udb;key=afvc_1835"
```

---

## The uldbsgen utility

**Function** Writes SQL statements defining the schema of an Adaptive Server Anywhere consolidated database to a file, including table definitions and MobiLink synchronization scripts.

This utility is called by the UltraLite Schema Painter when it generates synchronization scripts.

**Syntax** **uldbsgen** *options sql-file*

Command line options for this utility are case sensitive.

Option	Description
<b>-a</b> <i>sync-info-file</i>	Specify a file containing synchronization settings, saved by the UltraLite Schema Painter.
<b>-c</b> “ <i>key-word=value;...</i> ”	UltraLite database connection string. Required. For information about UltraLite connection strings, see <a href="#">“Connection Parameters” on page 63</a> .
<b>-d</b> <i>var=value</i>	Set initial value for a script.
<b>-e</b> <i>table,...</i>	Exclude listed tables.
<b>-gm</b>	Write MobiLink synchronization scripts only.
<b>-gt</b>	Write table definitions only.
<b>-m</b> <i>ml-version</i>	Set the MobiLink synchronization version.
<b>-oa</b>	Cancel if database would be upgraded (default)
<b>-or</b>	Read only. Do not upgrade database.
<b>-ou</b>	Upgrade database if it is from an older UltraLite release.
<b>-p</b> <i>publication,...</i>	Write statements only for tables contained in the listed publications.
<b>-q</b>	Quiet mode—do not print messages.
<b>-s</b> <i>event,...</i>	Write MobiLink synchronization scripts only for the specified synchronization events.
<b>-t</b> <i>table,...</i>	Write statements only for the listed tables.
<b>-v</b>	Verbose messages.
<b>-y</b>	Overwrite <i>sql-file</i> without confirmation.

- Remarks                    The uldbsgen utility is primarily used by the UltraLite Schema Painter. It is provided as a command-line utility for optional batch-mode use.
- See also                    ♦ “The UltraLite Schema Painter” on page 124  
                                 ♦ “Generate Consolidated Database and MobiLink Scripts dialog” [*SQL Anywhere Studio Help*, page 284]

---

## The ulinit utility

Applies to	UltraLite components.
Function	The <i>ulinit</i> utility lets you create a <i>.usm</i> file for use with any UltraLite component. The utility connects to an Adaptive Server Anywhere database. Consequently, SQL Anywhere Studio (version 8.0.2 or later) is required in order to use it.
Syntax	<b>ulinit -f <i>schema_file</i> -n <i>pub_name</i> [ <i>options</i> ]</b>

Option	Description
<b>-c</b> " <i>connection_string</i> "	Supply database connection parameters in the form <i>keyword=value</i> , separated by semi-colons. You supply these so you may connect to an Adaptive Server Anywhere database.
<b>-f</b> <i>schema_file</i>	Specify the name of the output file. This option is required.
<b>-m</b> <i>version</i>	Specify the version string for generated MobiLink scripts.
<b>-n</b> <i>pubname</i>	Add tables to the UltraLite database schema. <i>pubname</i> specifies a publication in the reference database. Tables in the publication are added to the UltraLite database schema. Specify the option multiple times to add multiple publications in to the UltraLite database schema. To add all tables in the reference database to the UltraLite schema, specify <b>-n*</b> . This option is required.
<b>-o</b> " <i>keyword=value;...</i> "	Supply schema creation options.
<b>-palm</b> <i>id</i>	Create a schema file compatible with Palm OS. <i>id</i> is the four digit Palm creator id that identifies the database.
<b>-q</b>	Quiet operation — only report errors and warnings.



Option	Description
<b>-s</b> <i>pubname</i>	Specify a publication for synchronization. <i>pubname</i> specifies a publication in the reference database that is added as a named publication to the UltraLite database. If <b>-s</b> is not supplied, the UltraLite schema has no named publications. This option can be used multiple times.
<b>-t</b> <i>file</i>	Specify the file containing the trusted root certificates.
<b>-w</b>	Do not display warnings.
<b>-z</b> <i>ordering</i>	Specify the order in which tables are uploaded during synchronization (for example, <b>-z table1,table2</b> ).

## Remarks

The **-n** and **-s** options both take publication names in the reference database as arguments, but serve different purposes:

- ◆ The **-n** option defines the tables to be included in the UltraLite database schema. It does not create named publications in the UltraLite database, and is not used for synchronization.
- ◆ The **-s** option defines named publications in the UltraLite database. These named publications are used for synchronization. The **-s** option does not define which tables are included in the UltraLite database schema.

## Examples

Create a file called *customer.usm* that contains the tables in TestPublication:

```
ulinit -c "uid=dba;pwd=sql" -f customer.usm -n TestPublication
```

Create a schema with two distinct publications:

```
ulinit -c "dsn=dsn-name" -f schema.usm -n Pub1 -n Pub2 -s Pub1 -s Pub2
```

For example, one of the publications may contain a small subset of data for priority synchronization, while the other would contain the bulk of the data.

Synchronization of publications is managed with a bit mask in the UltraLite schema. For more information, see [“Designing sets of data to synchronize separately” \[MobiLink Clients, page 280\]](#).

When creating an UltraLite schema for Palm with *ulinit*, use the **-palm** option to generate a *.pdb* file. For example:

---

```
ulinit -c "uid=dba;pwd=sql;dsn=ASA 9.0 Sample"  
-f tutcustomer.usm -n TutCustomersPub -palm Syb3
```

**Note**

*Syb3* is an example of a Palm creator ID. Use the four digit Palm registered creator ID that matches the creator ID of your application. For MobileVB developers, this must be set in your MobileVB project settings.

The PDB file generated by *ulinit* must be loaded to the Palm device. The creator ID used by the application should match the PDF filename. When an UltraLite application creates its database from the schema file, it should include the creator ID in the parameters of the call to `Open`, without the *.pdb* file extension. For example:

```
DatabaseManager.CreateDatabase( "palm_schema=Syb3" )
```

## The UltraLite Interactive SQL utility

Function Execute SQL statements against an UltraLite database.

Syntax **ulysql** [ *options* ] [ *command-file* ]

Option	Description
<b>-c</b> “ <i>key-word=value;...</i> ”	<p>UltraLite database connection string. If you do not supply a connection string, a connection dialog is displayed.</p> <p>The connection string must include the following keywords:</p> <ul style="list-style-type: none"> <li>dbf (database file)</li> <li>uid (user ID)</li> <li>pwd (password)</li> </ul> <p>For more information on connection strings, see <a href="#">“Connection Parameters” on page 63</a>.</p>
<b>-oa</b>	Do not open databases with old file formats.
<b>-or</b>	Open databases with old file formats as read-only.
<b>-ou</b>	Upgrade databases with old file formats to the format of the current version.
<i>command-file</i>	<p>A file containing one or more SQL statements to be executed. When <i>command-file</i> is specified, the user interface is not displayed.</p> <p>The file must follow the following conventions:</p> <ul style="list-style-type: none"> <li>Each statement must end with a semi-colon. No characters other than white space are allowed between the semi-colon and the end of the line.</li> <li>Each statement must begin on a new line. Only white space is allowed to precede a statement.</li> <li>There is no support for comment characters.</li> <li>Statements that return a result set (queries) are ignored.</li> </ul>

Remarks UltraLite Interactive SQL is a graphical utility for executing SQL statements against an UltraLite database. It is useful during application development for developing and testing SQL statements, and for browsing data in databases.

UltraLite Interactive SQL has a window that displays the access plan of a query. If you have queries that demonstrate slow performance, this query

---

plan may be helpful in diagnosing the problem. For example, it may be the case that creating a suitable index would help speed up your query.

If you choose to do so, UltraLite upgrades old database files to the native file format of the current version of the software. This upgrade makes it impossible to use the database from older versions of UltraLite, including applications developed with older versions of the software. Use the `-or` or `-oa` options to ensure that old databases are not upgraded. If you choose the `-or` option, UltraLite opens the database in read-only mode. Any changes you make are discarded when UltraLite Interactive SQL closes, even if you commit your changes.

If you do not specify any of the `-o` parameters and if the database needs to be upgraded, the Upgrade dialog box is displayed. This dialog allows you to choose whether to upgrade, open in read-only mode or cancel the operation.

See also

- ◆ [“Dynamic SQL statements” on page 172](#)
- ◆ [“Query optimization” on page 185](#)

## The ulload utility

Function Loads data from an XML file into an UltraLite database.

Syntax **ulload** *options xml-file*

Command line options for this utility are case sensitive.

Option	Description
<b>-c</b> “ <i>key-word=value;...</i> ”	UltraLite database connection string. Required. Use the DBF parameter to set the database file name. For information about UltraLite connection strings, see <a href="#">“Connection Parameters” on page 63</a> .
<b>-d</b>	Load data only.
<b>-D</b> <i>directory</i>	Specify the directory for data stored in external files. The default is the same directory as the XML file.  This option is primarily for use with XML files created by ulunload.
<b>-i</b>	Include inserted rows in the next upload synchronization. By default, rows inserted by this utility are not uploaded during synchronization.
<b>-m</b>	Create a multibyte database.
<b>-n</b>	Schema only—ignore data.
<b>-p</b> <i>creator-id</i>	Create a Palm OS database using the named <i>creator-id</i> .
<b>-q</b>	Quiet mode—do not print messages.
<b>-S</b> <i>schema-file</i>	Save UltraLite schema to the named schema file.
<b>-u</b>	Create a Unicode database. (default).
<b>-v</b>	Verbose messages.
<b>-y</b>	Replace the database without confirmation.  This option has no effect if <b>-d</b> is used.

Remarks The ulload utility loads data from an XML file into an UltraLite database. It creates the UltraLite database file if it does not already exist. Options allow you to specify whether to load database schema, data, or both, as well as to specify the character set of the UltraLite database.

Examples Create a new UltraLite database in the file *sample.udb* from the schema and

---

data in *sample.xml*:

```
ulload -c dbf=sample.udb sample.xml
```

Load the data from *sample.xml* into the existing database *sample.udb*:


```
ulunload -d -c dbf=sample.udb sample.xml
```

## The ulsync utility

Function Synchronize an UltraLite database.

Syntax **ulsync** *options sync-parms*

Command line options for this utility are case sensitive.

Option	Description
<b>-a</b> <i>authentication-parameter</i>	MobiLink authentication parameter.
<b>-c</b> “ <i>keyword=value;...</i> ”	UltraLite database connection string. Required. Use the DBF parameter to set the database file name. For information about UltraLite connection strings, see “ <a href="#">Connection Parameters</a> ” on page 63.
<b>-k</b> <i>stream-type</i>	Specify the synchronization stream. <i>stream-type</i> must be either <b>tcpip</b> or <b>http</b> . The default stream is <b>tcpip</b> .
<b>-q</b>	Quiet mode—do not print messages.
<b>-s</b> <i>stream-parm=value;...</i>	A semicolon-separated list of synchronization stream parameters. The default value is <b>host=localhost</b> .  For more information, see “ <a href="#">Network protocol options for UltraLite synchronization clients</a> ” [ <i>MobiLink Clients</i> , page 341].
<b>-v</b>	Verbose messages.
<b>-y</b>	Replace the database without confirmation.
<i>sync-parms</i>	A semicolon-separated list of keyword-value pairs. The keywords are case insensitive, and are drawn from the following list: downloadOnly (Boolean) newpasswd (string) passwd (string) publicationMask (integer) sendColumnNames (Boolean) uploadOnly (Boolean) username (string) version (string)

Remarks The ulsync utility synchronizes an UltraLite database with a MobiLink

---

synchronization server. It is a tool for testing synchronization during application development.

Do not confuse the *sync-parms* option (which sets synchronization parameters) with the *-s stream-parms* option (which sets synchronization *stream* parameters).

☞ For more information on synchronization parameters, see “Synchronization parameters” [*MobiLink Clients*, page 316]; for more information on stream parameters, see “Network protocol options for UltraLite synchronization clients” [*MobiLink Clients*, page 341].

#### Example

The following command synchronizes a database file *ul.udb* over HTTP with a MobiLink synchronization server running on a machine **server** and listening on port 8181. The MobiLink user name is **ml-user** and the script version is **script-version**.

The command must be entered on a single line.

```
ulsync -c "dbf=ul.udb"  
-k http  
-s "host=server;port=8181"  
"username=ml-user;version=script-version"
```



## The ulunload utility

**Function** Unloads data from an UltraLite database into an XML file.

**Syntax** `ulunload options xml-file`

Command line options for this utility are case sensitive.

Option	Description
<b>-B</b> <i>max-size</i>	Maximum size of binary or character data to be stored in the XML file. The default is 10K. Use <b>-B -1</b> to have no maximum size, and to store all data in the XML file.  Data that is over the maximum size is saved as a file named <i>tablename-columnname-rownumber.bin</i> in the same directory as the XML file, or in the directory specified by <b>-D</b> .
<b>-c</b> "key-word=value;..."	UltraLite database connection string. Required.  Use the DBF parameter to set the database file name. For information about UltraLite connection strings, see <a href="#">"Connection Parameters"</a> on page 63.
<b>-d</b>	Unload data only. Do not unload the schema definition.
<b>-D</b> <i>directory</i>	Specify the directory to store data larger than the maximum size specified by <b>-B</b> . The default is the same directory as the XML file.
<b>-e</b> <i>table,...</i>	Exclude the listed tables.
<b>-n</b>	Schema only—ignore data.
<b>-q</b>	Quiet mode—do not print messages.
<b>-S</b> <i>schema-file</i>	Save UltraLite schema to the named schema file.
<b>-t</b> <i>table,...</i>	Unload only the listed tables.
<b>-v</b>	Verbose messages.
<b>-y</b>	Overwrite <i>xml-file</i> without confirmation.

**Remarks** The ulunload utility unloads data from an UltraLite database into an XML file. As long as you do not use the **-d** option, the unloaded file can be loaded into a new database using the ulload utility.

**Examples** Unload the UltraLite database *sample.udb* into the XML file *sample.xml*:

```
ulunload -c "dbf=sample.udb" sample.xml
```

---

Unload the data from the UltraLite database *sample.udb* into the XML file *sample.xml*, overwriting the file if it exists:

```
ulunload -c "dbf=sample.udb" -d -y sample.xml
```

## The ULUtil utility

**Function** The UltraLite Palm utility is a Palm Computing Platform application that deletes all of the data stored in an UltraLite application's remote database.

**Remarks** The UltraLite Palm utility is installed as the following file:

```
%ASANY9%\UltraLite\Palm\68k\ULUtil.prc
```

**ULUtil** is useful in deployments where devices are shared between different users. When a different user gets a device, they may want to clear out the previous user's data, to save storage space. Also, the previous user might want to clear out their data because it is confidential. Without **ULUtil**, the only way to clear out an application's data would be to delete and re-install the application.

You can set **ULUtil** to back up the Palm store to the PC on subsequent synchronization. You can use this feature to perform an initial synchronization and then backup the store which can be deployed on other devices so they do not need to perform an initial synchronization. The backup option is automatically turned off by the UltraLite runtime to prevent subsequent backups. If you explicitly want to require the database to be backed up on every synchronization, you must add the `palm_allow_backup` parameter in `UL_STORE_PARMS`.

☞ For more information, see “[UL\\_STORE\\_PARMS macro](#)” [*UltraLite C/C++ User's Guide*, page 222].

Once **ULUtil** is installed on the device, you can delete an UltraLite application's data as follows:

1. Switch to **ULUtil**.
2. Select an application from the list of UltraLite Applications. Only applications built with version 8 or later are displayed.
3. Tap the Delete button.

On devices with expansion cards, ULUtil provides access to both file-based and record-based stores.

---

# The UltraLite Schema Painter

## Function

The UltraLite Schema Painter allows you to create a new UltraLite schema file or edit an existing one. You can also use it to define table definitions and synchronization scripts for an Adaptive Server Anywhere consolidated database.

For a tutorial that walks you through some of the features of the UltraLite Schema Painter, see [“Tutorial: Working with UltraLite Databases” on page 129](#).

## Starting the UltraLite Schema Painter

### ❖ To start the UltraLite Schema Painter

1. From the Start menu, choose Start ► Programs ► SQL Anywhere 9 ► UltraLite ► UltraLite Schema Painter.

Alternatively, open a command prompt and enter the following command:

```
ulview
```

## Creating, saving and exporting schema files

### ❖ To create a new schema file

1. Open the Tools folder and double-click Create UltraLite Schema.
2. In the New Schema dialog, type in a file name.
3. Click OK to create the schema.

### ❖ To save a file

1. Choose File ► Save to save the file.
2. You can select to Save in *.xml* or *.usm* format.

### ❖ To export a Palm schema file

1. Right-click the schema icon and choose Export Schema for Palm from the popup menu.
2. Enter a Palm Creator ID.
3. Click OK.

## Managing schema files

When you first rename a table or column in your schema UltraLite stores the original name of the table or column. For example, if you create a table named `cust`, and later rename it to `customer`, `cust` is saved as the old name. If you then renamed the table a second time, to `customer_info`, the old name remains `cust`.

The scheme is designed so that a schema file can be used to alter the schema of an existing database. For example, assume that version one of your application shipped with a table named `cust`. As part of the changes for version two, you modify your version one schema file by renaming the table to `customer`. This automatically saves `cust` as the old name. If you now apply this schema file to a version one database file, UltraLite looks for a table named `cust`, the old name, and renames it `Customer`. The same applies to columns in a table.

It is therefore important for future compatibility that you clear the old names from a schema file after a schema file is deployed.

 For more information, see [“Upgrading UltraLite database schemas” on page 54](#).

### ❖ To clear all of the old names in the schema file after deployment

1. Open the schema file in the UltraLite Schema Painter.
2. Choose File ► Clear Upgrade Information.

This sets all of the old names for tables and columns to empty values. You can then safely edit your schema file for the next version of your application.

Manual renaming of object names

Sometimes it may be desirable to manually alter the names of tables and columns. For example, you may have versions one and two of your application deployed and wish to create a single UltraLite schema file that can upgrade both versions one and two of this database to version three.

### ❖ To manually change object names

1. Open your schema in the UltraLite Schema Painter.
2. Choose File ► Export Schema for Palm.

You can use this feature to inspect the current old names in your schema. If you use `ulxml`, you can explicitly set the old name of tables and columns in the `<table>` and `<column>` XML elements.

---

# The ulxml utility

Applies to

UltraLite components.

Function

The *ulxml* utility lets you convert UltraLite database schema definitions among XML, USM and Palm PDB file formats. For example, you can create a *.usm* file from an XML file. The resulting schema file can be used with any UltraLite component.

Syntax

**ulxml** [ *options* ] *input-file output-file*

Option	Description
<b>-y</b>	Overwrite output file if it already exists.
<b>-totype</b> where <i>type=xml usm pdb</i>	Converts the file to one of these standard formats.
Note: <i>pdb</i> files require a CreatorID.	Use <i>toxml</i> to convert an UltraLite schema to XML. Use <i>tousm</i> to convert an XML file to an UltraLite schema Use <i>topdb creator-id</i> to convert an XML file to an UltraLite schema for Palm.

You can export your UltraLite schema so that you can work in XML format:

---

```
<?xml version="1.0" ?>
- <ul:ulschema xmlns:ul="urn:ultralite">
- <tables>
- <table name="ULCustomer">
- <columns>
  <column name="cust_id" type="integer" null="yes"
    default="global_autoincrement" />
  <column name="cust_name" type="varchar(30)" />
</columns>
- <primarykey>
  <primarycolumn name="cust_id" direction="asc" />
</primarykey>
- <indexes>
- <index name="ULCustomerName" unique="yes">
  <indexcolumn name="cust_name" direction="asc" />
</index>
</indexes>
</table>
+ <table name="ULProduct">
- <table name="ULEmployee">
- <columns>
  <column name="emp_id" type="integer" null="no" />
```

The XML schema that defines the documents is in *win32\usm.xsd* in your

SQL Anywhere installation. The ulxml utility requires that schema file.

You can view and use the documented sample located in  
*Samples\NativeUltraLiteForJava\sample.xml*,  
*Samples\UltraLiteActiveX\sample.xml*, and  
*Samples\UltraLiteForMobileVB\sample.xml*.

**UltraLite Schema Painter**

The UltraLite Schema Painter by default creates, opens and saves UltraLite schema files in their native USM file format. However, you are given the option to create, open and save XML files as well by choosing UltraLite XML Schema Files in any file type dropdown box.

Return code

0 on success, less than 0 on failure.





---

## CHAPTER 6

# Tutorial: Working with UltraLite Databases

### About this chapter

This chapter walks you through several tasks related to UltraLite database schemas and database files. It introduces the UltraLite Schema Painter, the UltraLite Interactive SQL utility, and the UltraLite command-line administration tools. It also shows how you can generate and synchronize with an Adaptive Server Anywhere consolidated database from an UltraLite database.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Lesson 1: Create an UltraLite database schema</a>	130
<a href="#">Lesson 2: Define and create a consolidated database</a>	133
<a href="#">Lesson 3: Enter data in your UltraLite database</a>	137
<a href="#">Lesson 4: Synchronize your databases</a>	138

---

# Lesson 1: Create an UltraLite database schema

In this lesson, you build a single-table UltraLite database schema for Windows CE, Windows XP, and Palm OS devices.

This lesson is the first in a complete tutorial on working with UltraLite databases. You may also have reached this lesson from one of several UltraLite component tutorials, in which case you should return to your main tutorial after creating a schema file.

☞ For more information on UltraLite schemas, see [“Creating UltraLite databases and schemas” on page 28](#).

To start this tutorial, create a directory to hold the schema and other files. This directory is assumed to be `C:\tutorial\`. If you create your tutorial directory elsewhere, supply the path to your location instead of `c:\tutorial\` throughout.

## ❖ To create a schema file

1. Start the UltraLite Schema Painter:

Choose Start ► Programs ► SQL Anywhere 9 ► UltraLite ► UltraLite Schema Painter.

2. Create a new schema file called *tutCustomer*.

- ◆ From the File menus, select New ► UltraLite Schema.
- ◆ For a filename, enter `c:\tutorial\tutCustomer.usm` or Browse to the folder and enter `tutCustomer.usm`.
- ◆ Leave the other settings at their default values and click OK to create the schema.

3. Create a table called customer.

- ◆ In the left pane of the UltraLite Schema Painter, expand the *tutCustomer* item and select the Tables folder.
- ◆ In the right pane, double-click Add Table.  
The New Table dialog appears.
- ◆ Enter the name Customer.
- ◆ Click Add to add the following columns:

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

**Tutorial use only**

The use of autoincrement as a primary key is not recommended in a database that is going to be synchronized. Instead, use a global autoincrement or UUID value. For the purposes of this tutorial, an autoincrement is sufficient.

For more information on maintaining unique primary keys in a synchronization setup, see [“Maintaining unique primary keys”](#) [*MobiLink Administration Guide*, page 56].

- ◆ Set ID as the primary key: click Primary Key and add ID to the index, marking it as ascending.
- ◆ Check your work and click OK to complete the table definition and dismiss the New Table dialog.

4. Click File ► Save to save the *tutcustomer.usm* file.

5. Optionally, export a Palm schema file.

If you intend to work with the Palm OS as a target platform, you may want to export a schema definition for the Palm OS.

- ◆ From the File menu, choose Export Schema for Palm.
- ◆ Enter a Palm Creator ID. For tutorial purposes you could use **Syb3**, but do not use this for deployed applications.

**A note on Palm Creator IDs**

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 production applications, you should obtain and use your own creator ID.

- ◆ Leave the filename at its default setting to save the PDB file in your tutorial directory. Click OK.

You have now defined the schema of an UltraLite database. Although this database contains only a single table, you can use many tables in UltraLite databases.

---

**UltraLite component tutorial readers**

You may be carrying out this lesson as part of an UltraLite component tutorial or you may be working through the complete tutorial in this chapter. If you reached this lesson from an UltraLite component tutorial, you only need the schema file and so you can return to your main tutorial now. Otherwise, continue.

**❖ To create an UltraLite database file**

1. From the Tools menu of the UltraLite Schema Painter, choose Create UltraLite Database.  
The Create UltraLite Database dialog appears.
2. Choose Multibyte Character Database (Desktop) and leave the other settings at their default values.
3. Click OK to create the database.

## Lesson 2: Define and create a consolidated database

You can use the UltraLite Schema Painter to generate a SQL command file that defines tables and synchronization scripts for an Adaptive Server Anywhere consolidated database. This feature is useful if you are extending an UltraLite application to include synchronization.

In this lesson, you create a consolidated database that manages timestamp-based synchronization with your UltraLite database. The task involves the following procedures:

1. Create a publication that holds the Customer table.
2. Define synchronization settings for the table.
3. Generate the table definitions and synchronization scripts.
4. Create the consolidated Adaptive Server Anywhere database.

This lesson assumes that you have the UltraLite Schema Painter open with the tutCustomer schema opened, as at the end of [“Lesson 1: Create an UltraLite database schema”](#) on page 130.

### ❖ To create a publication holding the Customer table

1. Create the publication:  
In the left pane, select the Publications folder. In the right pane, double click Add Publication. The Publication dialog appears.
2. In the Publication Name field, enter **CustomerPublication**.
3. Select the Customer table and click >> to add it to the list of tables in the publication. Click OK.

The next step defines synchronization settings.

### ❖ To define synchronization settings

1. Add the table to the list of tables with custom settings:
  - ◆ In the left pane, open the MobiLink Synchronization folder. In the right pane, double click Add Table-specific Settings. The MobiLink Table Settings dialog appears.
  - ◆ Select the Customer table and click >> to add it to the list of tables with MobiLink settings. Click OK.

---

2. Define the settings.

In the left pane, select the MobiLink Synchronization folder.

Right click the Customer table and choose Properties. The MobiLink Synchronization Property sheet appears. The settings here are chosen for simplicity in this demonstration: in a production environment the settings depend on your business rules.

- a. On the Direction tab, leave the setting at Full Synchronization.
- b. On the Row Increment tab, choose Timestamp. Leave the Use Shadow Table option.

An additional column will be created in the consolidated database that holds the timestamp values for this table.

- c. On the Row Partition tab, leave the setting at Same Data on All Remote Databases.
- d. On the Deletions tab, choose Download Deletions and select the first of the two options. A table named Customer\_deletes will be created in the consolidated database that holds identifying values for the deleted rows.
- e. On the Conflict tab and the Resolution tab, leave the values at their default settings.
- f. Click OK to save the settings.

3. Optionally, preview the table definitions and synchronization scripts.

You can preview the table definitions and synchronization scripts by right-clicking the table and choosing Preview Consolidated Tables and Scripts from the popup menu. The previewed definitions and scripts define the tables and synchronization scripts needed for an Adaptive Server Anywhere consolidated database that can synchronize with this UltraLite database. Comment lines are prefixed by –.

The next step is to generate the SQL command file that holds the table definitions and synchronization scripts.

❖ **To generate the consolidated database table and script definitions**

1. From the Tools menu, choose Generate Consolidated Tables and Scripts. The Generate Consolidated Tables and MobiLink Scripts dialog appears.
2. In the Settings group, set the MobiLink script version is to Tutorial. Leave the other two checkboxes at their default values.

3. In the Generated SQL group, ensure the Consolidated Tables, Triggers, MobiLink Scripts and Procedures checkbox is selected.

During the development of real applications, you may regenerate synchronization scripts and table definitions several times as you modify your application. For this reason, the dialog provides you with the option to generate only some of the database objects.

4. Leave the Generated SQL file as the default setting (*tutCustomer.sql*) and click OK to generate the scripts.

The final step is to use the generated SQL command file to create an Adaptive Server Anywhere consolidated database.

#### ❖ To create a consolidated database

1. Create the consolidated database file.

Open a command prompt, and change to the tutorial directory. Enter the following command to create a database file named *consol.db*:

```
dbinit consol.db
```

2. Define an ODBC data source for the database.
  - a. Open the ODBC Administrator.  
From the Start menu, choose Programs ► SQL Anywhere 9 ► Adaptive Server Anywhere ► ODBC Administrator.
  - b. On the User DSN table, click Add. The Create New Data Source dialog appears.
  - c. From the list, choose Adaptive Server Anywhere 9.0 and click Finish. The Adaptive Server Anywhere ODBC Configuration dialog appears.
  - d. Enter the following settings in the dialog:

Field	Value
Data Source Name (ODBC tab)	Consolidated
User ID (Login tab)	DBA
Password (Login tab)	SQL
Server name (Database tab)	consol
Database file (Database tab)	c:\tutorial\consol.db

- e. On the ODBC tab, click Test Connection to test the settings.
- f. Once the Test Connection succeeds, click OK to save your definition and close the ODBC Administrator. If it fails, review the settings.

- 
3. Connect to the consolidated database using Interactive SQL.
    - a. From the Start menu, choose Programs ► SQL Anywhere 9 ► Adaptive Server Anywhere ► Interactive SQL.
    - b. In the Connection dialog, specify an ODBC data source of Consolidated, and click OK to connect.

4. Open the generated SQL command file.

From the File Menu, choose Open. Open the file *c:\tutorial\tutCustomer.sql*.

5. Run the SQL command file.

Choose SQL ► Execute to execute the SQL statements and create the tables and synchronization scripts in the consolidated database.

Your consolidated database is now created. It does not, of course, contain any data.



## Lesson 3: Enter data in your UltraLite database

In this tutorial we use UltraLite Interactive SQL to add data to the UltraLite database. This lesson introduces the UltraLite Interactive SQL utility.

### ❖ To enter data in your UltraLite database

1. Start UltraLite Interactive SQL

At a command prompt, enter the following command:

```
ulysql
```

A Connect dialog appears.

2. In the Filename field, browse to the *tutCustomer.udb* file that you created in lesson 1.

Enter a user ID of DBA and a password of SQL. Click OK to connect to the database.

3. Enter the following statement to add a Customer name to the database:

```
INSERT Customer (FName, LName, City )  
VALUES ('Jane', 'Doe', 'Boston')
```

Press F5 to execute the statement and add the row to the table.

4. Optionally, add some other names of your choice by modifying this statement. The number of names is not important for this tutorial.
5. Enter the following statement to commit the changes.

```
COMMIT
```

6. Check the values in the table by executing this statement:

```
SELECT * FROM Customer
```

7. Close UltraLite Interactive SQL.

---

## Lesson 4: Synchronize your databases

In this lesson, you upload the changes you have made to your UltraLite database. As you do not have an UltraLite application, this lesson uses the `ulsync` command-line utility to synchronize.

### ❖ To synchronize your databases

1. Start the MobiLink synchronization server running against the consolidated database.

At a command prompt, change directories to your tutorial directory (holding the `consol.db` consolidated database file) and enter the following command:

```
dbmlsrv9 -c "dsn=Consolidated" -zu+
```

The `-zu+` command-line option is a convenience option that allows unrecognized users to synchronize. It should not be used in production environments.

2. Synchronize the UltraLite database.

Ensure that UltraLite Interactive SQL is closed, as only one application at a time can access this UltraLite database.

At the command prompt, enter the following command, which must be entered on a single line:

```
ulsync -c "dbf=tutCustomer.udb"  
"version=Tutorial;username=test"
```

The database synchronizes.

The tutorial is now complete. You can inspect the data in either of the databases, make more changes to either database using Interactive SQL (for the consolidated database) or UltraLite Interactive SQL (for the UltraLite database), and synchronize them using the techniques described here.

## PART II

# ULTRALITE SQL

This part describes the range of SQL available to UltraLite applications.

UltraLite components can construct queries and other SQL statements at runtime (dynamic SQL).

The static interfaces support a wider range of SQL, but the statements used in the application must be specified at compile time.



---

## CHAPTER 7

# SQL Language Elements

About this chapter

This chapter describes the building blocks of SQL statements and data management in UltraLite databases. These building blocks are common to all UltraLite databases.

Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Overview of SQL support in UltraLite</a>	142
<a href="#">Data types in UltraLite</a>	145
<a href="#">UltraLite SQL functions</a>	148

---

## Overview of SQL support in UltraLite

In UltraLite, both the data types available to represent data and the SQL features available to access that data depend on the development model you adopt.

If you use a static interface (embedded SQL, static C++ API, or static Java API), the range of SQL available is wider, but all statements used by the application must be specified at compile time. If you develop your application using an UltraLite component, dynamic SQL provides a narrower range of SQL, but the SQL statements can be constructed at runtime.

When an UltraLite program attempts to use a SQL statement or feature that is not supported in UltraLite, the SQL error `Feature not available with UltraLite (SQLCODE -749)` is reported. Dynamic SQL may also return syntax errors.

- ◆ **Data types** UltraLite supports a subset of the data types available in Adaptive Server Anywhere.

If you create a database from an Adaptive Server Anywhere reference database, you can use a wide range of data types. Those Adaptive Server Anywhere data types not supported in UltraLite are converted by the UltraLite generator into a smaller set of base types. If you create an UltraLite database using the Schema Painter, you are restricted to the smaller set of base types.

For a listing of the UltraLite base types, see [“Data types in UltraLite” on page 145](#).

☞ For a complete listing of Adaptive Server Anywhere data types, see [“SQL Data Types” \[ASA SQL Reference, page 53\]](#).

- ◆ **Identifiers** Identifiers are the names of database objects, such as columns and tables. UltraLite supports the same rules for identifiers as Adaptive Server Anywhere.

Tables in UltraLite do not have an owner. As a convenience for existing SQL and for SQL that is programmatically generated, UltraLite does support the syntax *owner.table-name* in SQL statements but the owner part is not checked.

For information about identifiers, see [“Identifiers” \[ASA SQL Reference, page 7\]](#).

- ◆ **Strings** Strings are used to hold character data in the database. UltraLite supports the same rules for strings as Adaptive Server Anywhere.

If you create an UltraLite database from an Adaptive Server Anywhere reference database, the rules for strings are determined by the database options in effect in the reference database when the UltraLite generator is run. The `QUOTED_IDENTIFIER` option is particularly important in setting rules for strings. Dynamic SQL always operates as if this option is ON (the default in Adaptive Server Anywhere).

☞ For information about strings, see “Strings” [*ASA SQL Reference*, page 9].

The results of comparisons on strings, and the sort order of strings, depends on both the case sensitivity of the database and the character set. These properties are set when the database is created.

For more information, see “Creating UltraLite databases” on page 30.

- ◆ **Functions** UltraLite supports the same range of functions as Adaptive Server Anywhere, with a few minor exceptions. The functions supported are the same for static interfaces such as embedded SQL as they are for dynamic SQL.

For a list of supported functions, see “UltraLite SQL functions” on page 148.

- ◆ **Expressions** Expressions are formed by combining data, often in the form of column references, with operators or functions.

Adaptive Server Anywhere provides a wide range of operators that it uses to form expressions. These operators are available if you develop your UltraLite application using a static interface (embedded SQL, static C++ API, or static Java API). One exception is that in Adaptive Server Anywhere you can use SQL variables to form expressions. You cannot use SQL variables (including global variables) in UltraLite applications. The @@identity global variable is an exception, and can be used within UltraLite applications.

☞ For information about expressions in Adaptive Server Anywhere, see “Expressions” [*ASA SQL Reference*, page 16].

Dynamic SQL is more limited in the range of expressions it supports than is static SQL. For example, dynamic SQL does not support subqueries in all the places that they are supported by static SQL.

☞ For information about the expressions available in dynamic SQL, see “Dynamic SQL expressions” on page 163.

- ◆ **Search conditions** Search conditions or predicates are used in the WHERE clause, the HAVING clause, and the ON clause of SELECT statements.

---

Dynamic SQL is more limited in the range of conditions that it supports than is static SQL.

☞ For information about search conditions available in dynamic SQL, see [“Dynamic SQL search conditions” on page 170](#).

Static interfaces have the entire range of conditions supported in Adaptive Server Anywhere available.

☞ For information about search conditions in Adaptive Server Anywhere, see [“Search conditions” \[ASA SQL Reference, page 23\]](#).

- ◆ **Statements** SQL statements are constructed from the building blocks listed above.

For a list of SQL statements available in dynamic SQL, see [“Dynamic SQL statements” on page 172](#).

The following SQL statements can be used in static UltraLite applications:

- **Data Manipulation Language** SELECT, INSERT, UPDATE, and DELETE statements can be included. You can use placeholders in these statements that are filled in at runtime.

For more information, see [“Writing UltraLite SQL statements” on page 205](#).

- **TRUNCATE TABLE statement** You can use this statement to rapidly delete entire tables.
- **Transaction control** You can use COMMIT and ROLLBACK statements to provide transaction control within your UltraLite application.
- **START/STOP SYNCHRONIZATION DELETE statements** These statements are used to temporarily suspend synchronization of delete operations.

For more information, see [“Temporarily stopping synchronization of deletes” \[MobiLink Clients, page 87\]](#).

☞ For information on other UltraLite limitations, see [“UltraLite database limitations” on page 50](#).



## Data types in UltraLite

The following are the SQL data types supported in UltraLite databases.

If you create an UltraLite database from an Adaptive Server Anywhere reference database, you can use other data types, including user-defined data types, in the reference database. The UltraLite generator casts those data types into a data type supported in UltraLite databases. You cannot use user-defined data types that include DEFAULT values or CHECK constraints.

If you use dynamic SQL, or if you design an UltraLite database using the Schema Painter, you are limited to the use of the types listed here.

☞ For data types in Adaptive Server Anywhere, see “SQL Data Types” [ASA SQL Reference, page 53].

Data type	Remarks
<b>BIT</b>	Boolean values (0 or 1). See “BIT data type” [ASA SQL Reference, page 66]
{ <b>CHAR</b>   <b>CHARACTER</b> } [ ( <i>max-length</i> ) ]	Character data of maximum length <i>max-length</i> characters. The maximum length is 2048 bytes. See “CHAR data type [Character]” [ASA SQL Reference, page 55]
{ <b>VARCHAR</b>   <b>CHARACTER</b> <b>VARYING</b> } [ ( <i>max-length</i> ) ]	In UltraLite, VARCHAR is implemented identically to CHAR. In other databases, VARCHAR is used for variable-length character data of maximum length <i>max-length</i> . See “CHARACTER VARYING (VARCHAR) data type [Character]” [ASA SQL Reference, page 55]
<b>LONG VARCHAR</b>	Arbitrary length character data. Conditions in SQL statements (such as in the WHERE clause) cannot operate on LONG VARCHAR columns. The only operations allowed on LONG VARCHAR columns are to insert, update, or delete them, or to include them in the <i>select-list</i> of a query.  For static interfaces, the maximum size of LONG VARCHAR values is 64 KB. There is no explicit limit for the component interfaces. See “LONG VARCHAR data type [Character]” [ASA SQL Reference, page 56]

Data type	Remarks
[UNSigned] BIGINT	An integer requiring 8 bytes of storage. See “BIG-INT data type [Numeric]” [ASA SQL Reference, page 58]
{DECIMAL   DEC} [( precision [,scale] )]	A decimal number with <i>precision</i> total digits and with <i>scale</i> of the digits after the decimal point. See “DECIMAL data type [Numeric]” [ASA SQL Reference, page 59]
NUMERIC [( precision [,scale] )]	Same as DECIMAL. See “NUMERIC data type [Numeric]” [ASA SQL Reference, page 62]
DOUBLE [PRECISION]	A double-precision floating-point number. See “DOUBLE data type [Numeric]” [ASA SQL Reference, page 60]
FLOAT [( precision )]	A floating point number, which may be single or double precision. See “FLOAT data type [Numeric]” [ASA SQL Reference, page 60]
[UNSigned] {INT   INTEGER}	An integer requiring 4 bytes of storage. See “INT or INTEGER data type [Numeric]” [ASA SQL Reference, page 61]
REAL	A single-precision floating-point number stored in 4 bytes. See “REAL data type [Numeric]” [ASA SQL Reference, page 63]
[UNSigned] SMALLINT	An integer requiring 2 bytes of storage. See “SMALLINT data type [Numeric]” [ASA SQL Reference, page 63]
[UNSigned] TINYINT	An integer requiring 1 byte of storage. See “TINYINT data type [Numeric]” [ASA SQL Reference, page 63]
DATE	A calendar date, such as a year, month and day. See “DATE data type [Date and Time]” [ASA SQL Reference, page 71]
TIME	The time of day, containing hour, minute, second and fraction of a second. See “TIME data type [Date and Time]” [ASA SQL Reference, page 72]

Data type	Remarks
<b>DATETIME</b>	Identical to <b>TIMESTAMP</b> . See “ <a href="#">DATETIME data type [Date and Time]</a> ” [ <i>ASA SQL Reference</i> , page 72]
<b>TIMESTAMP</b>	The point in time, containing year, month, day, hour, minute, second and fraction of a second. See “ <a href="#">TIMESTAMP data type [Date and Time]</a> ” [ <i>ASA SQL Reference</i> , page 73]
<b>VARBINARY</b> [( <i>max-length</i> )]	Identical to <b>BINARY</b> . See “ <a href="#">VARBINARY data type [BINARY]</a> ” [ <i>ASA SQL Reference</i> , page 76]
<b>BINARY</b> [( <i>max-length</i> )]	Binary data of maximum length <i>max-length</i> bytes. The maximum length is 2048 bytes. See “ <a href="#">BINARY data type [Binary]</a> ” [ <i>ASA SQL Reference</i> , page 74]
<b>LONG BINARY</b>	Arbitrary length binary data. Conditions in SQL statements (such as in the <b>WHERE</b> clause) cannot operate on <b>LONG BINARY</b> columns. The only operations allowed on <b>LONG BINARY</b> columns are to insert, update, or delete them, or to include them in the <i>select-list</i> of a query.  For static interfaces, the maximum size of <b>LONG BINARY</b> values is 64 KB. There is no explicit limit for the component interfaces. See “ <a href="#">LONG BINARY data type [BINARY]</a> ” [ <i>ASA SQL Reference</i> , page 74]
<b>UNIQUEIDENTIFIER</b>	Typically used for a primary key or other unique column to hold <b>UUID</b> (Universally Unique Identifier) values that uniquely identify rows. UltraLite provides functions that generate <b>UUID</b> values in such a way that a value produced on one computer will not match a <b>UUID</b> produced on another computer. <b>UNIQUEIDENTIFIER</b> values generated in this way can therefore be used as keys in a synchronization environment.  See “ <a href="#">UNIQUEIDENTIFIER data type [Binary]</a> ” [ <i>ASA SQL Reference</i> , page 75].

# UltraLite SQL functions

The following is a convenient reference for finding functions in dynamic SQL. Each function is listed, and the function type (numeric, character, and so on) is indicated next to it.

☞ For information about functions in Adaptive Server Anywhere, see “SQL Functions” [*ASA SQL Reference*, page 91].

Function	Remarks
<b>ABS</b> ( <i>numeric-expression</i> )	See “ABS function [Numeric]” [ <i>ASA SQL Reference</i> , page 106]
<b>ACOS</b> ( <i>numeric-expression</i> )	See “ACOS function [Numeric]” [ <i>ASA SQL Reference</i> , page 106]
<b>ARGN</b> ( <i>integer-expression</i> , <i>expression</i> [ , ... ] )	See “ARGN function [Miscellaneous]” [ <i>ASA SQL Reference</i> , page 107]
<b>ASCII</b> ( <i>string-expression</i> )	See “ASCII function [String]” [ <i>ASA SQL Reference</i> , page 107]
<b>ASIN</b> ( <i>numeric-expression</i> )	See “ASIN function [Numeric]” [ <i>ASA SQL Reference</i> , page 108]
<b>ATAN</b> ( <i>numeric-expression</i> )	See “ATAN function [Numeric]” [ <i>ASA SQL Reference</i> , page 108]
{ <b>ATN2</b>   <b>ATAN2</b> } ( <i>numeric-expression1</i> , <i>numeric-expression2</i> )	See “ATN2 function [Numeric]” [ <i>ASA SQL Reference</i> , page 109]
<b>AVG</b> ( <i>numeric-expression</i>   <b>DISTINCT</b> <i>column-name</i> )	<b>DISTINCT</b> <i>column-name</i> cannot be used from dynamic SQL. See “AVG function [Aggregate]” [ <i>ASA SQL Reference</i> , page 109]
<b>BYTE_LENGTH</b> ( <i>string-expression</i> )	See “BYTE_LENGTH function [String]” [ <i>ASA SQL Reference</i> , page 111]
<b>BYTE_SUBSTR</b> ( <i>string-expression</i> , <i>start</i> [ , <i>length</i> ] )	See “BYTE_SUBSTR function [String]” [ <i>ASA SQL Reference</i> , page 111]

Function	Remarks
<b>CAST</b> ( <i>expression AS data type</i> )	See “CAST function [Data type conversion]” [ASA SQL Reference, page 112]
<b>CEILING</b> ( <i>numeric-expression</i> )	See “CEILING function [Numeric]” [ASA SQL Reference, page 113]
<b>CHAR</b> ( <i>integer-expression</i> )	See “CHAR function [String]” [ASA SQL Reference, page 113]
<b>CHARINDEX</b> ( <i>string-expression1,</i> <i>string-expression2</i> )	See “CHARINDEX function [String]” [ASA SQL Reference, page 113]
<b>CHAR_LENGTH</b> ( <i>string-expression</i> )	See “CHAR_LENGTH function [String]” [ASA SQL Reference, page 114]
<b>COALESCE</b> ( <i>expression,</i> <i>expression [ , ...]</i> )	See “COALESCE function [Miscellaneous]” [ASA SQL Reference, page 115]
<b>CONVERT</b> ( <i>data-type,</i> <i>expression</i> [ <i>, format-style</i> ] )	See “CONVERT function [Data type conversion]” [ASA SQL Reference, page 121]
<b>COS</b> ( <i>numeric-expression</i> )	See “COS function [Numeric]” [ASA SQL Reference, page 124]
<b>COT</b> ( <i>numeric-expression</i> )	See “COT function [Numeric]” [ASA SQL Reference, page 124]
<b>COUNT</b> ( *   <i>expression</i>   <b>DISTINCT</b> { <i>expression</i> <i>column-name</i> } )	<b>DISTINCT</b> <i>column-name</i> cannot be used from dynamic SQL. See “COUNT function [Aggregate]” [ASA SQL Reference, page 125]
<b>DATALength</b> ( <i>expression</i> )	See “DATALength function [System]” [ASA SQL Reference, page 130]
<b>DATE</b> ( <i>expression</i> )	See “DATE function [Date and time]” [ASA SQL Reference, page 130]

<b>Function</b>	<b>Remarks</b>
<b>DATEADD</b> ( <i>date-part</i> , <i>numeric-expression</i> , <i>date-expression</i> )	See “DATEADD function [Date and time]” [ASA SQL Reference, page 131]
<b>DATEDIFF</b> ( <i>date-part</i> , <i>date-expression1</i> , <i>date-expression2</i> )	See “DATEDIFF function [Date and time]” [ASA SQL Reference, page 131]
<b>DATEFORMAT</b> ( <i>datetime-expression</i> , <i>string-expression</i> )	See “DATEFORMAT function [Date and time]” [ASA SQL Reference, page 133]
<b>DATENAME</b> ( <i>date-part</i> , <i>date-expression</i> )	See “DATENAME function [Date and time]” [ASA SQL Reference, page 133]
<b>DATEPART</b> ( <i>date-part</i> , <i>date-expression</i> )	See “DATEPART function [Date and time]” [ASA SQL Reference, page 134]
<b>DATETIME</b> ( <i>expression</i> )	See “DATETIME function [Date and time]” [ASA SQL Reference, page 134]
<b>DAY</b> ( <i>date-expression</i> )	See “DAY function [Date and time]” [ASA SQL Reference, page 135]
<b>DAYNAME</b> ( <i>date-expression</i> )	See “DAYNAME function [Date and time]” [ASA SQL Reference, page 135]
<b>DAYS</b> ( [ <i>datetime-expression</i> ,] <i>datetime-expression</i> )	See “DAYS function [Date and time]” [ASA SQL Reference, page 136]
<b>DAYS</b> ( <i>datetime-expression</i> , <i>integer-expression</i> )	See “DAYS function [Date and time]” [ASA SQL Reference, page 136]
<b>DEGREES</b> ( <i>numeric-expression</i> )	See “DEGREES function [Numeric]” [ASA SQL Reference, page 141]

Function	Remarks
<b>DIFFERENCE</b> ( <i>string-expression-1</i> , <i>string-expression-2</i> )	See “DIFFERENCE function [String]” [ASA SQL Reference, page 143]
<b>DOW</b> ( <i>date-expression</i> )	See “DOW function [Date and time]” [ASA SQL Reference, page 144]
<b>EXP</b> ( <i>numeric-expression</i> )	See “EXP function [Numeric]” [ASA SQL Reference, page 152]
<b>FLOOR</b> ( <i>numeric-expression</i> )	See “FLOOR function [Numeric]” [ASA SQL Reference, page 155]
<b>GETDATE</b> ( )	See “GETDATE function [Date and time]” [ASA SQL Reference, page 157]
<b>GREATER</b> ( <i>expression1</i> , <i>expression2</i> )	See “GREATER function [Miscellaneous]” [ASA SQL Reference, page 159]
<b>HEXTOINT</b> ( <i>hexadecimal-string</i> )	See “HEXTOINT function [Data type conversion]” [ASA SQL Reference, page 161]
<b>HOURL</b> ( <i>datetime-expression</i> )	See “HOURL function [Date and time]” [ASA SQL Reference, page 162]
<b>HOURS</b> ( [ <i>datetime-expression</i> , ] <i>datetime-expression</i> )	See “HOURS function [Date and time]” [ASA SQL Reference, page 162]
<b>HOURS</b> ( <i>datetime-expression</i> , <i>integer-expression</i> )	See “HOURS function [Date and time]” [ASA SQL Reference, page 162]
<b>IFNULL</b> ( <i>expression-1</i> , <i>expression-2</i> [ , <i>expression-3</i> ] )	See “IFNULL function [Miscellaneous]” [ASA SQL Reference, page 168]
<b>INSERTSTR</b> ( <i>integer-expression</i> , <i>string-expression-1</i> , <i>string-expression-2</i> )	See “INSERTSTR function [String]” [ASA SQL Reference, page 169]

Function	Remarks
<b>INTTOHEX</b> ( <i>integer-expression</i> )	See “ <a href="#">INTTOHEX function [Data type conversion]</a> ” [ <a href="#">ASA SQL Reference</a> , page 169]
<b>ISDATE</b> ( <i>string</i> )	See “ <a href="#">ISDATE function [Data type conversion]</a> ” [ <a href="#">ASA SQL Reference</a> , page 170]
<b>ISNULL</b> ( <i>expression</i> , <i>expression</i> [ , ... ] )	See “ <a href="#">ISNULL function [Data type conversion]</a> ” [ <a href="#">ASA SQL Reference</a> , page 170]
<b>LCASE</b> ( <i>string-expression</i> )	See “ <a href="#">LCASE function [String]</a> ” [ <a href="#">ASA SQL Reference</a> , page 172]
<b>LEFT</b> ( <i>string-expression</i> , <i>para-expression</i>	See “ <a href="#">LEFT function [String]</a> ” [ <a href="#">ASA SQL Reference</a> , page 172]
<b>LENGTH</b> ( <i>string-expression</i> )	See “ <a href="#">LENGTH function [String]</a> ” [ <a href="#">ASA SQL Reference</a> , page 173]
<b>LESSER</b> ( <i>expression1</i> , <i>expression2</i> )	See “ <a href="#">LESSER function [Miscellaneous]</a> ” [ <a href="#">ASA SQL Reference</a> , page 173]
<b>LIST</b> ( { <i>string-expression</i>   <b>DISTINCT</b> <i>column-name</i> } [ , <i>delimiter-string</i> ] )	<b>DISTINCT</b> <i>column-name</i> cannot be used from dynamic SQL. See “ <a href="#">LIST function [Aggregate]</a> ” [ <a href="#">ASA SQL Reference</a> , page 174]
<b>LOCATE</b> ( <i>string-expression-1</i> , <i>string-expression-2</i> [ , <i>integer-expression</i> ] )	See “ <a href="#">LOCATE function [String]</a> ” [ <a href="#">ASA SQL Reference</a> , page 176]
<b>LOG</b> ( <i>numeric-expression</i> )	See “ <a href="#">LOG function [Numeric]</a> ” [ <a href="#">ASA SQL Reference</a> , page 177]
<b>LOG10</b> ( <i>numeric-expression</i> )	See “ <a href="#">LOG10 function [Numeric]</a> ” [ <a href="#">ASA SQL Reference</a> , page 178]
<b>LOWER</b> ( <i>string-expression</i> )	See “ <a href="#">LOWER function [String]</a> ” [ <a href="#">ASA SQL Reference</a> , page 179]



Function	Remarks
<b>LTRIM</b> ( <i>string-expression</i> )	See “LTRIM function [String]” [ASA SQL Reference, page 179]
<b>MAX</b> ( <i>expression</i> )	See “MAX function [Aggregate]” [ASA SQL Reference, page 180]
<b>MIN</b> ( <i>expression</i> )	See “MIN function [Aggregate]” [ASA SQL Reference, page 180]
<b>MINUTE</b> ( <i>datetime-expression</i> )	See “MINUTE function [Date and time]” [ASA SQL Reference, page 181]
<b>MINUTES</b> ( [ <i>datetime-expression</i> , ] <i>datetime-expression</i> )	See “MINUTES function [Date and time]” [ASA SQL Reference, page 181]
<b>MINUTES</b> ( <i>datetime-expression</i> , <i>integer-expression</i> )	See “MINUTES function [Date and time]” [ASA SQL Reference, page 181]
<b>MOD</b> ( <i>dividend</i> , <i>divisor</i> )	See “MOD function [Numeric]” [ASA SQL Reference, page 182]
<b>MONTH</b> ( <i>date-expression</i> )	See “MONTH function [Date and time]” [ASA SQL Reference, page 183]
<b>MONTHNAME</b> ( <i>date-expression</i> )	See “MONTHNAME function [Date and time]” [ASA SQL Reference, page 183]
<b>MONTHS</b> ( [ <i>datetime-expression</i> , ] <i>datetime-expression</i> )	See “MONTHS function [Date and time]” [ASA SQL Reference, page 184]
<b>MONTHS</b> ( <i>datetime-expression</i> , <i>integer-expression</i> )	See “MONTHS function [Date and time]” [ASA SQL Reference, page 184]
<b>NEWID</b> ( )	This function is not supported by the Ultra-Lite static Java API. See “NEWID function [Miscellaneous]” [ASA SQL Reference, page 185]
<b>NOW</b> ( * )	See “NOW function [Date and time]” [ASA SQL Reference, page 189]

Function	Remarks
<b>NULLIF</b> ( <i>expression-1</i> , <i>expression-2</i> )	See “NULLIF function [Miscellaneous]” [ASA SQL Reference, page 189]
<b>PATINDEX</b> ( <i>'%pattern%'</i> , <i>string-expression</i> )	See “PATINDEX function [String]” [ASA SQL Reference, page 196]
<b>PI</b> ( * )	See “PI function [Numeric]” [ASA SQL Reference, page 197]
<b>POWER</b> ( <i>numeric-expression-1</i> , <i>numeric-expression-2</i> )	See “POWER function [Numeric]” [ASA SQL Reference, page 199]
<b>QUARTER</b> ( <i>date-expression</i> )	See “QUARTER function [Date and time]” [ASA SQL Reference, page 201]
<b>RADIANS</b> ( <i>numeric-expression</i> )	See “RADIANS function [Numeric]” [ASA SQL Reference, page 202]
<b>REMAINDER</b> ( <i>dividend</i> , <i>divisor</i> )	See “REMAINDER function [Numeric]” [ASA SQL Reference, page 212]
<b>REPEAT</b> ( <i>string-expression</i> , <i>integer-expression</i> )	See “REPEAT function [String]” [ASA SQL Reference, page 212]
<b>REPLACE</b> ( <i>original-string</i> , <i>search-string</i> , <i>replace-string</i> )	See “REPLACE function [String]” [ASA SQL Reference, page 213]
<b>REPLICATE</b> ( <i>string-expression</i> , <i>integer-expression</i> )	See “REPLICATE function [String]” [ASA SQL Reference, page 213]
<b>RIGHT</b> ( <i>string-expression</i> , <i>integer-expression</i> )	See “RIGHT function [String]” [ASA SQL Reference, page 216]
<b>ROUND</b> ( <i>numeric-expression</i> , <i>integer-expression</i> )	See “ROUND function [Numeric]” [ASA SQL Reference, page 216]

Function	Remarks
<b>RTRIM</b> ( <i>string-expression</i> )	See “RTRIM function [String]” [ <i>ASA SQL Reference</i> , page 218]
<b>SECOND</b> ( <i>datetime-expression</i> )	See “SECOND function [Date and time]” [ <i>ASA SQL Reference</i> , page 218]
<b>SECONDS</b> ( [ <i>datetime-expression</i> , ] <i>datetime-expression</i> )	See “SECONDS function [Date and time]” [ <i>ASA SQL Reference</i> , page 219]
<b>SECONDS</b> ( <i>datetime-expression</i> , <i>integer-expression</i> )	See “SECONDS function [Date and time]” [ <i>ASA SQL Reference</i> , page 219]
<b>SIGN</b> ( <i>numeric-expression</i> )	See “SIGN function [Numeric]” [ <i>ASA SQL Reference</i> , page 220]
<b>SIMILAR</b> ( <i>string-expression-1</i> , <i>string-expression-2</i> )	See “SIMILAR function [String]” [ <i>ASA SQL Reference</i> , page 221]
<b>SIN</b> ( <i>numeric-expression</i> )	See “SIN function [Numeric]” [ <i>ASA SQL Reference</i> , page 221]
<b>SOUNDEX</b> ( <i>string-expression</i> )	See “SOUNDEX function [String]” [ <i>ASA SQL Reference</i> , page 225]
<b>SPACE</b> ( <i>integer-expression</i> )	See “SPACE function [String]” [ <i>ASA SQL Reference</i> , page 226]
<b>SQRT</b> ( <i>numeric-expression</i> )	See “SQRT function [Numeric]” [ <i>ASA SQL Reference</i> , page 227]
<b>STR</b> ( <i>numeric-expression</i> [ , <i>length</i> [ , <i>decimal</i> ] ] )	See “STR function [String]” [ <i>ASA SQL Reference</i> , page 229]
<b>STRING</b> ( <i>string-expression</i> [ , ... ] )	See “STRING function [String]” [ <i>ASA SQL Reference</i> , page 230]

Function	Remarks
<code>STRTOUUID ( string-expression )</code>	This function is not supported by the Ultra-Lite static Java API. See “STRTOUUID function [String]” [ASA SQL Reference, page 230]
<code>STUFF ( string-expression1, start, length, string-expression2 )</code>	See “STUFF function [String]” [ASA SQL Reference, page 231]
<code>{ SUBSTRING   SUBSTR }( string-expression, start [, length ] )</code>	See “SUBSTRING function [String]” [ASA SQL Reference, page 232]
<code>SUM ( expression   DISTINCT column-name )</code>	<b>DISTINCT</b> column-name cannot be used from dynamic SQL. See “SUM function [Aggregate]” [ASA SQL Reference, page 233]
<code>TAN ( numeric-expression )</code>	See “TAN function [Numeric]” [ASA SQL Reference, page 234]
<code>TODAY ( * )</code>	See “TODAY function [Date and time]” [ASA SQL Reference, page 235]
<code>TRIM ( string-expression )</code>	See “TRIM function [String]” [ASA SQL Reference, page 236]
<code>"TRUNCATE" ( numeric-expression, integer-expression )</code>	See “TRUNCATE function [Numeric]” [ASA SQL Reference, page 236]
<code>TRUNCNUM ( numeric-expression, integer-expression )</code>	See “TRUNCNUM function [Numeric]” [ASA SQL Reference, page 237]
<code>UCASE ( string-expression )</code>	See “UCASE function [String]” [ASA SQL Reference, page 238]
<code>UPPER ( string-expression )</code>	See “UPPER function [String]” [ASA SQL Reference, page 238]

Function	Remarks
<b>UIDTOSTR</b> ( <i>uuid-expression</i> )	This function is not supported by the Ultra-Lite static Java API. See “UIDTOSTR function [String]” [ <i>ASA SQL Reference</i> , page 239]
<b>WEEKS</b> ( [ <i>datetime-expression</i> , ] <i>datetime-expression</i> )	See “WEEKS function [Date and time]” [ <i>ASA SQL Reference</i> , page 243]
<b>WEEKS</b> ( <i>datetime-expression</i> , <i>integer-expression</i> )	See “WEEKS function [Date and time]” [ <i>ASA SQL Reference</i> , page 243]
<b>YEAR</b> ( [ <i>datetime-expression</i> , ] <i>datetime-expression</i> )	See “YEAR function [Date and time]” [ <i>ASA SQL Reference</i> , page 249]
<b>YEARS</b> ( <i>datetime-expression</i> , <i>integer-expression</i> )	See “YEARS function [Date and time]” [ <i>ASA SQL Reference</i> , page 250]
<b>YMD</b> ( <i>integer-expression</i> , <i>integer-expression</i> , <i>integer-expression</i> )	See “YMD function [Date and time]” [ <i>ASA SQL Reference</i> , page 251]



---

## CHAPTER 8

# Dynamic SQL

### About this chapter

Dynamic SQL is the version of SQL available to UltraLite components. This chapter describes the features of the dynamic SQL in UltraLite.

Dynamic SQL statements can be constructed at run time. This is in contrast to the static SQL available to embedded SQL, static C++ API, and static Java API applications, which must have all SQL statements specified at compile time.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Introduction to dynamic SQL</a>	160
<a href="#">Dynamic SQL expressions</a>	163
<a href="#">Dynamic SQL operators</a>	166
<a href="#">Dynamic SQL search conditions</a>	170
<a href="#">Dynamic SQL statements</a>	172
<a href="#">Query optimization</a>	185

---

## Introduction to dynamic SQL

Structured Query Language (SQL) can be used by an application to perform a database task, such as retrieving information using a query or inserting a new row into a table. SQL is a relational database language standardized by the ANSI and ISO standards bodies. UltraLite dynamic SQL is a variant designed for use on small-footprint devices.

SQL statements are supplied as strings in function calls from the programming language you are using. UltraLite components provide functions for building and generating SQL statements. The programming interface delivers the SQL statement to the database. The database receives the statement and executes it, returning the required information (such as query results) back to the application.

Queries are one form of Data Manipulation Language used in SQL. In fact, the “Q” in “SQL” stands for query. You query, or retrieve, data from a database with a `SELECT` statement. A query produces a result set, which is a collection of rows that satisfy the query. The basic query operations in a relational system are projection, restriction, and join. The `SELECT` statement implements all of them.

A projection is a subset of the columns in a table. A restriction, also called selection, is a subset of the rows in a table, based on some conditions. For example, the following `SELECT` statement retrieves the names and prices of all products that cost more than \$15:

```
SELECT name, unit_price
FROM product
WHERE unit_price > 15
```

This query uses both a projection, as shown in the `SELECT` clause, and a restriction, given in the `WHERE` clause.

You can do more with dynamic SQL than just query the data. It also includes statements that modify data (the `INSERT`, `UPDATE`, and `DELETE` statements), statements that control transactions (`COMMIT` and `ROLLBACK`) and statements for creating and dropping tables and indexes (`CREATE` or `DROP TABLE` or `INDEX`).

### Availability

Dynamic SQL is the variant of SQL available for UltraLite components. UltraLite static interfaces use a different variant of SQL. The UltraLite components can use a table-based interface as well as dynamic SQL.

☞ For a comparison of these data access methods, see [“Choosing between components and static interfaces”](#) on page 11.



## Using dynamic SQL

Dynamic SQL can be used from UltraLite components, but not from static development models. The steps in executing dynamic SQL statements are common to all components:

1. Prepare the statement using a prepared statement method on the connection object. The name of the method varies slightly with the interface.

Preparing a statement causes the character string representing the statement to be parsed and optimized (prepared) and returns an object representing the prepared statement. The optimization is necessarily less involved than that in Adaptive Server Anywhere.

2. Set the value of any parameters.

Optionally, when the statement has input parameters (identified by question marks), then your application can call methods on the prepared statement object to set the value of these parameters. Any parameters for which values are not set are set to NULL.

3. Execute the statement.

If the statement is an INSERT, UPDATE, or DELETE, use the `ExecuteStatement` method. This method returns the number of rows modified by the statement.

If the statement is a SELECT statement, use the `ExecuteQuery` method. This method returns an object that holds the query result set.

4. For queries, navigate the result set and access the values in the result set.

- ◆ You can use methods on the result set object to set the position to different rows in the result set. Some examples are `MoveNext`, `MovePrevious`, `MoveFirst`, `MoveLast`, `Relative`, `BeforeFirst`, and `AfterLast`.
- ◆ When the current position is at a row of the result set, the values of columns in the result set can be obtained by methods that get values. The names of the methods depend on the interface. The methods convert data to application data types automatically. For example, an integer result expression can automatically be converted to a string if the result is assigned to a string variable.

5. For repeated execution of a prepared statement, repeat steps 2 through 4. Using parameters is more efficient than preparing the entire statement again.

---

The values for input variables persist after a prepared statement is executed. If you use a different value, you must reset the value of the parameter.

## Dynamic SQL expressions

Expressions in UltraLite dynamic SQL are built from column names, constants, and operators. Expressions evaluate to a value, and so have data types associated with them.

Syntax

```
expression :
  constant
| column-name
| - expression
| expression operator expression
| ( expression )
| function-name ( expression, ... )
| if-expression
| case-expression
```

See also

- ◆ [“Subqueries in expressions” on page 163](#)
- ◆ [“IF expressions” on page 164](#)
- ◆ [“CASE expressions” on page 164](#)
- ◆ [“UltraLite SQL functions” on page 148](#)
- ◆ [“Dynamic SQL operators” on page 166](#) [“CASE expressions” on page 164](#) [“Dynamic SQL operators” on page 166](#)

Aggregate expressions

An aggregate expression calculates a single value from a range of rows. For example, the following query computes the total payroll for employees in the employee table. In this query, `SUM( salary )` is an aggregate expression:

```
SELECT sum( salary )
FROM employee
```

An aggregate expression is one in which either an aggregate function is used, or in which one or more of the operands is an aggregate expression.

When a SELECT statement does not have a GROUP BY clause, the expressions in the *select-list* must be either all aggregate expressions or none of the expressions can be an aggregate expression. When a SELECT statement does have a GROUP BY clause, any non-aggregate expression in the *select-list* must appear in the GROUP BY list.

### Subqueries in expressions

A subquery is a SELECT statement that is nested inside another SELECT statement. Subqueries can be used as a table expression in the FROM clause, where they are also called derived tables.

Subqueries can be written with references to names that are specified before (to the left of) the subquery, sometimes known as outer references to the left.

---

There can be no references to items within subqueries, sometimes known as inner references. In the case of derived tables, it is required to have a derived table name and to specify the column names by which values in the SELECT list are fetched.

☞ For other uses of subqueries, see [“Dynamic SQL search conditions” on page 170](#).

## IF expressions

The syntax of the IF expression is as follows:

```
IF condition  
THEN expression1  
[ ELSE expression2 ]  
ENDIF
```

This expression returns the following:

- ◆ If *condition* is TRUE, the IF expression returns *expression1*.
- ◆ If *condition* is FALSE, the IF expression returns *expression2*.
- ◆ If *condition* is FALSE, and there is no *expression2*, the IF expression returns NULL.
- ◆ If *condition* is UNKNOWN, the IF expression returns NULL.

☞ For more information about TRUE, FALSE and UNKNOWN conditions, see [“NULL value” \[ASA SQL Reference, page 49\]](#), and [“Search conditions” \[ASA SQL Reference, page 23\]](#).

## CASE expressions

The CASE expression provides conditional SQL expressions. Case expressions can be used anywhere an expression can be used.

Syntax 1

```
CASE expression  
WHEN expression THEN expression, ...  
[ ELSE expression ]  
END
```

If the expression following the CASE statement is equal to the expression following the WHEN statement, then the expression following the THEN statement is returned. Otherwise the expression following the ELSE statement is returned, if it exists.

For example, the following code uses a case expression as the second clause in a SELECT statement.

```

SELECT id,
       ( CASE name
         WHEN 'Tee Shirt' then 'Shirt'
         WHEN 'Sweatshirt' then 'Shirt'
         WHEN 'Baseball Cap' then 'Hat'
         ELSE 'Unknown'
        END ) as Type
FROM Product

```

## Syntax 2

```

CASE
WHEN search-condition
THEN expression, ...
[ ELSE expression ]
END

```

If the search-condition following the WHEN statement is satisfied, the expression following the THEN statement is returned. Otherwise the expression following the ELSE statement is returned, if it exists.

For example, the following statement uses a case expression as the third clause of a SELECT statement to associate a string with a search-condition.

```

SELECT id, name,
       ( CASE
         WHEN name='Tee Shirt' then 'Sale'
         WHEN quantity >= 50 then 'Big Sale'
         ELSE 'Regular price'
        END ) as Type
FROM Product

```

NULLIF function for abbreviated CASE expressions

The NULLIF function provides a way to write some CASE statements in short form. The syntax for NULLIF is as follows:

```

NULLIF ( expression-1, expression-2 )

```

NULLIF compares the values of the two expressions. If the first expression equals the second expression, NULLIF returns NULL. If the first expression does not equal the second expression, NULLIF returns the first expression.

---

## Dynamic SQL operators

Operators are used to compare, combine, or modify expressions. Dynamic SQL supports the operators listed in this section. UltraLite static interfaces have access to all of the Adaptive Server Anywhere operators.

☞ For information about operators in Adaptive Server Anywhere, see “Operators” [*ASA SQL Reference*, page 11].

### Binary comparison operators

The syntax for binary comparison conditions is as follows:

*expression compare expression*

where *compare* is a comparison operator. The following comparison operators are available:

Operator	Description
=	Equal to
[ NOT ] LIKE	A text comparison, possibly using regular expressions
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to
<>	Not equal to
!>	Not greater than
!<	Not less than

◆ **Case sensitivity** Comparisons are carried out with the same attention to case as the database on which they are operating. By default, UltraLite databases are created as case insensitive.

◆ **NULL operators** Comparisons involving NULL expressions follow these rules:

Two null values compare as equals. When exactly one of the operands being compared is NULL, the result is UNKNOWN. Thus, SQL comparisons produce one of three results (TRUE, FALSE, and

UNKNOWN). Similarly, logical expressions (AND, OR, NOT) can also produce these results.

## Arithmetic operators

- expression + expression** Addition. If either expression is NULL, the result is NULL.
- expression – expression** Subtraction. If either expression is NULL, the result is NULL.
- expression** Negation. If the expression is NULL, the result is NULL.
- expression \* expression** Multiplication. If either expression is NULL, the result is NULL.
- expression / expression** Division. If either expression is NULL or if the second expression is 0, the result is NULL.
- expression % expression** Modulo finds the integer remainder after a division involving two whole numbers. For example,  $21 \% 11 = 10$  because 21 divided by 11 equals 1 with a remainder of 10.

## String operators

- expression || expression** String concatenation (two vertical bars). If either string is NULL, it is treated as the empty string for concatenation.
- expression + expression** Alternative string concatenation. When using the + concatenation operator, you must ensure the operands are explicitly set to character data types rather than relying on implicit data conversion.

For example, the following query returns the integer value **579**:

```
SELECT 123 + 456
```

whereas the following query returns the character string **123456**:

```
SELECT '123' + '456'
```

You can use the CAST or CONVERT function to explicitly convert data types.

## Bitwise operators

The following operators can be used on integer data types in UltraLite.

---

Operator	Description
&	bitwise AND
	bitwise OR
^	bitwise exclusive OR
~	bitwise NOT

The bitwise operators &, | and ~ are not interchangeable with the logical operators AND, OR, and NOT. The bitwise operators operate on integer values using the bit representation of the values.

#### Example

For example, the following statement selects rows in which the correct bits are set.

```
SELECT *
FROM tableA
WHERE (options & 0x0101) <> 0
```

## Logical operators

Logical operators compare conditions (AND, OR, and NOT) or test the truth or NULL value nature of expressions (IS)

Conditions are combined using AND as follows:

*condition1* **AND** *condition2*

The combined condition is TRUE if both conditions are TRUE, FALSE if either condition is FALSE, and UNKNOWN otherwise.

Conditions are combined using OR as follows:

*condition1* **OR** *condition2*

The combined condition is TRUE if either condition is TRUE, FALSE if both conditions are FALSE, and UNKNOWN otherwise.

The syntax for the NOT operator is as follows:

**NOT** *condition*

The NOT condition is TRUE if *condition* is FALSE, FALSE if *condition* is TRUE, and UNKNOWN if *condition* is UNKNOWN.

The IS operator provides a means to test a logical value. The syntax for the IS operator is as follows:

*expression* **IS** [ **NOT** ] { *truth-value* | **NULL** }



The condition is TRUE if the *expression* evaluates to the supplied *truth-value*, which must be one of TRUE, FALSE, or UNKNOWN. Otherwise, the value is FALSE.

## Operator precedence

The precedence of operators in expressions is as follows. The operators at the top of the list are evaluated before those at the bottom of the list.

1. Names, functions, constants
2. ()
3. unary operators (operators that require a single operand): +, -
4. ~
5. &, |, ^
6. \*, /, %
7. +, -
8. ||
9. Comparisons: >, <, <>, !=, <=, >=, [ NOT ] BETWEEN, [ NOT ] IN, [ NOT ] LIKE
10. Comparisons: IS [NOT] TRUE, FALSE, UNKNOWN
11. NOT
12. AND
13. OR

When you use more than one operator in an expression, it is recommended that you make the order of operation explicit using parentheses rather than relying on an identical operator precedence in UltraLite.

---

## Dynamic SQL search conditions

Search conditions appear in the WHERE clause or the ON phrase in SQL queries. The following search conditions are supported in dynamic SQL.

Syntax

```
search-condition:  
expression compare expression  
| expression IS [ NOT ] { NULL | TRUE | FALSE | UNKNOWN }  
| expression [ NOT ] BETWEEN expression AND expression  
| expression [ NOT ] IN ( expression, ... )  
| expression [ NOT ] IN ( subquery )  
| expression [ NOT ] { ANY | ALL } ( subquery )  
| expression [ NOT ] EXISTS ( subquery )  
| NOT search-condition  
| search-condition AND search-condition  
| search-condition OR search-condition  
| ( search-condition )
```

### ALL conditions

The syntax for ALL conditions is

```
expression compare [ NOT ] ALL ( subquery )
```

where *compare* is a comparison operator.

### ANY conditions

The syntax for ANY conditions is

```
expression compare [ NOT ] ANY ( subquery )
```

where *compare* is a comparison operator.

For example, an ANY condition with an equality operator,

```
expression = ANY ( subquery )
```

is TRUE if *expression* is equal to any of the values in the result of the subquery, and FALSE if the expression is not NULL and does not equal any of the values returned by the subquery. The ANY condition is UNKNOWN if *expression* is the NULL value, unless the result of the subquery has no rows, in which case the condition is always FALSE.

### BETWEEN conditions

The syntax for BETWEEN conditions is as follows:

```
expression [ NOT ] BETWEEN start-expression AND end-expression
```

The BETWEEN condition can evaluate as TRUE, FALSE, or UNKNOWN. Without the NOT keyword, the condition evaluates as TRUE if *expression* is between *start-expression* and *end-expression*. The NOT keyword reverses the meaning of the condition but leaves UNKNOWN unchanged.

The BETWEEN conditions is equivalent to a combination of two inequalities:

```
[ NOT ] ( expression >= start-expression
          AND expression <= end-expression )
```

## EXISTS conditions

The syntax for EXISTS conditions is as follows:

```
[ NOT ] EXISTS( subquery )
```

The EXISTS condition is TRUE if the subquery result contains at least one row, and FALSE if the subquery result does not contain any rows. The EXISTS condition cannot be UNKNOWN.

## IN conditions

The syntax for IN conditions is as follows:

```
expression [ NOT ] IN { ( subquery ) | ( value-expr, ... ) }
```

An IN condition, without the NOT keyword, evaluates according to the following rules:

- ◆ TRUE if *expression* is not NULL and equals at least one of the values.
- ◆ UNKNOWN if *expression* is NULL and the values list is not empty, or if at least one of the values is NULL and *expression* does not equal any of the other values.
- ◆ FALSE if *expression* is NULL and *subquery* returns no values; or if *expression* is not NULL, none of the values are NULL, and *expression* does not equal any of the values.

The NOT keyword interchanges TRUE and FALSE.

The search condition *expression* IN ( *values* ) is identical to the search condition *expression* = ANY ( *values* ). The search condition *expression* NOT IN ( *values* ) is identical to the search condition *expression* <> ALL ( *values* ).

The *value-expr* arguments are expressions that take on a single value, which may be a string, a number, a date, or any other SQL data type.

---

# Dynamic SQL statements

The following are dynamic SQL statements that you can use in UltraLite.

## COMMIT statement

Description	Use this statement to make changes to the database permanent, or to terminate a user-defined transaction.
Syntax	<b>COMMIT [ WORK ]</b>
Usage	<p>A transaction is the logical unit of work done on one database connection to a database between COMMIT or ROLLBACK statements. The COMMIT statement ends a transaction and makes all changes made during this transaction permanent in the database.</p> <p>CREATE and DROP statements both carry out a COMMIT automatically.</p>
Side effects	Closes all cursors.

## CREATE INDEX statement

Description	<p>Use this statement to create an index on a specified table. Indexes can improve query performance by providing quick ways for UltraLite to look up specific rows.</p> <p>☞ For more information about indexes, see <a href="#">“Query optimization” on page 185</a>.</p>
Syntax	<b>CREATE [ UNIQUE ] INDEX</b> <i>index-name</i> <b>ON</b> [ <i>owner.</i> ] <i>table-name</i> ( <i>column-name</i> , ... )
Parameters	<p><b>UNIQUE keyword</b> The UNIQUE attribute ensures that there will not be two rows in the table with identical values in all the columns in the index. Each index key must be unique or contain a NULL in at least one column.</p> <p>There is a difference between a unique constraint on a table and a unique index. Columns of a unique index are allowed to be NULL, while columns in a unique constraint are not. A foreign key can reference either a primary key or a column with a unique constraint, but not a unique index, because it can include multiple instances of NULL.</p> <p>UltraLite tables do not have owners. The optional <i>owner</i> is supported as a convenience for existing SQL and for programmatically-generated SQL. UltraLite accepts but ignores <i>owner</i>.</p>
Usage	The CREATE INDEX statement creates a sorted index on the specified columns of the named table. Indexes are automatically used to improve the performance of queries issued to the database, and to sort queries with an

ORDER BY clause. Once an index is created, it is never referenced in a SQL statement again except to delete it with DROP INDEX.

Indexes do take space in the database. Also, the additional work required to maintain indexes can affect the performance of data modification operations. For these reasons, you should avoid creating indexes that do not assist in query performance.

- ◆ **Exclusive table use** CREATE INDEX is prevented whenever the statement affects a table currently being used by another connection. CREATE INDEX can be time consuming and the server will not process requests referencing the same table while the statement is being processed.
- ◆ **Automatically created indexes** UltraLite automatically creates indexes for primary keys and for unique constraints.

## CREATE TABLE statement

**Description** Use this statement to create a table in the database.

**Syntax**

```
CREATE TABLE [ owner.] table-name
( { column-definition | table-constraint }, ... )
```

*column-definition* :

```
column-name data-type [ NOT NULL ]
[ DEFAULT default-value ]
[ UNIQUE | PRIMARY KEY ]
```

*default-value* :

```
| constant-expression
| AUTOINCREMENT
| GLOBAL AUTOINCREMENT [ ( partition-size ) ]
| NULL
| NEWID ( )
| CURRENT DATE
| CURRENT TIME
| CURRENT TIMESTAMP
```

*table-constraint* :

```
  UNIQUE ( column-name, ... )
  | PRIMARY KEY ( column-name, ... )
  | foreign-key-constraint
```

*foreign-key-constraint* :

```
[ NOT NULL ] FOREIGN KEY ( column-name, ... )
REFERENCES table-name ( column-name, ... )
[ CHECK ON COMMIT ]
```

---

## Parameters

**column-definition** Define a column in the table. The following are part of column definitions.

- ◆ **column-name** The column name is an identifier. Two columns in the same table cannot have the same name.
- ◆ **data-type** For information on data types, see [“Data types in UltraLite” on page 145](#).
- ◆ **NOT NULL** If NOT NULL is specified, or if the column is in a UNIQUE or PRIMARY KEY constraint, the column cannot contain NULL in any row.
- ◆ **DEFAULT** The DEFAULT value is used as the value for the column in any INSERT statement that does not specify a value for the column. If no DEFAULT is specified, it is equivalent to DEFAULT NULL.
  - **constant-expression** The default value is the supplied constant expression. Only constant expressions that do not reference database objects are allowed in a DEFAULT clause. If the expression is not a simple value, for example if it is an expression involving an addition operator, it must be enclosed in parentheses.
  - **AUTOINCREMENT** The default value is an autoincremented value for each row in the table. When using AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type.

On inserts into the table, if a value is not specified for the AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column, it is used; if the specified value is larger than the current maximum value for the column, that value will be used as a starting point for subsequent inserts.

Deleting rows does not decrement the AUTOINCREMENT counter. Gaps created by deleting rows can only be filled by explicit assignment when using an insert. After an explicit insert of a row number less than the maximum, subsequent rows without explicit assignment are still automatically incremented with a value of one greater than the previous maximum.

The next value to be used for each column is stored as an integer. Using values greater than  $(2^{31} - 1)$  may cause wraparound to incorrect values, and AUTOINCREMENT should not be used in such cases.
  - **GLOBAL AUTOINCREMENT** The default value is an autoincremented value within a partition of values provided for this database. The GLOBAL AUTOINCREMENT default is intended for use when multiple databases will be used in a synchronization environment.

This default is similar to AUTOINCREMENT, except that the domain is partitioned. Each partition contains the same number of values. You assign each copy of the database a unique global database identification number.

The partition size can be specified in parentheses immediately following the AUTOINCREMENT keyword. The partition size may be any positive integer, although the partition size is generally chosen so that the supply of numbers within any one partition will rarely, if ever, be exhausted.

If the column is of type BIGINT or UNSIGNED BIGINT, the default partition size is  $2^{32} = 4294967296$ ; for columns of all other types the default partition size is  $2^{16} = 65536$ . Since these defaults may be inappropriate, especially if the column is not of type INT or BIGINT, it is best to specify the partition size explicitly.

When using this default, the Global Database ID of each database must be set to a unique, non-negative integer. This value uniquely identifies the database and indicates from which partition default values are to be assigned. The range of allowed values is  $n p + 1$  to  $(n + 1) p$ , where  $n$  is the value of the Global Database ID and  $p$  is the partition size.

For example, if you define the partition size to be 1000 and set the Global Database ID to 3, then the range is from 3001 to 4000.

If the previous value is less than  $(n + 1) p$ , the next default value will be one greater than the previous largest value in column. If the column contains no values, the first default value is  $n p + 1$ . Default column values are not affected by values in the column outside of the current partition; that is, by numbers less than  $p n + 1$  or greater than  $p(n + 1)$ . Such values may be present if they have been replicated from another database via MobiLink synchronization.

Because the Global Database ID cannot be set to negative values, the values chosen are always positive. The maximum identification number is restricted only by the column data type and the partition size. If the Global Database ID is set to the default value of 2147483647, a NULL is inserted into the column. Should null values not be permitted, attempting to insert the row causes an error. This situation arises, for example, if the column is contained in the table's primary key.

NULL default values are generated when the supply of values within the partition has been exhausted. In this case, a new Global Database ID should be assigned to the database to allow default values to be chosen from another partition. Attempting to insert the null value causes an error if the column does not permit nulls.

- **NULL** The default value is NULL. A column that is DEFAULT NULL is equivalent to a column that allows NULL and has no explicit default value.

- 
- **NEWID** The default value is a universal unique identifier (UUID). UUIDs can be used as an alternative to GLOBAL AUTOINCREMENT to uniquely identify rows in a table even in a synchronized environment of many databases. The values are generated such that a value produced on one computer will not match that produced on another. Hence they can also be used as keys in synchronization environments. The column type must be a UNIQUEIDENTIFIER or a BINARY column of size at least 16.  
NEWID is a non-deterministic function. Successive calls to NEWID may return different values.
  - **CURRENT DATE** The default value is the current date.
  - **CURRENT TIME** The default value is the current time.
  - **CURRENT TIMESTAMP** The default value is the current timestamp, which specifies a date and time.

◆ **UNIQUE** Imposes the restriction that all values in the column must be distinct. A column that is declared UNIQUE does not allow NULL.

◆ **PRIMARY KEY** Declares the column as the primary key for the table. The primary key column cannot hold NULL, and each row must be unique.

A table can have only one primary key, although that key may be composed of more than one column. To create a table with a multiple column primary key, declare the primary key in a table constraint.

**table-constraint** A table constraint restricts the values that one or more columns in the table can hold.

Table constraints help ensure the integrity of data in the database. UltraLite prevents the execution of statements that cause constraint violations and reports an error.

Table constraints must be used instead of column constraints when the constraint references more than one column in the table.

◆ **UNIQUE** Identifies one or more columns that uniquely identify each row in the table. No two rows in the table can have the same values in all the named column(s). A table may have more than one unique constraint.

Columns of a unique index are allowed to be NULL, while columns in a unique constraint are not. A foreign key can reference either a primary key or a unique constraint, but not a unique index, because it can include multiple instances of NULL.

☞ For information about unique indexes, see [“CREATE INDEX statement” on page 172](#).



- ◆ **PRIMARY KEY** This is the same as a unique constraint, except that a table can have only one primary key constraint.

Columns included in primary keys cannot allow NULL. Each row in the table has a unique primary key value. A table can have only one PRIMARY KEY.

- ◆ **foreign-key-constraint** A foreign key constraint restricts the values for a set of columns to match the values in a primary key or, less commonly, a unique constraint of another table (the primary table). For example, a foreign key constraint could be used to ensure that a customer number in an invoice table corresponds to a customer number in the customer table.

If you do not explicitly define a foreign key column, it is created with the same data type as the corresponding column in the primary table. These automatically-created columns cannot be part of the primary key of the foreign table. Thus, a column used in both a primary key and foreign key of the same table must be explicitly created.

If foreign key column names are specified, then primary key column names must also be specified, and the column names are paired according to position in the lists. If the primary table column names are not specified in a FOREIGN KEY table constraint, then the primary key columns are used. If foreign key column names are not specified then the foreign key columns are given the same names as the columns in the primary table.

If at least one value in a multi-column foreign key is NULL, there is no restriction on the values that can be held in other columns of the key.

The CHECK ON COMMIT clause causes UltraLite to wait for a COMMIT before checking that the action does not violate the foreign key constraint.

UltraLite tables do not have owners. The optional *owner* is supported as a convenience for existing SQL and for programmatically-generated SQL. UltraLite accepts but ignores *owner*.

## DELETE statement

Description	Use this statement to delete rows from the database.
Syntax	<b>DELETE</b> [ <b>FROM</b> ] [ <i>owner</i> .] <i>table-name</i> [ <b>WHERE</b> <i>search-condition</i> ]
Usage	The DELETE statement deletes all the rows that satisfy the search condition from the named table.

---

**Caution**

*If no WHERE clause is specified, all rows from the named table are deleted.*

UltraLite tables do not have owners. The optional *owner* is supported as a convenience for existing SQL and for programmatically-generated SQL. UltraLite accepts but ignores *owner*.

Side effects

None.

Example

Remove employee 105 from the database.

```
DELETE
FROM employee
WHERE emp_id = 105
```

Remove all data prior to 2000 from the `fin_data` table.

```
DELETE
FROM fin_data
WHERE year < 2000
```

## DROP INDEX statement

Description

Use this statement to permanently remove an index definition from the database.

Syntax

```
DROP INDEX [[ owner.]table-name.]index-name
```

Usage

The DROP INDEX statement removes the definition of the indicated index.

DROP INDEX is prevented whenever the statement affects a table that is currently being used by another connection.

UltraLite tables do not have owners. The optional *owner* is supported as a convenience for existing SQL and for programmatically-generated SQL. UltraLite accepts but ignores *owner*.

## DROP TABLE statement

Description

Use this statement to permanently remove a table definition and all data in the table from a database.

Syntax

```
DROP TABLE [ owner.]table-name
```

Usage

The DROP TABLE statement removes the definition of the indicated table. All data in the table is automatically deleted as part of the dropping process. Also, all indexes and keys for the table are dropped by the DROP TABLE statement.

DROP TABLE is prevented whenever the statement affects a table that is

currently being used by another connection.

UltraLite tables do not have owners. The optional *owner* is supported as a convenience for existing SQL and for programmatically-generated SQL. UltraLite accepts but ignores *owner*.

## INSERT statement

Description	Use this statement to insert a single row into a table (Syntax 1) or to insert the results from a SELECT statement into the table (Syntax 2).
Syntax 1	<b>INSERT [ INTO ] [ <i>owner</i>.]<i>table-name</i> [ ( <i>column-name</i>, ... ) ] <b>VALUES</b> ( <i>expression</i>, ... )</b>
Syntax 2	<b>INSERT [ INTO ] [ <i>owner</i>.]<i>table-name</i> [ ( <i>column-name</i>, ... ) ] <b>SELECT statement</b></b>
Usage	<p>The INSERT statement is used to add new rows to a database table.</p> <p>Insert a single row with the specified expression values. If the optional list of column names is given, the values are inserted one for one into the specified columns. If the list of column names is not specified, the values are inserted into the table columns in the order they were created (the same order as retrieved with SELECT *). The row is inserted into the table at an arbitrary position.</p> <p>If you specify column names, the columns from the select list are matched ordinally with the columns specified in the column list, or sequentially in the order in which the columns were created.</p> <p>Character strings inserted into tables are always stored in the same case as they are entered, regardless of whether the database is case sensitive or not. Thus a string <b>Value</b> inserted into a table is always held in the database with an upper-case V and the remainder of the letters lower case. SELECT statements return the string as <b>Value</b>. If the database is not case sensitive, however, all comparisons make <b>Value</b> the same as <b>value</b>, <b>VALUE</b>, and so on. Further, if a single-column primary key already contains an entry <b>Value</b>, an INSERT of <b>value</b> is rejected, as it would make the primary key not unique.</p> <p>UltraLite tables do not have owners. The optional <i>owner</i> is supported as a convenience for existing SQL and for programmatically-generated SQL. UltraLite accepts but ignores <i>owner</i>.</p>
Side effects	None.
Examples	Add an Eastern Sales department to the database.

---

```
INSERT
INTO department ( dept_id, dept_name )
VALUES ( 230, 'Eastern Sales' )
```

## ROLLBACK statement

Description	Use this statement to end a transaction and undo any changes made since the last COMMIT or ROLLBACK.
Syntax	<b>ROLLBACK [ WORK ]</b>
Usage	A transaction is the logical unit of work done on one database connection to a database between COMMIT or ROLLBACK statements. The ROLLBACK statement ends the current transaction and undoes all changes made to the database since the previous COMMIT or ROLLBACK.

## SELECT statement

Description	Use this statement to retrieve information from the database.
Syntax	<pre><b>SELECT [ DISTINCT ] [ FIRST   TOP n ] select-list</b> <b>[ FROM table-expression ]</b> <b>[ WHERE search-condition ]</b> <b>[ GROUP BY group-by-expression,...group-by-expression ]</b> <b>[ HAVING search-condition ]</b> <b>[ ORDER BY order-by-expression,...order-by-expression ]</b> <b>[ OPTION ( FORCE ORDER ) ]</b></pre> <p><i>table-expression</i> :</p> <pre>[ owner.]table-name [ [ AS ] correlation-name ]   table-expression { join-operator table-expression     [ ON join-condition ] ,... }   ( table-expression, ... )   ( select-statement ) [ AS ]    derived-table-name ( [ column-name, ...     ] column-name )</pre> <p><i>join-operator</i> :</p> <pre>, (ON condition not allowed)   <b>CROSS JOIN</b> (ON condition not allowed)   <b>INNER JOIN</b>   <b>JOIN</b> (ON phrase required)   <b>LEFT OUTER JOIN</b></pre> <p><i>order-by-expression</i> :</p> <pre>{ integer   expression } [ <b>ASC</b>   <b>DESC</b> ]</pre>
Parameters	<b>DISTINCT</b> If you do not specify DISTINCT, all rows that satisfy the clauses of the SELECT statement. If DISTINCT is specified, duplicate

output rows are eliminated. Many statements take significantly longer to execute when `DISTINCT` is specified, so you should reserve `DISTINCT` for cases where it is necessary.

**FIRST or TOP** You can explicitly retrieve only the first row of a query or the first *n* rows of a query. These keywords are principally for use with `ORDER BY` queries.

**select-list** The *select-list* is a list of expressions, separated by commas, specifying what will be retrieved from the database. An asterisk (\*) means select all columns of all tables in the `FROM` clause. Subqueries are not allowed in the *select-list*.

An alias name can be specified following an expression in the *select-list* to represent that expression. The alias name can then be used elsewhere in the query, such as in the `WHERE` clause or `ORDER BY` clause.

**FROM clause** Rows are retrieved from the tables and views specified in the *table-expression*. The *table-expression* is built from base tables and subqueries, as listed in the syntax above.

For information on expressions, see [“Dynamic SQL expressions” on page 163](#).

**ON condition** The `ON` condition is specified for a single join operation and indicates how the join is to create rows in the result set. A `WHERE` clause is used to restrict the rows in the result set, after potential rows have been created by a join. For `INNER` joins restricting with an `ON` or `WHERE` is equivalent. For `OUTER` joins, they are not equivalent.

**WHERE clause** This clause limits the rows that are selected from the tables named in the `FROM` clause. It can be used to restrict rows between multiple tables.

Although both the `ON` phrase (which is part of the `FROM` clause) and the `WHERE` clause restrict the rows in the result set, they differ in that the `WHERE` clause is applied at a later stage of query execution. The `ON` phrase is part of the join operation between tables, while the `WHERE` clause is applied after the join is complete. In some queries, a condition can be specified in a `WHERE` clause or in the `ON` phrase with the same net result, but in other cases the results differ. For example, for outer joins, a condition specified in a `WHERE` clause gives different results from the same condition specified in the `ON` phrase.

The *search-condition* is built from expressions, including subqueries. For more information, see [“Dynamic SQL search conditions” on page 170](#).

**GROUP BY clause** You can group by columns, alias names, or functions.

---

The result of the query contains one row for each distinct set of values in the named columns, aliases, or functions. All NULL-containing rows are treated as a single set. The resulting rows are often referred to as groups since there is one row in the result for each group of rows from the table list. Aggregate functions can then be applied to these groups to get meaningful results.

A *group-by-expr* is a (non-aggregate) expression written exactly the same as one of the expressions in the *select-list*.

When GROUP BY is used, the *select-list* and ORDER BY expressions must not reference any identifier that is not named in the GROUP BY clause. The exception is that the *select-list* may contain aggregate functions.

**HAVING clause** This clause selects rows based on the group values and not on the individual row values. The HAVING clause can only be used if either the statement has a GROUP BY clause or the select list consists solely of aggregate functions. Any column names referenced in the HAVING clause must either be in the GROUP BY clause or be used as a parameter to an aggregate function in the HAVING clause.

**ORDER BY clause** This clause sorts the results of a query. Each item in the ORDER BY list can be labeled as ASC for ascending order (the default) or DESC for descending order. If the expression is an integer *n*, then the query results will be sorted by the *n*th item in the select list.

The only way to ensure that rows are returned in a particular order is to use ORDER BY. In the absence of an ORDER BY clause, UltraLite returns rows in whatever order is most efficient. This means that the appearance of result sets may vary depending on when you last accessed the row and other factors.

**OPTION ( FORCE ORDER ) clause** This clause is not recommended for general use. It overrides UltraLite's choice of the order in which to access tables, and require that UltraLite access the tables in the order they appear in the query. In general, it is best to let UltraLite decide on the table access order.

☞ For more information, see [“Query optimization” on page 185](#).

#### Usage

The SELECT statement is used for retrieving results from the database.

UltraLite tables do not have owners. The optional *owner* is supported as a convenience for existing SQL and for programmatically-generated SQL. UltraLite accepts but ignores *owner*.

#### See also

[“SELECT statement” \[ASA SQL Reference, page 597\]](#)

#### Example

How many employees are there?

```
SELECT count(*)
FROM employee
```

## UPDATE statement

Description	Use this statement to modify existing rows in database tables.
Syntax	<b>UPDATE</b> [ <i>owner.</i> ] <i>table-name</i> <b>SET</b> <i>column-name</i> = <i>expression</i> [ <b>WHERE</b> <i>search-condition</i> ]
Parameters	<p><b>table-name</b> The <i>table-name</i> specifies the name of the table to be updated. Only a single table is allowed.</p> <p><b>SET clause</b> Each named column is set to the value of the expression on the right hand side of the equal sign. There are no restrictions on the expression. If the expression is a <i>column-name</i>, the old value is used.</p> <p>Only columns specified in the SET clause have their values changed. In particular, you cannot use UPDATE to set a column's value to its default.</p> <p><b>WHERE clause</b> If a WHERE clause is specified, only rows satisfying <i>search-condition</i> are updated. For information about search conditions, see <a href="#">“Dynamic SQL search conditions” on page 170</a>.</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p><b>Caution</b> <i>If no WHERE clause is specified, every row in the table is updated.</i></p> </div> <p><b>Case sensitivity</b> Character strings inserted into tables are always stored in the same case as they are entered, regardless of whether the database is case sensitive or not. A CHAR data type column updated with a string <b>Value</b> is always held in the database with an upper case V and the remainder of the letters lower case. SELECT statements return the string as <b>Value</b>. If the database is not case sensitive, however, all comparisons make <b>Value</b> the same as <b>value</b>, <b>VALUE</b>, and so on. Further, if a single-column primary key already contains an entry <b>Value</b>, an INSERT of <b>value</b> is rejected, as it would make the primary key not unique.</p>
Usage	<p>The UPDATE statement modifies values in a table.</p> <p>UltraLite tables do not have owners. The optional <i>owner</i> is supported as a convenience for existing SQL and for programmatically-generated SQL. UltraLite accepts but ignores <i>owner</i>.</p>
Side effects	None.
See also	<ul style="list-style-type: none"> <li>◆ <a href="#">“INSERT statement” on page 179</a></li> <li>◆ <a href="#">“DELETE statement” on page 177</a></li> </ul>

---

Example

Transfer employee Philip Chin (employee 129) from the sales department to the marketing department.

```
UPDATE employee
SET dept_id = 400
WHERE emp_id = 129
```



## Query optimization

There are many different ways for UltraLite to execute any query. Each distinct way of executing a query is called a **plan**. In UltraLite a plan is defined mainly by the order in which tables are accessed and by whether each table is searched using an index or by scanning the rows directly. For some queries, there can be an orders of magnitude difference in execution times between efficient and inefficient plans.

UltraLite includes a **query optimizer**: an internal component of the UltraLite runtime that inspects alternative plans and attempts to select an efficient one. The primary goal of optimization in UltraLite is to choose indexes so that data can be accessed in an efficient order. The optimizer attempts to avoid the use of temporary tables to store intermediate results and attempts to ensure that only the pertinent subset of a table is accessed when a query joins two tables.

UltraLite optimizes queries automatically. The main area in which you can tune execution time is by creating indexes in your database that UltraLite can exploit as it optimizes queries.

### Inspecting query plans

As a development aid, you can use the UltraLite Interactive SQL to display the plan that summarizes how a prepared statement is to be executed. The plan is displayed on a tab in the bottom pane of the utility. You can choose whether to display a plan graphically or in a text format.

For example, the statement

```
SELECT I.inv_no, I.name, T.quantity, T.prod_no
FROM Invoice I, Transactions T
WHERE I.inv_no = T.inv_no
```

might produce the following plan:

```
join[scan(Invoice,0),index-scan(Transactions,1)]
```

The plan indicates that the join operation is accomplished by reading all rows from the Invoice table (following index[0]) and then using the index[1] from the Transaction table to read only the row whose inv\_no column matches.

☞ For more information about UltraLite Interactive SQL, see [“The UltraLite Interactive SQL utility” on page 115](#).

### Overriding the optimizer

To be usable on small devices, query optimization in UltraLite is not as extensive as that carried out in Adaptive Server Anywhere. You can override the table order it selects by adding the `OPTION (FORCE ORDER)` clause to a query, which forces UltraLite to access the tables in the order they appear in the query. *This option is not recommended for general use.* If performance

---

is slow, a better approach is usually to create appropriate indexes to speed up execution.

## PART III

# APPLICATION DEVELOPMENT

This part introduces the embedded SQL, Static C++ API, and Static Java programming interfaces.

When using a static interface, all queries must be specified at compile time.



---

## CHAPTER 9

# Developing Applications for the Palm OS

About this chapter

This chapter describes general issues when developing applications for the Palm OS.

Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Choosing database storage on the Palm OS</a>	190
<a href="#">Understanding the Palm Creator ID</a>	191

---

## Choosing database storage on the Palm OS

On the Palm OS, it is important to distinguish between UltraLite databases and the Palm data store, which is sometimes called the Palm database. In this documentation, the term **PDB** means a Palm database and **database** refers to an UltraLite relational database. In addition to the Palm data store, Palm OS version 4.0 and later also support a virtual file system (VFS) on expansion cards.

Palm data store or virtual file system      UltraLite databases can be stored either in the Palm data store on the virtual file system on expansion cards. The way of specifying the storage depends on the interface used:

- ◆ **UltraLite for MobileVB**      To use the virtual file system, set the VFS On Palm parameter. See “[VFS On Palm parameter](#)” on page 81.
- ◆ **UltraLite C++ component, static C++ API, embedded SQL**      Call `ULEnablePalmRecordDB` or `ULEnableFileDB` at the beginning of your application. See “[ULEnablePalmRecordDB function](#)” [*UltraLite C/C++ User’s Guide*, page 210] and “[ULEnableFileDB function](#)” [*UltraLite C/C++ User’s Guide*, page 208].

Storage details      When using the Palm data store, UltraLite actually stores database information in multiple PDBs, whose names are constructed using the given creator ID. For example, a database created with a creator ID of ABCD causes the following files to be created:

- ◆ `ul_state_ABCD`
- ◆ `ul_udb_ABCD`

UltraLite uses the **state PDB** (`ul_state_ABCD`) to hold the current row on which the application is positioned for any open tables when the application exits. The state PDB allows UltraLite allows you to write your application so that when it is launched, users can resume where they left off.

## Understanding the Palm Creator ID

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

☞ For information about assigning creator IDs to applications, see your development tool documentation. For information about registering creator IDs with HotSync manager, see “[Registering the MobiLink HotSync conduit to HotSync Manager](#)” [*MobiLink Clients*, page 299].

UltraLite uses the creator ID to manage databases and HotSync synchronization. In many scenarios, you do not need to know the details of how UltraLite uses creator IDs. However, if you create an application that connects to more than one UltraLite database and that uses HotSync for synchronization, you do need to know more about how UltraLite uses creator IDs. This section is provided for those users.

Palm OS uses a creator ID to associate related PDBs and applications. For example, the CustDB sample application has the creator ID Syb1. The MobiLink conduit uses this Syb1 creator ID to find the associated UltraLite database on the device.

The creator ID is a string from one to four characters long. The first character should be an upper case letter, as Palm OS uses an initial lower-case letter for its system files.

### UltraLite database names

UltraLite assigns each new database a creator ID and a PDB name based on the value of the creator ID supplied in the Database On Palm connection parameter or, if no Database On Palm connection parameter is supplied, the creator ID of your application:

- ◆ For applications that use the record-based Palm OS store, the PDB name is `ul_ldb_creator-id`.
- ◆ For applications that use the virtual file system, the name of the file is `ul_ldb_creator-id.ldb`.

☞ For more information, see “[Database On Palm connection parameter](#)” on page 71.

### HotSync and creator IDs

The use of creator IDs to identify both applications and their related databases can cause problems if your application connects to multiple databases.

During HotSync synchronization, HotSync Manager checks the creator ID of each application. It passes those creator IDs registered by the MobiLink

---

conduit to the conduit for synchronization. The conduit looks for a database named `ul_udb_creator-id`, where *creator-id* is the name supplied to it by HotSync manager.

If your application connects to two databases, at least one must have a different creator ID from the application. Because the database and application creator IDs do not match, HotSync Manager does not find an application associated with the database, which is therefore not included in the synchronization.

To work around this limitation, and to use HotSync for several UltraLite applications on the Palm OS, write a dummy Palm application for each creator ID. The dummy Palm application does not need to carry out any actions at all; it simply has to have a creator ID that HotSync can hand to the MobiLink conduit for synchronization, and which the MobiLink conduit can map to your UltraLite database creator ID. With this approach, the MobiLink HotSync conduit can identify multiple copies of UltraLite databases and synchronize them.



---

## CHAPTER 10

# Using UltraLite Static Interfaces

### About this chapter

This chapter presents an overview of the UltraLite static programming interfaces.

When using static interfaces, the SQL statements to be used in an application must be specified at compile time. In a dynamic model, SQL statements can be specified at run time. The static interfaces are embedded SQL, the Static C++ API, and the Static Java API which uses JDBC. This chapter describes aspects common to all static UltraLite interfaces.

### Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Overview</a>	194
<a href="#">Choosing an UltraLite static interface</a>	197
<a href="#">Preparing a reference database</a>	198
<a href="#">Defining SQL statements for your application</a>	202
<a href="#">Generating the UltraLite data access code</a>	207
<a href="#">Configuring development tools for static UltraLite development</a>	208

---

# Overview

This section describes the development environment and process for UltraLite static interfaces.

## The development environment for static UltraLite applications

Developing UltraLite applications using a static interface requires the following tools.

- ◆ **A reference database** A reference database is an Adaptive Server Anywhere database that serves as a model of the UltraLite database you want to create. You create this database yourself, using tools such as Sybase Central.

Your UltraLite database is a subset of the columns, tables, and indexes, in your reference database. The arrangement of tables and of the foreign key relationships between them is called the database **schema**.

In addition to modeling the UltraLite database, you need to add the SQL statements that are to be included in your UltraLite application to the reference database.

☞ For more information, see [“Preparing a reference database” on page 198](#).

- ◆ **A supported development tool** You use a standard development tool to develop UltraLite applications. For the non-UltraLite specific portions of your application, such as the user interface, use your development tool in the usual way. For the UltraLite-specific data-access portions, you also need to use the UltraLite development tools.

It can be convenient to separate the data access code from the user interface and internal logic of your application.

☞ For information about supported application development tools, see [“UltraLite development platforms” \[Introducing SQL Anywhere Studio, page 99\]](#).

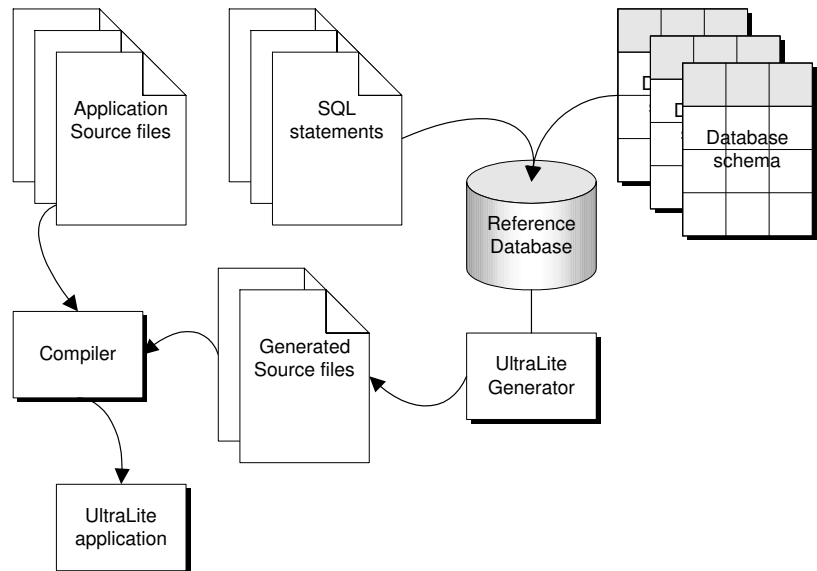
- ◆ **UltraLite development tools** UltraLite includes several tools for development using the static interfaces.
  - **The UltraLite generator** This application generates source code that implements the underlying query execution, data storage, and synchronization features of your application. The generator is required for all kinds of UltraLite development using static SQL.
  - **The SQL preprocessor** This application is needed only if you are developing an UltraLite application using embedded SQL. It reads your embedded SQL source files and generates standard C/C++ files.

As it scans the embedded SQL source files, it also stores information in the reference database that is used by the generator.

- ◆ **UltraLite runtime libraries** UltraLite includes a runtime library for each target platform. On some platforms, this is a static library that becomes part of your application executable; on other platforms it is a dynamic link library. For Java, the runtime library is a jar file. UltraLite includes all the header files and import files needed to use the runtime libraries.

## The static interface UltraLite development process

The basic features of the development process are common to all static interfaces. The following diagram summarizes the key features.



- ◆ Create a reference database, which contains a superset of the tables to be included in your application. It may also contain representative data for your application. This reference database is needed only as part of the development process, and is not required by your final application.
- ◆ Add the SQL statements into a special table in the reference database. The way this is accomplished is dependent on the interface you choose:
  - If you are using the Static C++ API or Java, these statements are added to your database using Sybase Central or a stored procedure.
  - If you are using embedded SQL, the SQL preprocessor adds the statements to the reference database for you.

- 
- ◆ Run the UltraLite generator, which produces source files that include code needed to execute your SQL statements, and code needed to define the database schema for your UltraLite application. This generated code includes function calls into the UltraLite runtime library.
  - ◆ Create application source files. If you are using embedded SQL, the SQL preprocessor reads your `.sql` files and inserts the SQL statements into the reference database for you.
  - ◆ Compile your application source files together with the generated source files to produce your UltraLite application.

## Adding synchronization

Most UltraLite applications include synchronization to integrate their data with data on a consolidated database.

☞ For more information about synchronization, and the kinds of synchronization available, see “UltraLite Clients” [*MobiLink Clients*, page 277].

## Choosing an UltraLite static interface

There are three static interfaces for developing UltraLite applications:

- ◆ **Static C++ API** Development using C or C++ with data access features using a result-set based API.
- ◆ **Embedded SQL** Development using C or C++ with data access features using embedded SQL statements.
- ◆ **Static Java API** Development using the Java programming language.

The decision whether to use Java or C/C++ development will be determined primarily by your target platform.

Here are some considerations when choosing between embedded SQL and the Static C++ API:

- ◆ Embedded SQL is an industry standard programming method, while the Static C++ API is a proprietary API.
- ◆ Embedded SQL gives more control in designing your application. If you are experienced with embedded SQL development, you can design a more efficient application using this method.
- ◆ Many programmers are more familiar with API-based programming. The Static C++ API requires less learning for these developers.
- ◆ The Static C++ API generates classes and associated methods for manipulating the database. It enforces standardized function names and so can be a quicker approach in terms of development time.

---

## Preparing a reference database

To implement the UltraLite database engine for your application, the UltraLite generator must have access to an Adaptive Server Anywhere reference database. This database must contain the following information:

- ◆ **Database schema** The database objects used in your UltraLite application, including tables and any indexes on those tables you wish to use in your application.

☞ For more information, see [“Using an existing database as a reference database” on page 200](#).

- ◆ **Data** (Optional) You can fill your reference database with data that is similar in quantity and distribution to the data you expect your UltraLite database to hold. The UltraLite analyzer automatically uses this information to optimize the performance of your application.

☞ For more information, see [“Using an existing database as a reference database” on page 200](#).

- ◆ **Queries** The UltraLite system tables must contain any SQL statements you wish to use in your application.

☞ For more information, see [“Defining SQL statements for your application” on page 202](#).

- ◆ **Publications** If you wish to add multiple synchronization options to your application, you can do so using publications. You also add publications to your database if you wish to develop a C++ API application without defining queries.

☞ For information on multiple synchronization options, see [“Designing sets of data to synchronize separately” \[MobiLink Clients, page 280\]](#).

- ◆ **Database options** Database options such as date formats and govern some aspects of database behavior that can make applications behave differently. The UltraLite database is generated with the same option settings as those in the reference database.

For many purposes, you can leave all database options at their default settings.

☞ For more information, see [“Setting database options in the reference database” on page 199](#).

## Creating a reference database

The analyzer uses the reference database as a template when constructing your UltraLite application.

### ❖ To create a reference database

1. Start with an existing Adaptive Server Anywhere database or create a new database using the `dbinit` command.
  - ☞ For more information on upgrading a database, see [“Using an existing database as a reference database” on page 200](#).
2. Add the tables and foreign key relationships that you need within your application. You can use any convenient tool, such as Sybase Central or Sybase PowerDesigner Physical Architect (included with SQL Anywhere Studio), or a more powerful database design tool such as the complete Sybase PowerDesigner package.

#### Performance tip

You do not need to include any data in your reference database. However, if you populate your database tables with data representative of the data you expect to be stored by a typical user of your application, the UltraLite analyzer automatically uses this data to optimize the performance of your application.

- ☞ For information about designing a database and creating a schema, see [“Designing Your Database” \[ASA SQL User’s Guide, page 3\]](#).

#### Example

1. Create a database.

From a command prompt, execute the following statement:

```
dbinit path\dbname.db
```

2. Use Sybase Central to add tables for your UltraLite application, based on your own needs.
3. Add your sample data. Interactive SQL includes an Import menu item that allows several common file formats to be imported.

- ☞ For more information, see [“Importing and Exporting Data” \[ASA SQL User’s Guide, page 555\]](#).

## Setting database options in the reference database

UltraLite does not support the getting or setting of option values.

When the UltraLite application is generated, certain option values in the reference database affect the behavior of the generated code. The following options have an effect:

- ◆ `Date_format`
- ◆ `Date_order`

- 
- ◆ Nearest\_century
  - ◆ Precision
  - ◆ Scale
  - ◆ Time\_format
  - ◆ Timestamp\_format

By setting these options in the reference database, you can control the behavior of your UltraLite database. The option setting in your reference database is used when generating your UltraLite application.

## Using an existing database as a reference database

Many UltraLite applications synchronize data via MobiLink with a central, master store of data called the **consolidated database**. Do not confuse a reference database with a consolidated database. The reference database for the UltraLite application is generally a different database from the consolidated database.

Only an Adaptive Server Anywhere consolidated database can also be used as a reference database. If your consolidated database is of another type, you must create an Adaptive Server Anywhere reference database. Even if your consolidated database is Adaptive Server Anywhere, you must create a separate reference database if you wish to have a different schema or use different settings in your UltraLite application.

You can choose any of the supported ODBC-compliant database management products to create and manage the consolidated database, including Adaptive Server Enterprise, Adaptive Server Anywhere, Oracle, Microsoft SQL Server, and IBM DB2.

If you have an existing Adaptive Server Anywhere database that you will be using as a consolidated database, you could make a copy of it for your reference database.

### ❖ **To create a reference database from a non-Adaptive Server Anywhere database**

1. Create a new Adaptive Server Anywhere database.

You can use the *dbinit* command or use Sybase Central.

2. Add the tables and foreign-key relationships that you need within your application using your consolidated database as a guide.

You can use a tool such as Sybase Physical Data Architect to re-engineer the consolidated database.



3. Populate your database tables with representative data from your consolidated database.

You need not transfer all the information in your consolidated database, only a representative sample. In the early stages of development, you do not need sample data at all. For production applications, you may want to use representative data because access plans of UltraLite queries are based on the distribution of data in the reference database.

☞ For more information on creating reference databases from non-Adaptive Server Anywhere databases, see “[Migrating databases to Adaptive Server Anywhere](#)” [*ASA SQL User’s Guide*, page 591].

## Optimizing query execution

You can improve the performance of your static UltraLite applications using the following techniques.

- ◆ **add an index** If you frequently retrieve information in a particular order, consider adding an index to your reference database. Primary keys are automatically indexed, but other columns are not. Particularly on slow devices, an index can improve performance dramatically.
- ◆ **add representative data** The Adaptive Server Anywhere optimizer automatically optimizes the performance of your queries. It chooses access plans using the information present in your reference database. To improve application performance, fill your reference database with data that is representative in size and distribution of the data you expect your application will hold once it is deployed.

---

## Defining SQL statements for your application

All the data access instructions for your application are defined by adding SQL statements to the reference database.

If you use the Static C++ API, you can also use publications to define data access methods. For information on using publications, see “[Defining UltraLite tables](#)” [*UltraLite C/C++ User's Guide*, page 43].

If you are using embedded SQL, the SQL preprocessor carries out the tasks in this section for you.

### Creating an UltraLite project

When you add SQL statements to a reference database, you assign them to an UltraLite **project**. By grouping them this way, you can develop multiple applications using the same reference database.

When the UltraLite generator runs against a reference database to generate the database source code files, it takes a project name as an argument and generates the code for the SQL statements in that project.

You can define an UltraLite project using Sybase Central or by directly calling a system stored procedure.

If you are using embedded SQL, the SQL preprocessor defines the UltraLite project for you and you do not need to create it explicitly.

#### ❖ To create an UltraLite project (Sybase Central)

1. In Sybase Central, connect to your database if you are not already connected.
2. In the left pane, open the database container.
3. In the left pane, open the UltraLite Projects folder.
4. From the File menu, choose New ► UltraLite Project.  
The UltraLite Project Creation wizard appears.
5. Enter an UltraLite project name and click Finish to create the project in the database.

 For information on UltraLite project naming rules, see “[ul\\_add\\_project system procedure](#)” on page 210.

❖ **To create an UltraLite project (SQL)**

1. From Interactive SQL or another application, enter the following command:

```
call ul_add_project( 'project-name' )
```

where *project-name* is the name of the project.

☞ For more information, see “[ul\\_add\\_project system procedure](#)” on [page 210](#).

❖ **To create an UltraLite project (embedded SQL)**

1. If you are using the embedded SQL interface, specify the UltraLite project name on the SQL Preprocessor command line, and the preprocessor adds the project to the database for you.

☞ For more information, see “[Building embedded SQL applications](#)” [[UltraLite C/C++ User's Guide, page 97](#)].

**Notes**

UltraLite project names must conform to the rules for database identifiers. If you include spaces in the project name, do not enclose the name in double quotes, as these are added for you by Sybase Central or the stored procedure.

☞ For more information, see “[Identifiers](#)” [[ASA SQL Reference, page 7](#)].

**Adding SQL statements to an UltraLite project**

Each UltraLite application carries out a set of data access requests. These requests are implemented differently in each interface, but the data access requests are defined in the same way for each model.

You define the data access requests that an UltraLite application can carry out by adding a set of SQL statements to the UltraLite project for that application in your reference database. The UltraLite generator then creates the code for a database engine that can execute the set of SQL statements.

In the Static C++ API, you can also use publications to define data access methods. For information on using publications, see “[Defining UltraLite tables](#)” [[UltraLite C/C++ User's Guide, page 43](#)].

You can add SQL statements to an UltraLite project using Sybase Central, or by directly calling a system stored procedure. If you are using embedded SQL, the SQL preprocessor adds the SQL statements in your embedded SQL source files to the reference database for you.

---

### ❖ To add a SQL statement to an UltraLite project (Sybase Central)

1. In Sybase Central, connect to your database if you are not already connected.
2. In the left pane, open the database container.
3. In the left pane, open the UltraLite Projects folder.
4. Open the project for your application.
5. From the File menu, choose New ► UltraLite statement.  
The UltraLite Statement Creation wizard appears.
6. Enter a short, descriptive name for the statement, and click Next
7. Enter the statement itself, and click Finish to add the statement to the project.

You can test the SQL statements against the database by right-clicking the statement and choosing Execute From Interactive SQL from the popup menu.

☞ For information on what kinds of statement you can use, see [“Writing UltraLite SQL statements” on page 205](#).

### ❖ To add a SQL statement to an UltraLite project (SQL)

1. From Interactive SQL or another application, enter the following command:

```
call ul_add_statement( 'project-name',  
                      'statement-name',  
                      'sql-statement' )
```

where *project-name* is the name of the project, *statement-name* is a short descriptive name, and *sql-statement* is the actual SQL statement.

☞ For more information, see [“ul\\_add\\_statement system procedure” on page 210](#).

### ❖ To add a SQL statement to an UltraLite project (embedded SQL)

1. If you are using the embedded SQL interface, specify the UltraLite project name on the SQL Preprocessor command line.

No statement name is used in embedded SQL development.

☞ For more information, see [“Building embedded SQL applications” \[UltraLite C/C++ User’s Guide, page 97\]](#).

## Notes

Statement names should be short and descriptive. They are used by the UltraLite generator to identify the statement for use in Java or in the C++ API. For example, a statement named **ProductQuery** generates a C++ API class named **ProductQuery** and a Java constant named **PRODUCT\_QUERY**. Names should be valid SQL identifiers.

The SQL statement syntax is checked when you add the statement to the database, and syntax errors give an error message to help you identify mistakes.

You can use Sybase Central or `ul_add_statement` to update a statement in a project, in just the same way as you add a statement. If a statement already exists, it is overwritten with the new syntax. You must regenerate the UltraLite code whenever you modify a statement.

## Writing UltraLite SQL statements

This section describes what SQL statements you can add to an UltraLite project, and describes how to use placeholders in your SQL statements.

☞ For information on the range of SQL that you can use, see [“Overview of SQL support in UltraLite” on page 142](#).

### How to supply double quotes

The SQL statement that you enter, whether into Sybase Central or as an argument to `ul_add_statement`, is added to the reference database as a string. It must therefore conform to the rules for SQL strings.

You must escape some characters in your SQL statements using the backslash character.

☞ For information on SQL strings, see [“Strings” \[ASA SQL Reference, page 9\]](#).

### Using variables with statements

For most insert or update statements, you do not know the new values ahead of time. You can use question marks as placeholders for variables, and supply values at run time:

```
call ul_add_statement(
    'ProductApp',
    'AddCap',
    'INSERT INTO \"DBA\".product ( id, name, price )
     VALUES( ?, ?, ? )'
)
```

Placeholders can also be used in the WHERE clause of queries:

---

```
call ul_add_statement(  
    'ProductApp',  
    'ProductQuery',  
    'SELECT id, name, price  
    FROM \"DBA\".product  
    WHERE price > ?'  
)
```

The backslash characters are used to escape the double quotes.

In embedded SQL, you use **host variables** as placeholders. For more information, see [“Using host variables” \[UltraLite C/C++ User’s Guide, page 68\]](#).

For SQL statements containing placeholders, an extra parameter on the **Open** or **Execute** method of the generated C++ class is defined for each parameter. For Java applications, you use the JDBC set methods to assign values for the parameters.

## Generating the UltraLite data access code

To generate the code for storing and accessing the UltraLite database, the **UltraLite generator** analyzes your reference database and the SQL statements you use in your application. The UltraLite generator is a command-line application. It takes a set of command-line options to customize the behavior for each project. For example, it can generate either C/C++ or Java code, depending on the command-line options you supply.

The data storage code includes only those tables and columns of the reference database that you use in your application. Additionally, the UltraLite generator includes indexes present in your reference database whenever they improve the efficiency of your application.

The data access code includes only those SQL statements that you have added to the project in the reference database.

The result is a custom database engine tailored to your application. The engine is much smaller than a general-purpose database engine because the UltraLite generator includes only the features your application uses.

☞ For more information about the UltraLite generator, see [“The UltraLite Generator” on page 89](#).

---

## Configuring development tools for static UltraLite development

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

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

☞ For specific instructions on adding UltraLite projects to a dependency-based development environment, see [“Configuring development tools for embedded SQL development”](#) [*UltraLite C/C++ User’s Guide*, page 102].



---

## CHAPTER 11

# UltraLite Static Interfaces Reference

About this chapter

This chapter provides reference information about for the Ultralite static interfaces: embedded SQL, the Static C++ API, and the Static Java API.

Contents

<b>Topic:</b>	<b>page</b>
<a href="#">Reference database stored procedures</a>	210

---

## Reference database stored procedures

This section describes system stored procedures in the Adaptive Server Anywhere reference database, which can be used to add SQL statements to a project.

For each SQL statement added in this way, the UltraLite generator defines a C++ or Java class.

These system procedures are owned by the built-in user ID **dbo**.

### ul\_add\_statement system procedure

Function	Adds a SQL statement to an UltraLite project.
Syntax	<b>ul_add_statement</b> (in <b>@project</b> char(128), in <b>@name</b> char(128), in <b>@statement</b> text )
Permissions	DBA authority required
Side effects	None
See also	<a href="#">“ul_add_project system procedure” on page 210</a> <a href="#">“ul_delete_statement system procedure” on page 211</a>
Description	<p>Adds or modifies a statement to an UltraLite project.</p> <p><b>project</b> The UltraLite project to which the statement should be added. The UltraLite generator defines classes for all statements in a project at one time.</p> <p><b>name</b> The name of the statement. This name is used in the generated classes.</p> <p><b>statement</b> A string containing the SQL statement.</p> <p>If a statement of the same name in the same project exists, it is updated with the new syntax. If <i>project</i> does not exist, it is created.</p>
Examples	<p>The following call adds a statement to the TestSQL project:</p> <pre>call ul_add_statement(   'TestSQL', 'TestQuery',   'select prod_id, price, prod_name from ulproduct where price &lt;     ?' )</pre>

### ul\_add\_project system procedure

Function	Creates an UltraLite project.
----------	-------------------------------

Syntax	<b>ul_add_project</b> (in <b>@project</b> char(128) )
Permissions	DBA authority required
Side effects	None
See also	<a href="#">“ul_delete_statement system procedure” on page 211</a>
Description	Adds an UltraLite project to the database. The project acts as a container for the SQL statements in an application, and the project name is supplied on the UltraLite generator command line so that it can define classes for all statements in the project.  <b>project</b> The UltraLite project name.
Examples	The following call adds a project named <b>Product</b> to the database:  <pre>call ul_add_project( 'Product' )</pre>

## ul\_delete\_project system procedure

Function	Removes an UltraLite project from a database.
Syntax	<b>ul_delete_project</b> (in <b>@project</b> char(128) )
Permissions	DBA authority required
Side effects	None
See also	<a href="#">“ul_add_project system procedure” on page 210</a> <a href="#">“ul_delete_statement system procedure” on page 211</a>
Description	Removes an UltraLite project from the database.  <b>project</b> The UltraLite project to be deleted from the database.
Examples	The following call deletes the <b>Product</b> project:  <pre>call ul_delete_project( 'Product' )</pre>

## ul\_delete\_statement system procedure

Function	Removes a SQL statement from an UltraLite project.
Syntax	<b>ul_delete_statement</b> (in <b>@project</b> char(128), in <b>@name</b> char(128) )
Permissions	DBA authority required
Side effects	None

---

See also	<a href="#">“ul_add_project system procedure” on page 210</a> <a href="#">“ul_add_statement system procedure” on page 210</a>
Description	<p>Removes a statement from an UltraLite project.</p> <p><b>project</b> The UltraLite project from which the statement should be removed.</p> <p><b>name</b> The name of the statement. This name is used in the generated classes.</p>
Examples	<p>The following call removes a statement from the <b>Product</b> project:</p>

```
call ul_delete_statement( 'Product', 'AddProd' )
```

## ul\_set\_codeselement system procedure

Function	For Palm Computing Platform development using the C++ API, assigns a SQL statement from an UltraLite project to a particular segment.
Syntax	<b>ul_set_codeselement</b> (in <b>@project</b> char(128), in <b>@name</b> char(128), in <b>@segment_name</b> char(8) )
Side effects	None
See also	<a href="#">“ul_add_statement system procedure” on page 210</a> <a href="#">“Explicitly assigning segments” [UltraLite C/C++ User’s Guide, page 123]</a>
Description	<p>Explicitly assigns the generated code for a C++ API SQL statement to a named Palm segment.</p> <p><b>project</b> The UltraLite project to which the statement applies.</p> <p><b>name</b> The name of the statement as defined in <a href="#">“ul_add_statement system procedure” on page 210</a>..</p> <p><b>segment_name</b> The name of the segment to which the statement is assigned.</p>
Examples	<p>The following call assigns the statement <b>mystmt</b> in project <b>myproject</b> to segment <b>MYSEG1</b>.</p>

```
call ul_set_codeselement(
    'myproject', 'mystmt', 'MYSEG1' )
```

---

# Index

## Symbols

% operator	
modulo function	148
&	
bitwise operator for UltraLite	167
^	
bitwise operator for UltraLite	167
~	
bitwise operator for UltraLite	167
bitwise operator for UltraLite	167

## A

ABS function	
UltraLite SQL syntax	148
ACOS function	
UltraLite SQL syntax	148
AdditionalParams connection parameter	
UltraLite	68
AdditionalParams property	
connection strings	66
AES encryption algorithm	
UltraLite databases	36
aggregate expressions	
UltraLite	163
aliases	
columns in UltraLite	181
DELETE statement for UltraLite dynamic SQL	178
ALL conditions	
UltraLite dynamic SQL	170
altering	
UltraLite databases	54
AND	
bitwise operators for UltraLite	167
logical operators for UltraLite	168
ANSI character sets	
UltraLite databases	45
ANY conditions	
UltraLite dynamic SQL	170
applications	

writing static UltraLite applications	194
architecture	
UltraLite	7
ARGN function	
UltraLite SQL syntax	148
arithmetic	
operators and UltraLite dynamic SQL syntax	167
ASCII	
function and UltraLite SQL syntax	148
ASIN function	
UltraLite SQL syntax	148
ATAN function	
UltraLite SQL syntax	148
ATAN2 function	
UltraLite SQL syntax	148
ATN2 function	
UltraLite SQL syntax	148
autocommit	
UltraLite	48
AUTOINCREMENT	
about (UltraLite)	174
AVG function	
UltraLite SQL syntax	148

## B

backups	
UltraLite databases	48
UltraLite databases on Palm	123
BETWEEN conditions	
UltraLite dynamic SQL	170
BIGINT data type	
UltraLite	145
BINARY data type	
UltraLite	145
bitwise operators	
UltraLite dynamic SQL syntax	167
BYTE_LENGTH function	
UltraLite SQL syntax	148
BYTE_SUBSTR function	
UltraLite SQL syntax	148

<b>C</b>		
cache_size connection parameter		
UltraLite	73	
CacheSize connection parameter		
UltraLite	73	
cascading deletes		
not supported in UltraLite	51	
cascading updates		
not supported in UltraLite	51	
CASE expression		
NULLIF function for UltraLite	148	
UltraLite dynamic SQL syntax	164	
case sensitivity		
comparison operators for UltraLite	166	
UltraLite databases	30	
UltraLite strings	143	
CAST function		
UltraLite SQL syntax	148	
ce_file connection parameter		
about UltraLite	69	
ce_schema connection parameter		
UltraLite	78	
CEILING function		
UltraLite SQL syntax	148	
Certicom		
security for UltraLite	92	
changeEncryptionKey method		
UltraLite Static C++	37	
CHAR data type		
UltraLite	145	
CHAR function		
UltraLite SQL syntax	148	
CHAR_LENGTH function		
UltraLite SQL syntax	148	
character sets		
synchronization for UltraLite	46	
synchronization in UltraLite	43	
UltraLite	43	
UltraLite databases	30, 43	
UltraLite Java	45	
UltraLite on Palm Computing Platform	44	
UltraLite on Windows	45	
UltraLite on Windows CE	44	
UltraLite strings	143	
character strings		
UltraLite embedded SQL	97	
CHARINDEX function		
UltraLite SQL syntax	148	
check constraints		
UltraLite limitations	51	
choosing		
UltraLite programming interface	10	
COALESCE function		
UltraLite SQL syntax	148	
code generation		
UltraLite	207	
code pages		
synchronization in UltraLite	43	
collation sequences		
UltraLite databases	43	
columns		
aliases in UltraLite	181	
comma-separated lists		
LIST function UltraLite syntax	148	
COMMIT statement		
UltraLite dynamic SQL syntax	172	
commits		
UltraLite databases	47	
committing		
transactions in UltraLite	48, 172	
comparison operators		
dynamic SQL syntax for UltraLite	166	
UltraLite dynamic SQL	166	
compatibility		
UltraLite databases	45	
UltraLite dynamic SQL	166	
components		
choosing for UltraLite	11	
UltraLite development	11	
compression		
UltraLite databases	47	
computed columns		
UltraLite limitations	51	
con connection parameter		
UltraLite	74	
concatenating strings		
string operators for UltraLite	167	
concurrency		
synchronizing UltraLite applications	60	
UltraLite databases	58	
concurrent access		
UltraLite engine	61	
conditions		
ALL conditions for UltraLite dynamic SQL	170	

ANY for UltraLite dynamic SQL	170	connection strings	
BETWEEN for UltraLite dynamic SQL	170	about UltraLite	66
EXISTS for UltraLite dynamic SQL	171	ConnectionName connection parameter	
IN for UltraLite dynamic SQL	171	UltraLite	74
conduit		connections	
installing	99	concurrency in UltraLite	58
installing for CustDB	99	UltraLite limitations	50
configuring		consolidated databases	
development tools for UltraLite	208	UltraLite sample	25
connecting		conventions	
UltraLite database troubleshooting	67	documentation	x
UltraLite databases	40	CONVERT function	
connection parameters		UltraLite SQL syntax	148
about UltraLite	66	converting	
AdditionalParms for UltraLite	68	UltraLite databases	101
cache_size for UltraLite	73	COS function	
CacheSize for UltraLite	73	UltraLite SQL syntax	148
ce_file for UltraLite	69	COT function	
ce_schema for UltraLite	78	UltraLite SQL syntax	148
ConnectionName for UltraLite	74	COUNT function	
DatabaseName for UltraLite	82	UltraLite SQL syntax	148
DatabaseOnCE for UltraLite	69	CREATE TABLE statement	
DatabaseOnDesktop for UltraLite	70	SQL syntax (UltraLite)	173
dbn for UltraLite	82	creating	
EncryptionKey for UltraLite	75	reference databases for UltraLite	198
key for UltraLite	75	tables (UltraLite)	173
obfuscate for UltraLite	83	UltraLite databases	30, 101, 108
page_size for UltraLite	83	UltraLite reference databases	198
PageSize for UltraLite	83	UltraLite schema files	29
palm_db for UltraLite	71	creator IDs	
palm_fs for UltraLite	81	about	191
palm_schema for UltraLite	80	Database On Palm connection parameter	71
password for UltraLite	76	HotSync synchronization	191
precedence for UltraLite	67	Palm OS applications	191
reserve_size for UltraLite	84	current row	
schema_file for UltraLite	79	concurrency in UltraLite	59
SchemaOnCE for UltraLite	78	CURRENT_TIMESTAMP	
SchemaOnDesktop for UltraLite	79	SQL special value for UltraLite	52
SchemaOnPalm for UltraLite	80	cursors	
specifying UltraLite	66	concurrency in UltraLite	59
UltraLite	63	CustDB application	
UltraLite ConnectionName	74	file locations in UltraLite	17
UltraLite DatabaseOnPalm	71	installing conduit	99
UltraLite file_name	70	location in UltraLite	17
UltraLite overview	64	source code in UltraLite	17
UserID for UltraLite	76	starting in UltraLite	20
VFSONPalm for UltraLite	81	synchronization in UltraLite	18

UltraLite	16	database identification parameters	
UltraLite tutorial	15	UltraLite	68
custdb.db		database options	
location in UltraLite	17	setting in UltraLite	33
		UltraLite	33
		UltraLite DateFormat	34
		UltraLite DateOrder	34
		UltraLite NearestCentury	34
		UltraLite Precision	36
		UltraLite reference databases	199
		UltraLite Scale	36
		UltraLite TimeFormat	34
		UltraLite TimestampFormat	35
		UltraLite TimestampIncrement	35
<b>D</b>		database properties	
data		UltraLite	33
selecting rows in UltraLite	180	database schemas	
data access		UltraLite	30
UltraLite	11	DatabaseName connection parameter	
Data Manager		UltraLite	82
UltraLite database storage	31	DatabaseOnCE connection parameter	
data types		UltraLite	69
BIGINT in UltraLite	145	DatabaseOnDesktop connection parameter	
BINARY in UltraLite	145	UltraLite	70
CHAR in UltraLite	145	DatabaseOnPalm connection parameter	
DATE in UltraLite	145	UltraLite	71
DECIMAL in UltraLite	145	databases	
DOUBLE in UltraLite	145	collation sequences for UltraLite	43
FLOAT in UltraLite	145	creating UltraLite	30
INT in UltraLite	145	deleting UltraLite	123
INTEGER in UltraLite	145	introduction to UltraLite	6
LONG BINARY in UltraLite	145	Palm database	190
LONG VARCHAR in UltraLite	145	UltraLite database storage	31
NUMERIC in UltraLite	145	UltraLite introduction	28
REAL in UltraLite	145	UltraLite limitations	50
retrieving in UltraLite	148	UltraLite reference	198
SMALLINT in UltraLite	145	DATALENGTH function	
TIME in UltraLite	145	UltraLite SQL syntax	148
TIMESTAMP in UltraLite	145	DATE data type	
TINYINT in UltraLite	145	UltraLite	145
UltraLite	145	DATE function	
UltraLite SQL	142	UltraLite SQL syntax	148
VARBINARY in UltraLite	145	DATE_FORMAT option	
VARCHAR in UltraLite	145	UltraLite databases	199
database creation parameters		DATE_ORDER option	
UltraLite	82	UltraLite databases	199
database engine		DATEADD function	
UltraLite runtime	61		
database files			
changing the encryption key in UltraLite Static			
C++	37		
encrypting for UltraLite	37		
encrypting UltraLite	75		
UltraLite	47		
UltraLite connection parameters	65		
UltraLite on Palm OS	47		



UltraLite SQL syntax	148	deletes	
DATEDIFF function		UltraLite databases	47
UltraLite SQL syntax	148	deleting	
DateFormat database option		UltraLite utility to delete databases	123
UltraLite	34	derived tables	
DATEFORMAT function		UltraLite dynamic SQL	163
UltraLite SQL syntax	148	development	
DATENAME function		UltraLite static development process	195
UltraLite SQL syntax	148	development tools	
DateOrder database option		configuring for UltraLite	208
UltraLite	34	UltraLite preprocessing	208
DATEPART function		DIFFERENCE function	
UltraLite SQL syntax	148	UltraLite SQL syntax	148
dates		DISTINCT keyword	
formatting in UltraLite	34	UltraLite dynamic SQL	180
interpreting in UltraLite	34	documentation	
UltraLite databases	199	conventions	x
DATETIME function		SQL Anywhere Studio	viii
UltraLite SQL syntax	148	UltraLite	8
DAY function		DOUBLE data type	
UltraLite SQL syntax	148	UltraLite	145
DAYNAME function		double quotes	
UltraLite SQL syntax	148	static UltraLite SQL statements	205
DAYS function		DOW function	
UltraLite SQL syntax	148	UltraLite SQL syntax	148
dbcond9 utility		dynamic SQL	
syntax	99	arithmetic operators for UltraLite	167
DBF connection parameter		bitwise operators for UltraLite	167
UltraLite	70	comparison operators for UltraLite	166
dbn connection parameter		introduction for UltraLite	160
about UltraLite	82	logical operators for UltraLite	168
dbuleng9	<i>see also</i> UltraLite engine	operator precedence for UltraLite	169
syntax	88	string operators for UltraLite	167
dbulstop		UltraLite	161
UltraLite engine	61	UltraLite data access	11
dbulstop utility		UltraLite limitations	144
syntax	100	dynamic SQL UltraLite syntax	
DECIMAL data type		operators	166
UltraLite	145		
DEFAULT TIMESTAMP columns		<b>E</b>	
UltraLite	174	ELSE	
defaults		CASE expression for UltraLite	164
autoincrement (UltraLite)	174	IF expressions for UltraLite	164
DEGREES function		embedded SQL	
UltraLite SQL syntax	148	authorization for UltraLite	97
DELETE statement		character strings for UltraLite	97
dynamic SQL syntax (UltraLite)	177	line numbers for UltraLite	97

preprocessor for UltraLite	95	CustDB sample application in UltraLite	17
encryption		FIRST clause	
changing keys in UltraLite Static C++	37	UltraLite dynamic SQL SELECT statement	180
Palm Computing Platform	38	FLOAT data type	
storing the encryption key in UltraLite embedded SQL	38	UltraLite	145
UltraLite databases	36, 37	FLOOR function	
UltraLite encryption keys	75	UltraLite SQL syntax	148
encryption keys		FOR clause	
guidelines for UltraLite	37	SELECT statement for UltraLite dynamic SQL	182
EncryptionKey connection parameter		FORCE ORDER clause	
UltraLite	75	UltraLite dynamic SQL	182
END		foreign keys	
CASE expression for UltraLite	164	integrity constraints (UltraLite)	177
ENDIF		role names (UltraLite)	177
IF expressions for UltraLite	164	UltraLite	6
exclusive OR		unnamed (UltraLite)	177
bitwise operator for UltraLite	167	FROM clause	
EXISTS conditions		SELECT statement for UltraLite dynamic SQL	181
UltraLite dynamic SQL	171	functions	
EXP function		UltraLite SQL	143
UltraLite SQL syntax	148	functions, aggregate	
expressions		AVG for UltraLite	148
aggregate for UltraLite	163	COUNT for UltraLite	148
CASE expressions for UltraLite	164	LIST for UltraLite	148
data types of	148	MAX for UltraLite	148
IF expressions for UltraLite	164	MIN for UltraLite	148
SQL operator precedence for UltraLite	169	functions, data type conversion	
subqueries for UltraLite dynamic SQL	163	CAST for UltraLite	148
UltraLite SQL	143	CONVERT for UltraLite	148
EXPRTYPE function		HEXTOINT for UltraLite	148
UltraLite SQL syntax	148	INTTOHEX for UltraLite	148
<b>F</b>		ISDATE for UltraLite	148
features		ISNULL for UltraLite	148
UltraLite	4	functions, date and time	
UltraLite sample	16	DATE for UltraLite	148
feedback		DATEADD for UltraLite	148
documentation	xiv	DATEDIFF for UltraLite	148
providing	xiv	DATEFORMAT for UltraLite	148
fetching rows		DATENAME for UltraLite	148
concurrency in UltraLite	59	DATEPART for UltraLite	148
file_name connection parameter		DATETIME for UltraLite	148
UltraLite	70	DAY for UltraLite	148
filenames		DAYNAME for UltraLite	148
UltraLite connection parameters	65	DAYS for UltraLite	148
files		DOW for UltraLite	148

GETDATE for UltraLite	148	TRUNCATE for UltraLite	148
HOUR for UltraLite	148	TRUNCNUM for UltraLite	148
HOURS for UltraLite	148	functions, string	
MINUTE for UltraLite	148	ASCII for UltraLite	148
MINUTES for UltraLite	148	BYTE_LENGTH for UltraLite	148
MONTH for UltraLite	148	BYTE_SUBSTR for UltraLite	148
MONTHNAME for UltraLite	148	CHAR for UltraLite	148
MONTHS for UltraLite	148	CHAR_LENGTH for UltraLite	148
NOW for UltraLite	148	CHARINDEX for UltraLite	148
QUARTER for UltraLite	148	DIFFERENCE for UltraLite	148
SECOND for UltraLite	148	INSERTSTR for UltraLite	148
SECONDS for UltraLite	148	LCASE for UltraLite	148
TODAY for UltraLite	148	LEFT for UltraLite	148
WEEKS for UltraLite	148	LENGTH for UltraLite	148
YMD for UltraLite	148	LOCATE for UltraLite	148
functions, miscellaneous		LOWER for UltraLite	148
ARGN for UltraLite	148	LTRIM for UltraLite	148
COALESCE for UltraLite	148	PATINDEX for UltraLite	148
GREATER for UltraLite	148	REPEAT for UltraLite	148
IFNULL for UltraLite	148	REPLACE for UltraLite	148
LESSER for UltraLite	148	REPLICATE for UltraLite	148
NEWID for UltraLite	148	RIGHT for UltraLite	148
NULLIF for UltraLite	148	RTRIM for UltraLite	148
functions, numeric		SIMILAR for UltraLite	148
ABS for UltraLite	148	SOUNDEX for UltraLite	148
ACOS for UltraLite	148	SPACE for UltraLite	148
ASIN for UltraLite	148	STR for UltraLite	148
ATAN for UltraLite	148	STRING for UltraLite	148
ATAN2 for UltraLite	148	STRTOUUID for UltraLite	148
ATN2 for UltraLite	148	STUFF for UltraLite	148
CEILING for UltraLite	148	SUBSTRING for UltraLite	148
COS for UltraLite	148	TRIM for UltraLite	148
COT for UltraLite	148	UCASE for UltraLite	148
DEGREES for UltraLite	148	UPPER for UltraLite	148
FLOOR for UltraLite	148	UIDTOSTR for UltraLite	148
LOG for UltraLite	148	functions, system	
LOG10 for UltraLite	148	DATALENGTH for UltraLite	148
MOD for UltraLite	148		
PI for UltraLite	148	<b>G</b>	
POWER for UltraLite	148	generator	
RADIANS for UltraLite	148	database options for UltraLite	200
REMAINDER for UltraLite	148	GETDATE function	
ROUND for UltraLite	148	UltraLite SQL syntax	148
SIGN for UltraLite	148	global temporary tables	
SIN for UltraLite	148	creating (UltraLite)	173
SQRT for UltraLite	148	GLOBAL_DATABASE_ID option	
TAN for UltraLite	148	CREATE TABLE statement (UltraLite)	174

globally unique identifiers  
 UltraLite SQL syntax for NEWID function 148

GREATER function  
 UltraLite SQL syntax 148

GROUP BY clause  
 SELECT statement for UltraLite dynamic SQL  
 181

GUIDs  
 UltraLite SQL syntax for NEWID function 148  
 UltraLite SQL syntax for STRTOUUID function  
 148  
 UltraLite SQL syntax for UUIDTOSTR function  
 148

**H**

HAVING clause  
 SELECT statement for UltraLite dynamic SQL  
 182

HEXTOINT function  
 UltraLite SQL syntax 148

hooks  
 sqlpp customization for UltraLite 96  
 ulgen customization for UltraLite 90

host platforms  
 static interfaces UltraLite development 194

HotSync conduit  
 installing 99  
 installing for CustDB 99

HotSync synchronization  
 creator IDs 191

HOUR function  
 UltraLite SQL syntax 148

HOURS function  
 UltraLite SQL syntax 148

**I**

icons  
 used in manuals xii

identifiers  
 UltraLite SQL 142

IF expressions  
 UltraLite dynamic SQL syntax 164

IFNULL function  
 UltraLite SQL syntax 148

IN conditions  
 UltraLite dynamic SQL 171

indexes

automatically created in UltraLite 173

foreign keys in UltraLite 173

primary keys in UltraLite 173

UltraLite 6

UltraLite databases 48

UltraLite static interfaces 201

unique in UltraLite 172

inner references  
 subqueries for UltraLite dynamic SQL 163

INSERT statement  
 dynamic SQL syntax (UltraLite) 179

inserting  
 rows into UltraLite tables 179

INSERTSTR function  
 UltraLite SQL syntax 148

INT data type  
 UltraLite 145

INTEGER data type  
 UltraLite 145

integrity  
 constraints (UltraLite) 176

INTO clause  
 SELECT statement for UltraLite dynamic SQL  
 181

INTTOHEX function  
 UltraLite SQL syntax 148

IS  
 logical operators for UltraLite 168

ISDATE function  
 UltraLite SQL syntax 148

ISNULL function  
 UltraLite SQL syntax 148

isolation levels  
 UltraLite 59

**J**

Java  
 UltraLite character sets 45

**K**

key connection parameter  
 UltraLite 75

**L**

large files  
 UltraLite generator 94

LCASE function		columns per table for UltraLite	50
UltraLite SQL syntax	148	connections per UltraLite database	50
LEFT function		rows per table for UltraLite	50
UltraLite SQL syntax	148	tables per UltraLite database	50
LENGTH function		media failures	
UltraLite SQL syntax	148	UltraLite databases	48
LESSER function		memory usage	
UltraLite SQL syntax	148	UltraLite database storage	31
limitations		UltraLite indexes	48
UltraLite	50	UltraLite row states	47
UltraLite data types	145	MIN function	
UltraLite SQL	144	UltraLite SQL syntax	148
line length		MINUTE function	
sqlpp output for UltraLite	97	UltraLite SQL syntax	148
LIST function		MINUTES function	
UltraLite SQL syntax	148	UltraLite SQL syntax	148
lists		MobiLink conduit	
LIST function UltraLite syntax	148	installing	99
loading		MOD function	
UltraLite databases	101, 117	UltraLite SQL syntax	148
LOCATE function		monitoring	
UltraLite SQL syntax	148	UltraLite schema upgrades	55
locking		MONTH function	
concurrency in UltraLite	59	UltraLite SQL syntax	148
LOG function		MONTHNAME function	
UltraLite SQL syntax	148	UltraLite SQL syntax	148
LOG10 function		MONTHS function	
UltraLite SQL syntax	148	UltraLite SQL syntax	148
logical operators		multiple databases	
UltraLite dynamic SQL syntax	168	UltraLite	59
LONG BINARY data type		multi-process access	
UltraLite	145	UltraLite engine	61
LONG VARCHAR data type		multi-threaded applications	
UltraLite	145	UltraLite	60
LOWER function		UltraLite Static Java	60
UltraLite SQL syntax	148	UltraLite thread-safe	194
LTRIM function		<b>N</b>	
UltraLite SQL syntax	148	NEAREST_CENTURY option	
<b>M</b>		UltraLite databases	199
managing schemas		NearestCentury database option	
UltraLite Schema Painter	125	UltraLite	34
mathematical expressions		NEWID function	
arithmetic operators for UltraLite	167	UltraLite SQL syntax	148
MAX function		newsgroups	
UltraLite SQL syntax	148	technical support	xiv
maximum		NOT	

bitwise operator for UltraLite	167
logical operators for UltraLite	168
NOW function	
UltraLite SQL syntax	148
NULL	
ISNULL function	148
NULLIF function	
UltraLite	148
using with CASE expressions in UltraLite	165
NUMERIC data type	
UltraLite	145
<b>O</b>	
obfuscate connection parameter	
UltraLite	83
obfuscation	
UltraLite databases	36, 83
operator precedence	
UltraLite dynamic SQL syntax	169
operators	
arithmetic operators for UltraLite	167
bitwise operators for UltraLite	167
comparison operators for UltraLite	166
logical operators for UltraLite	168
precedence of operators for UltraLite	169
string operators for UltraLite	167
UltraLite dynamic SQL syntax	166
optimization	
UltraLite dynamic SQL	185
optimizer	<i>see also</i> query optimizer
options	
setting in UltraLite	33
UltraLite databases	33
UltraLite reference databases	199
OR	
bitwise operators for UltraLite	167
logical operators for UltraLite	168
ORDER BY clause	
UltraLite dynamic SQL	182
order of operations	
SQL operator precedence for UltraLite	169
outer references	
subqueries for UltraLite dynamic SQL	163
owners	
UltraLite tables	142

**P**

page size	
UltraLite databases	83
page_size connection parameter	
UltraLite	83
PageSize connection parameter	
UltraLite	83
Palm Computing Platform	
code pages in UltraLite	43
collation sequences for UltraLite	43
creator IDs	191
UltraLite character sets	44
UltraLite databases	190
Palm databases	
PDB	190
Palm OS	
creator IDs	191
UltraLite databases	190
palm_allow_backup parameter	
persistent storage for UltraLite	84
palm_db connection parameter	
UltraLite	71
palm_fs connection parameter	
UltraLite	81
palm_schema connection parameter	
UltraLite	80
Password connection parameter	
UltraLite	76
passwords	
PASSWORD UltraLite connection parameter	76
sharing MobiLink and UltraLite	41
UltraLite databases	40
paths	
UltraLite connection parameters	65
PATINDEX function	
UltraLite SQL syntax	148
pattern matching	
PATINDEX function for UltraLite	148
wildcards for UltraLite	148
performance	
database cache for UltraLite	73
UltraLite static interfaces	201
persistent memory	
UltraLite database storage	31
persistent storage	
file_name parameter for UltraLite	70
palm_allow_backup for UltraLite	84

physical limitations			
UltraLite	50		
PI function			
UltraLite SQL syntax	148		
POWER function			
UltraLite SQL syntax	148		
precedence			
SQL operator precedence for UltraLite	169		
precision			
arithmetic operations in UltraLite	36		
Precision database option			
UltraLite	36		
PRECISION option			
UltraLite databases	199		
predicates			
ALL for UltraLite dynamic SQL	170		
ANY for UltraLite dynamic SQL	170		
BETWEEN for UltraLite dynamic SQL	170		
comparison operators for UltraLite	166		
EXISTS for UltraLite dynamic SQL	171		
IN for UltraLite dynamic SQL	171		
preprocessor			
database options for UltraLite	200		
primary keys			
generating unique values in UltraLite	148		
generating unique values using UUIDs in UltraLite	148		
integrity constraints (UltraLite)	176		
order of columns (UltraLite)	176		
UltraLite	6		
UltraLite requirements	48		
UUIDs and GUIDs for UltraLite	148		
procedures			
UltraLite limitations	51		
projects			
UltraLite	202, 203		
properties			
UltraLite databases	33		
PWD connection parameter			
UltraLite	76		
<b>Q</b>			
QUARTER function			
UltraLite SQL syntax	148		
query optimization			
UltraLite dynamic SQL	185		
UltraLite static interfaces	201		
query optimizer			
UltraLite	185		
quotation marks			
static UltraLite SQL statements	205		
<b>R</b>			
RADIANS function			
UltraLite SQL syntax	148		
reading			
rows in UltraLite	59		
REAL data type			
UltraLite	145		
recovery			
UltraLite databases	47, 48		
reference database			
performance for UltraLite	201		
reference databases			
creating for UltraLite	198		
creating from existing databases	200		
options for UltraLite	199		
referential integrity			
UltraLite databases	51		
REMAINDER function			
UltraLite SQL syntax	148		
remote databases			
deleting UltraLite data	123		
UltraLite	16		
remote servers			
creating tables (UltraLite)	173		
renaming schemas			
UltraLite Schema Painter	125		
REPEAT function			
UltraLite SQL syntax	148		
REPLACE function			
UltraLite SQL syntax	148		
REPLICATE function			
UltraLite SQL syntax	148		
requests			
concurrency in UltraLite	59		
reserve_size connection parameter			
UltraLite	84		
restoring			
UltraLite databases	48		
RIGHT function			
UltraLite SQL syntax	148		
role names			
about (UltraLite)	177		

ROLLBACK statement			
UltraLite dynamic SQL syntax	180		
rollbacks			
UltraLite databases	47		
rolling back			
transactions (UltraLite)	180		
transactions in UltraLite	48, 180		
ROUND function			
UltraLite SQL syntax	148		
rows			
inserting into UltraLite tables	179		
selecting in UltraLite	180		
updating for UltraLite dynamic SQL	183		
RTRIM function			
UltraLite SQL syntax	148		
runtime library			
UltraLite	58		
<b>S</b>			
sample application			
CustDB in UltraLite	15		
starting CustDB in UltraLite	20		
synchronization in UltraLite	18		
samples			
CustDB file locations in UltraLite	17		
CustDB in UltraLite	16		
scale			
arithmetic operations in UltraLite	36		
Scale database option			
UltraLite	36		
SCALE option			
UltraLite databases	199		
schema changes			
UltraLite databases	54, 56		
schema files			
creating for UltraLite	29		
introduction for UltraLite	30		
Schema Painter			
starting	124		
UltraLite	124		
schema parameters			
UltraLite	78		
schema upgrades			
monitoring in UltraLite	55		
UltraLite databases	54, 56		
schema_file connection parameter			
UltraLite	79		
SchemaOnCE connection parameter			
UltraLite	78		
SchemaOnDesktop connection parameter			
UltraLite	79		
SchemaOnPalm connection parameter			
UltraLite	80		
schemas			
UltraLite databases	30, 194		
search conditions			
ALL for UltraLite dynamic SQL	170		
ANY for UltraLite dynamic SQL	170		
BETWEEN for UltraLite dynamic SQL	170		
EXISTS for UltraLite dynamic SQL	171		
IN for UltraLite dynamic SQL	171		
SECOND function			
UltraLite SQL syntax	148		
SECONDS function			
UltraLite SQL syntax	148		
security			
Certicom for UltraLite	92		
changing the encryption key in UltraLite Static			
C++	37		
database encryption for UltraLite	37		
encryption on Palm	38		
UltraLite generator	92		
UltraLite user authentication	40		
segments			
assigning statements for Palm	212		
Palm Computing Platform	212		
SELECT statement			
UltraLite dynamic SQL syntax	180		
selecting			
rows in UltraLite	180		
SET OPTION statement			
UltraLite limitations	52		
SIGN function			
UltraLite SQL syntax	148		
SIMILAR function			
UltraLite SQL syntax	148		
SIN function			
UltraLite SQL syntax	148		
SMALLINT data type			
UltraLite	145		
SOUNDEX function			
UltraLite SQL syntax	148		
source control			
storing UltraLite database schema in	29		



ulxml command line utility	29	UltraLite dynamic SQL COMMIT syntax (UltraLite)	172
sp_hook_ulgen_begin		UltraLite dynamic SQL ROLLBACK syntax	180
sqlpp for UltraLite	96	UltraLite dynamic SQL SELECT syntax	180
ulgen hook	90	UPDATE syntax for UltraLite dynamic SQL	183
sp_hook_ulgen_end		static interfaces	
sqlpp for UltraLite	96	choosing for UltraLite	11
ulgen hook	90	UltraLite	194
SPACE function		UltraLite development	11
UltraLite SQL syntax	148	static SQL	
SQL		UltraLite data access	11
UltraLite data access	11	STOP SYNCHRONIZATION DELETE statement	
SQL Anywhere Studio		UltraLite SQL	144
documentation	viii	stored procedures	
SQL in UltraLite		UltraLite limitations	51
overview	142	STR function	
SQL preprocessor		UltraLite SQL syntax	148
syntax for UltraLite	95	STRING function	
UltraLite	95	UltraLite SQL syntax	148
SQL statements		string operators	
CREATE TABLE syntax (UltraLite)	173	dynamic SQL syntax for UltraLite	167
DELETE dynamic SQL syntax (UltraLite)	177	strings	
INSERT syntax (UltraLite)	179	replacing for UltraLite	148
UltraLite	205	static UltraLite SQL statements	205
UltraLite dynamic SQL COMMIT syntax (UltraLite)	172	UltraLite case sensitivity	143
UltraLite dynamic SQL ROLLBACK syntax	180	UltraLite SQL	142
UltraLite dynamic SQL SELECT syntax	180	strong encryption	
UltraLite SQL	144	UltraLite databases	36
UPDATE syntax for UltraLite dynamic SQL	183	STRTOUUID function	
SQL syntax		UltraLite SQL syntax	148
CASE expression for UltraLite	164	STUFF function	
IF expressions for UltraLite	164	UltraLite SQL syntax	148
sqlpp utility		subqueries	
syntax for UltraLite	95	UltraLite dynamic SQL	163
SQRT function		SUBSTR function	
UltraLite SQL syntax	148	UltraLite SQL syntax	148
START SYNCHRONIZATION DELETE statement		SUBSTRING function	
UltraLite SQL	144	UltraLite SQL syntax	148
state bytes		substrings	
UltraLite databases	47	replacing for UltraLite	148
state PDB		UltraLite SQL	148
about	190	support	
statements		newsgroups	xiv
CREATE TABLE syntax (UltraLite)	173	Sybase Central	
DELETE dynamic SQL syntax (UltraLite)	177	adding SQL statements to an UltraLite project	203
INSERT syntax (UltraLite)	179	browsing CustDB in UltraLite	25
		connecting to CustDB in UltraLite	25

creating UltraLite projects	202	UltraLite SQL syntax	148
synchronization		target platforms	
character sets in UltraLite	43	UltraLite development for static interfaces	194
concurrency in UltraLite	60	technical support	
CustDB application in UltraLite	18	newsgroups	xiv
ulsync utility for UltraLite databases	119	temporary files	
UltraLite character sets	46	UltraLite	47
synchronization logic		temporary tables	
browsing Sybase Central in UltraLite	25	creating (UltraLite)	173
synchronization scripts		UltraLite limitations	51
browsing the UltraLite sample	25	THEN	
syntax		IF expressions for UltraLite	164
arithmetic operators for UltraLite	167	threads	
bitwise operators for UltraLite	167	concurrency in UltraLite	58
CASE expression for UltraLite	164	UltraLite applications	60, 194
comparison operators for UltraLite	166	UltraLite Static Java applications	60
IF expressions for UltraLite	164	TIME data type	
logical operators for UltraLite	168	UltraLite	145
SQL operator precedence for UltraLite	169	TIME_FORMAT option	
string operators for UltraLite	167	UltraLite databases	199
UltraLite dynamic SQL operators	166	TimeFormat database option	
system failures		UltraLite	34
UltraLite databases	48	times	
system functions		formatting in UltraLite	34
UltraLite limitations	52	UltraLite databases	199
system procedures		TIMESTAMP	
ul_add_project	210	TIMESTAMP columns (UltraLite)	174
ul_add_statement	210	timestamp columns	
ul_delete_project	211	UltraLite limitations	52
ul_delete_statement	211	TIMESTAMP data type	
ul_set_codesegment	212	UltraLite	145
system tables		TIMESTAMP_FORMAT option	
UltraLite limitations	52	UltraLite databases	199
<b>T</b>		TimestampFormat database option	
table constraints		UltraLite	35
UltraLite	176	TimestampIncrement database option	
table-based API		UltraLite	35
UltraLite data access	11	timestamps	
tables		formatting in UltraLite	35
creating (UltraLite)	173	increments in UltraLite	35
inserting rows into for UltraLite	179	synchronizing	35
owner in UltraLite	142	TINYINT data type	
UltraLite	6	UltraLite	145
UltraLite limitations	50	TODAY function	
UltraLite requirements	48	UltraLite SQL syntax	148
TAN function		TOP clause	
		UltraLite dynamic SQL SELECT statement	180

transactions			
committing in UltraLite	172	ULClearEncryptionKey function	
concurrency in UltraLite	59	using	38
rolling back (UltraLite)	180	ulconv utility	
rolling back in UltraLite	180	creating UltraLite schema files	29
UltraLite databases	47	syntax	101
triggers		ulcreate utility	
UltraLite limitations	52	syntax	108
TRIM function		uldbsgen utility	
UltraLite SQL syntax	148	syntax	110
troubleshooting		ULEnableFileDB function	
connecting to UltraLite databases	67	creating UltraLite databases	190
UltraLite compilation problems	94	ULEnablePalmRecordDB function	
TRUNCATE function		creating UltraLite databases	190
UltraLite SQL syntax	148	ulgen utility	
TRUNCATE TABLE statement		syntax	89
UltraLite SQL	144	ulinit utility	
TRUNCNUM function		syntax	112
UltraLite SQL syntax	148	ulisql utility	<i>see</i> UltraLite Interactive SQL
tutorials		ullload utility	
UltraLite CustDB sample	15	syntax	117
UltraLite Interactive SQL	129	ULRetrieveEncryptionKey function	
UltraLite Schema Painter	129	using	38
<b>U</b>		ULSaveEncryptionKey function	
UCASE function		using	38
UltraLite SQL syntax	148	ulsync utility	
UID connection parameter		syntax	119
UltraLite	76	UltraLite	
ul_add_project system procedure		about	3
about	210	architecture	7
ul_add_statement system procedure		choosing a programming interface	10
about	210	code generation	207
ul_delete_project system procedure		database identification parameters	68
about	211	features	4
ul_delete_statement procedure		introduction	4
about	211	SQL support	142
ul_delete_statement system procedure		static interface development overview	195
about	211	upgrading	57
ul_set_codesegment procedure		utility programs	87
about	212	UltraLite components	
ul_set_codesegment system procedure		choosing	11
about	212	development process	13
UL_STORE_PARAMS macro		UltraLite connection parameters	
connection parameters	66	about	66
ULChangeEncryptionKey function		UltraLite consolidated database generation utility	
using in UltraLite Static C++	37	syntax	110
		UltraLite database converter	
		syntax	101

UltraLite database creation utility		UltraLite runtime	
syntax	108	about	58
UltraLite database files		UltraLite schema files	
about	47	XML format	126
page size	47	UltraLite Schema Painter	
Palm OS	47	about	124
UltraLite database loading utility		managing schemas	125
syntax	117	tutorial	129
UltraLite database synchronization utility		UltraLite SQL functions	
syntax	119	ABS function syntax	148
UltraLite database unloading utility		ACOS function syntax	148
syntax	121	ARGN function syntax	148
UltraLite databases		ASCII function syntax	148
concurrency	58	ASIN function syntax	148
encrypting	36	ATAN function syntax	148
introduction	6, 28	ATAN2 function syntax	148
Palm OS	190	ATN2 function syntax	148
properties	33	AVG function syntax	148
storage	31	BYTE_LENGTH function syntax	148
user IDs	40	BYTE_SUBSTR function syntax	148
UltraLite documentation		CAST function syntax	148
using	8	CEILING function syntax	148
UltraLite engine		CHAR function syntax	148
about	58, 61	CHAR_LENGTH function syntax	148
deployment	61	CHARINDEX function syntax	148
starting	62	COALESCE function syntax	148
syntax	88	CONVERT function syntax	148
UltraLite engine stop utility		COS function syntax	148
syntax	100	COT function syntax	148
UltraLite generator		COUNT function syntax	148
defined	207	DATALENGTH function syntax	148
introduction	207	DATE function syntax	148
syntax	89	DATEADD function syntax	148
UltraLite initialization utility		DATEDIFF function syntax	148
about	112	DATEFORMAT function syntax	148
UltraLite Interactive SQL		DATENAME function syntax	148
about	115	DATEPART function syntax	148
tutorial	129	DATETIME function syntax	148
UltraLite passwords		DAY function syntax	148
about	40	DAYNAME function syntax	148
UltraLite programming interfaces		DAYS function syntax	148
choosing	10	DEGREES function syntax	148
UltraLite project creation wizard		DIFFERENCE function syntax	148
using	202	DOW function syntax	148
UltraLite projects		EXP function syntax	148
about	202	EXPRTYPE syntax	148
adding statements	203	FLOOR function syntax	148

GETDATE function syntax	148	SIN function syntax	148
GREATER function syntax	148	SOUNDEX function syntax	148
HEXTOINT function syntax	148	SPACE function syntax	148
HOUR function syntax	148	SQRT function syntax	148
HOURS function syntax	148	STR function syntax	148
IFNULL function syntax	148	STRING function syntax	148
INSERTSTR function syntax	148	STRTOUUID function syntax	148
INTTOHEX function syntax	148	STUFF function syntax	148
ISDATE function syntax	148	SUBSTR function syntax	148
ISNULL function syntax	148	SUBSTRING function syntax	148
LCASE function syntax	148	TAN function syntax	148
LEFT function syntax	148	TODAY function syntax	148
LENGTH function syntax	148	TRIM function syntax	148
LESSER function syntax	148	TRUNCATE function syntax	148
LIST function syntax	148	TRUNCNUM function syntax	148
LOCATE function syntax	148	UCASE function syntax	148
LOG function syntax	148	UPPER function syntax	148
LOG10 function syntax	148	UUIDTOSTR function syntax	148
LOWER function syntax	148	WEEKS function syntax	148
LTRIM function syntax	148	YMD function syntax	148
MAX function syntax	148	UltraLite statement creation wizard	
MIN function syntax	148	using	203
MINUTE function syntax	148	UltraLite static interfaces	
MINUTES function syntax	148	choosing	11
MOD function syntax	148	UltraLite temporary files	
MONTH function syntax	148	about	47
MONTHNAME function syntax	148	UltraLite user IDs	
MONTHS function syntax	148	about	40
NEWID function syntax	148	ulunload utility	
NOW function syntax	148	syntax	121
NULLIF function syntax	148	ULUtil	
PATINDEX function syntax	148	about	123
PI function syntax	148	ulxml utility	
POWER function syntax	148	about	126
QUARTER function syntax	148	creating UltraLite schema files	29
RADIANS function syntax	148	source control systems	29
REMAINDER function syntax	148	syntax	126
REPEAT function syntax	148	undoing	
REPLACE function syntax	148	changes by rolling back transactions (UltraLite)	
REPLICATE function syntax	148	180	
RIGHT function syntax	148	Unicode	
ROUND function syntax	148	UltraLite databases	45
RTRIM function syntax	148	unique	
SECOND function syntax	148	constraint (UltraLite)	176
SECONDS function syntax	148	unique indexes	
SIGN function syntax	148	UltraLite databases	172
SIMILAR function syntax	148	UNIQUEIDENTIFIER data type	

UltraLite	145	UltraLite database loading	117
universally unique identifiers		UltraLite database synchronization	119
UltraLite SQL syntax for NEWID function	148	UltraLite database unloading	121
unloading		UltraLite engine	88
UltraLite databases	101, 121	UltraLite engine stop utility	100
UPDATE statement		UltraLite generator	89
UltraLite dynamic SQL syntax	183	UltraLite Palm utility	123
updates		ulunload	121
UltraLite databases	47	UUIDs	
updating		UltraLite SQL syntax for NEWID function	148
rows for UltraLite dynamic SQL	183	UltraLite SQL syntax for STRTOUUID function	148
upgrading		UltraLite SQL syntax for UUIDTOSTR function	148
UltraLite database schema	54, 56	UltraLite SQL syntax for UUIDTOSTR function	148
UltraLite database schemas	125	UUIDTOSTR function	
UltraLite software	57	UltraLite SQL syntax	148
UPPER function		<b>V</b>	
UltraLite SQL syntax	148	VARBINARY data type	
user authentication		UltraLite	145
PASSWORD UltraLite connection parameter	76	VARCHAR data type	
sharing MobiLink and UltraLite	41	UltraLite	145
UltraLite	40	variables	
UltraLite databases	40	UltraLite SQL	143
user authentication parameters		VFSOnPalm connection parameter	
UltraLite	73	creating UltraLite databases	190
user IDs		UltraLite	81
UltraLite databases	40	virtual file system	
user-defined data types		Palm OS	81, 190
unsupported in UltraLite	145	<b>W</b>	
UserID connection parameter		warnings	
UltraLite	76	UltraLite generator	91
userid connection parameter		WEEKS function	
UltraLite	76	UltraLite SQL syntax	148
usm files		WHEN	
creating for UltraLite	29	CASE expression for UltraLite	164
usm.xsd		WHERE clause	
ulconv utility	101	SELECT statement for UltraLite dynamic SQL	181
utilities		wildcards	
SQL preprocessor for UltraLite	95	pattern matching for UltraLite	148
ulconv	101	Windows	
ulcreate	108	UltraLite character sets	45
uldbsgen	110	Windows CE	
ulisql	115	collation sequences in UltraLite	43
ulload	117	Windows CE	
ulsync	119		
UltraLite consolidated database generation	110		
UltraLite database conversion	101		
UltraLite database creation	108		

---

UltraLite character sets	44
wizards	
UltraLite project creation	202
UltraLite statement creation	203
<b>X</b>	
XML	
loading UltraLite databases from	101
saving UltraLite databases as	101
<b>Y</b>	
Y2K problem	
UltraLite NearestCentury option	34
year 2000	
NEAREST_CENTURY option for UltraLite	199
YMD function	
UltraLite SQL syntax	148