UltraLite™ User's Guide for PenRight! MobileBuilder

Last modified: October 2002
Part Number: 38175-01-0802-01

Last modified October 2002. Part number 38175-01-0802-01.

Contents

About This Manual

Subject

Sybase UltraLite technology provides a relational database system for mobile devices, including built-in synchronization with enterprise data. This book describes how to build UltraLite applications using PenRight! MobileBuilder 2.0, the integrated development environment (IDE) for mobile computing platforms.

Audience

This manual is intended for anyone building UltraLite applications using MobileBuilder. It is a supplement to the MobileBuilder documentation and to the *UltraLite User's Guide*, included in the SQL Anywhere Studio documentation set. The *UltraLite User's Guide* describes UltraLite applications and databases in detail.

Before you begin

This book assumes an elementary familiarity with the following:

- MobileBuilder and the C programming language.
- The basic architecture of UltraLite applications and MobiLink synchronization, as described in "Introduction to UltraLite" on page 3 of the book *UltraLite User's Guide*.
- Relational databases and SQL, as described in *Adaptive Server Anywhere Getting Started*.

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 set

The SQL Anywhere Studio documentation set 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 Getting Started** This book is for people new to relational databases or new to Adaptive Server Anywhere. It provides a quick start to using the Adaptive Server Anywhere database-management system and introductory material on designing, building, and working with databases.
- **Adaptive Server Anywhere Database Administration Guide** This book covers material related to running, managing, and configuring databases.
- **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.

- **Adaptive Server Anywhere Error Messages** This book provides a complete listing of Adaptive Server Anywhere error messages together with diagnostic information.
- **Adaptive Server Anywhere C2 Security Supplement** 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. The book does *not* include the security features added to the product since certification.
- **MobiLink Synchronization User's Guide** This book describes all aspects of 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.
- **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.
- **UltraLite User's Guide** This book describes how to build database applications for small devices such as handheld organizers using the UltraLite deployment technology for Adaptive Server Anywhere databases.
- **UltraLite User's Guide for PenRight! MobileBuilder** This book is for users of the PenRight! MobileBuilder development tool. It describes how to use UltraLite technology in the MobileBuilder programming environment.
- **SQL Anywhere Studio Help** This book is provided online only. It includes the context-sensitive help for Sybase Central, Interactive SQL, and other graphical tools.

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

Documentation formats

SQL Anywhere Studio provides documentation in the following formats:

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

  To access the online books on Windows operating systems, choose Start➤Programs➤Sybase SQL Anywhere 8➤Online Books. You can navigate the online books using the HTML Help table of contents, index, and search facility in the left pane, and using the links and menus in the right pane.

  To access the online books on UNIX operating systems, run the following command at a command prompt:

  ```
  dbbooks
  ```

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

  The PDF files are available on the CD ROM in the *pdf_docs* directory. You can choose to install them when running the setup program.

- **Printed books** The following books are included in the SQL Anywhere Studio box:

  - *Introducing SQL Anywhere Studio*.
  - *Adaptive Server Anywhere Getting Started*.
  - *SQL Anywhere Studio Quick Reference*. This book is available only in printed form.

  The complete set of books is available as the SQL Anywhere Documentation set from Sybase sales or from e-Shop, the Sybase online store, at http://e-shop.sybase.com/cgi-bin/eshop.storefront/.

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 are shown 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. 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.

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

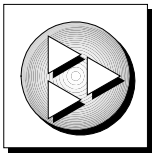
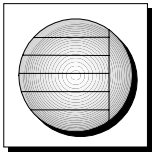
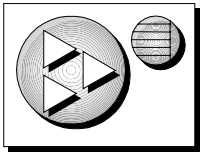
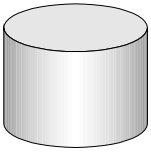
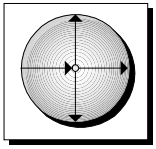
If the QUOTES option is chosen, one of ON or OFF must be provided. The brackets and braces should not be typed.

- **One or more options** If you choose more than one, separate your choices with commas.

  { **CONNECT**, **DBA**, **RESOURCE** }

Graphic icons

The following icons are used in this documentation:

Icon	Meaning
	A client application.
	A database server, such as Sybase Adaptive Server Anywhere or Adaptive Server Enterprise.
	An UltraLite application and database server. In UltraLite, the database server and the application are part of the same process.
	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, SQL Remote Message Agent, and the Replication Agent (Log Transfer Manager) for use with Replication Server.
	A Sybase Replication Server.
	A programming interface.

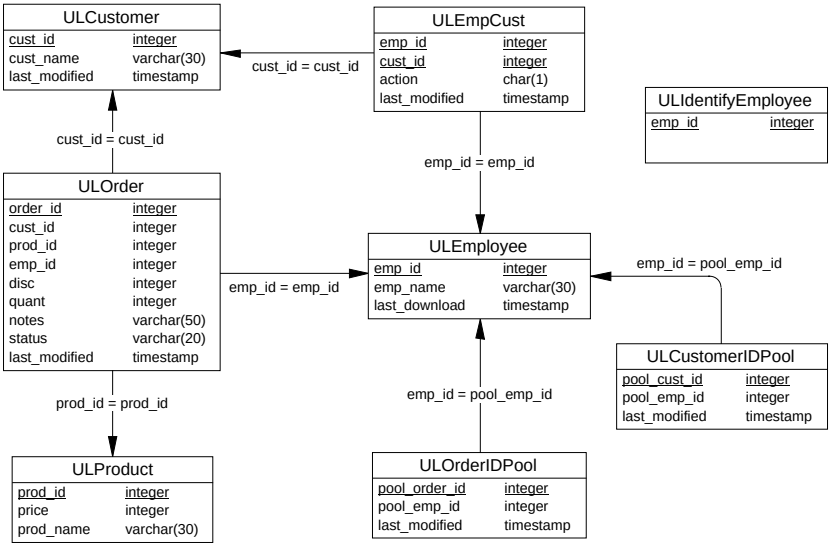
The UltraLite sample database

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

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 figure shows the tables in the CustDB database and how they are related to each other.



Finding out more and providing feedback

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

You can provide feedback on this documentation and on the software through newsgroups set up to discuss SQL Anywhere technologies. These newsgroups can be found 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.

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 insure 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.

CHAPTER 1

Providing Solutions with UltraLite and MobileBuilder

About this chapter

Together, UltraLite and MobileBuilder provide solutions to the problem of developing data-centric applications for mobile devices. This chapter provides an overview of the benefits of using these two technologies together.

Contents

Combining UltraLite and MobileBuilder

MobileBuilder and UltraLite together provide a complete solution for building data-centric applications for small devices. MobileBuilder provides a cross-platform rapid application development environment for small devices, and UltraLite provides the relational database and synchronization technologies to manage and synchronize data.

PenRight! MobileBuilder is a separate product, not included with SQL Anywhere Studio.

UltraLite overview

UltraLite is a deployment technology for Adaptive Server Anywhere databases, aimed at small, mobile, and embedded devices. Target platforms include cell phones, pagers, and personal organizers.

UltraLite provides the following benefits for users of small devices:

- The functionality and reliability of a transaction-processing SQL database.
- The ability to synchronize data with a central database-management system.
- An extremely small memory footprint.

Full-featured SQL

UltraLite allows applications on small devices to use full-featured SQL to accomplish data storage, retrieval, and manipulation. UltraLite supports referential integrity, transaction processing, and multi-table joins of all varieties. In fact, UltraLite supports most of the same data types, runtime functions, and SQL data manipulation features as Sybase Adaptive Server Anywhere.

Synchronization to industry-standard RDBMS

UltraLite uses Sybase MobiLink synchronization technology to synchronize with industry-standard database-management systems. MobiLink synchronization works with ODBC-compliant data sources such as Sybase Adaptive Server Anywhere, Sybase Adaptive Server Enterprise, IBM DB2, Microsoft SQL Server, and Oracle.

C/C++ UltraLite custom database engines for your application can be as small as 50 kb, depending on your deployment platform and the number of SQL statements in the application and the SQL features used.

MobileBuilder overview

MobileBuilder is a rapid application development tool for small devices. It uses the C programming language to provide the small executables needed on target platforms with very limited resources. MobileBuilder provides a development environment that makes cross platform development easy. The MobileBuilder form designer, built-in controls, and Code Assistant ease the path to developing full C applications.

PenRight! MobileBuilder is a separate product, not included with SQL Anywhere Studio.

The benefits of using UltraLite and MobileBuilder

Using UltraLite and MobileBuilder together, you can develop powerful database-hosted applications for small devices, complete with enterprise synchronization, in a productive and convenient development environment.

MobileBuilder assists you with form design, compiling, cross-platform development. UltraLite provides the underlying database technology, and an UltraLite component within MobileBuilder provides easy access to UltraLite features from within the MobileBuilder environment.

Installing the UltraLite component

The installation process consists of the following steps:

1 Confirm that you have the system requirements for using UltraLite and MobileBuilder.

For information, see "System requirements" on page 4.

2 Install the files into their proper locations. The UltraLite component for MobileBuilder is installed as part of your SQL Anywhere Studio installation.

3 Add the UltraLite component group to your MobileBuilder interface.

For information, see "Adding the UltraLite component group to MobileBuilder" on page 5.

System requirements

This section describes the system requirements for using the UltraLite component group for MobileBuilder. It describes requirements for the development or host machine. The UltraLite component is included in SQL Anywhere Studio.

For information on requirements for the target platforms to which you can deploy your applications, see "Supported target platforms" on page 5.

Required software

To install and use the UltraLite component group for MobileBuilder, you must have the following software on your computer:

- PenRight! MobileBuilder 2.0 build 889 or later.
- Sybase SQL Anywhere Studio.

  You must have installed Adaptive Server Anywhere and UltraLite from the SQL Anywhere Studio setup program, and you must install MobiLink synchronization if you wish to add synchronization to your applications.
- A compiler for your target platform.

Operating system requirements

You must be running on one of the following operating systems:

- Windows 95/98/Me.
- Windows NT, Windows 2000, or Windows XP.

Supported target platforms

You can develop applications for the following target platforms:

- Windows operating systems (Windows 95/98/Me, Windows NT/2000/XP).
- The Palm Computing Platform using the GNU compiler.

You must have a MobileBuilder-supported compiler installed in order to develop applications for a particular target platform.

Adding the UltraLite component group to MobileBuilder

Once you have run the SQL Anywhere Studio setup program to install the UltraLite component group, you still need to make MobileBuilder aware of the new component.

❖ **To add the UltraLite component group to MobileBuilder:**

1. Start MobileBuilder.

   PenRight! MobileBuilder is a separate product, not included with SQL Anywhere Studio.

2. From the Tools menu, choose Options.

   The Options dialog appears.

3. Click the Component Groups tab, and click Install.

4. Locate the file *DBComp.mbg* in the *Comps\UltraLite* subdirectory of your MobileBuilder directory, and click Open.

   An entry for UltraLite appears in the list of component groups.

   A copy of *DBComp.mbg* is also installed into the *UltraLite\MobileBuilder* subdirectory of your SQL Anywhere directory.

5. In the Options dialog, click OK to complete the process.

   An UltraLite tab appears in the Component groups toolbar at the top of your MobileBuilder window.

The UltraLite component group contains a single component, named **ULDatabase**.

CHAPTER 2

Tutorial: Build an UltraLite Application Using MobileBuilder

About this chapter

This chapter provides a tutorial that guides you through the process of developing an UltraLite application using MobileBuilder. It describes how to build a very simple application, and how to add synchronization to your application.

Contents

Introduction

This tutorial describes how to create a simple UltraLite application using MobileBuilder. It includes the following topics:

- How to create an UltraLite database.
- How to add the UltraLite database control to your MobileBuilder project.
- How to configure the UltraLite database control for your application.
- How to bind a user interface component to a column of an UltraLite table.
- How to use the MobileBuilder Code Assistant to add functionality to your application.
- How to build and preview your application.

For the simple application you build in this tutorial, you do not need to write any C code directly. The application carries out the following tasks.

1. Connects to an UltraLite database, consisting of a single table. The table is the *ULProduct* table of the UltraLite Sample database.
2. Synchronizes with the consolidated database.
3. Displays some of the data in a list box.

Lessons

The lessons in this tutorial carry out the following steps:

1. Create a directory to hold the tutorial files.
2. Create a MobileBuilder project.
3. Add the UltraLite component to your MobileBuilder project.
4. Design the UltraLite database for the application in an Adaptive Server Anywhere reference database.
5. Configure the UltraLite component.
6. Design a form and bind controls to the UltraLite database.
7. Compile, link, and run the application.

Lesson 1: Getting started

In this lesson you prepare for the remainder of the tutorial. You make a directory to hold the files you will be using during the tutorial. In addition, you should make a copy of the UltraLite sample database as a backup copy so that you can easily restore it to its original state for future use.

During the tutorial, we use the original UltraLite Sample database. At the end of the tutorial, you can copy the untouched version from the *Tutorial* directory back into place.

❖ **Prepare a tutorial directory:**

- From a command prompt, enter the following command:

  ```
  mkdir c:\Tutorial
  ```

  This creates a directory to hold the tutorial files. You can choose your own name for the directory, but in the remainder of the tutorial, we assume that this directory is *c:\Tutorial*.

❖ **Copy the sample database:**

- From a command prompt, change to the *Samples\UltraLite \CustDB* subdirectory of your SQL Anywhere directory. This directory holds the UltraLite Sample database.

  Enter the following command:

  ```
  copy CustDB.db c:\Tutorial
  ```

  This makes a copy of the UltraLite Sample database in the tutorial directory.

Lesson 2: Create a MobileBuilder project

In this lesson, you start building your MobileBuilder application. Here, you create a new MobileBuilder project and add the UltraLite database component to it.

❖ **Create a MobileBuilder project:**

1 Start MobileBuilder.

2 Create the project:

- From the File menu, choose New Project. The New Project dialog appears.
- Enter **Tutorial** as your project name
- Enter your tutorial directory (**c:\Tutorial**) as the location.
- Choose **C** as the language.
- Choose a target for which you have a compiler installed. This tutorial uses **Win32** as the target (Visual C++ 6.0 for Windows 32), but you can use any other compiler for a supported MobileBuilder target platform.
- Click OK to create the project. A Project Manager Pane appears on the left side of your MobileBuilder window.

You have created a standard MobileBuilder project. The next lesson describes how to add UltraLite features to this project.

Lesson 3: Add the UltraLite Component to your project

The UltraLite database component manages all the UltraLite-specific tasks within your MobileBuilder project. This lesson describes how to add the component to your project and configure it for use with your reference database.

The UltraLite database component is a global component. It is not visible, and you do not need to add it to any form in your project. You need only one instance of the component in any application, and the component holds application-wide properties.

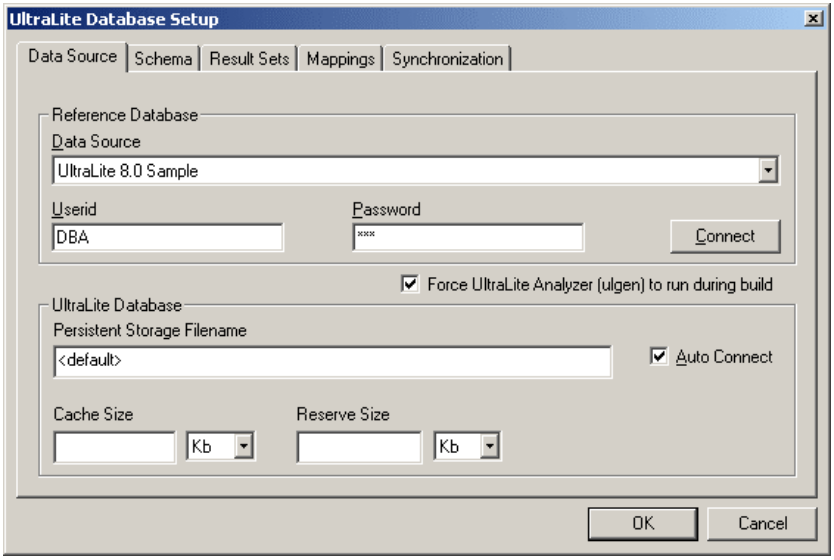
❖ **Add the UltraLite database component to your project:**

1 Add the UltraLite database component to your project:

- On the Component Palette at the top of your MobileBuilder window, click the UltraLite tab.
- Double-click the ULDatabase component. An UltraLite database component appears in the Global Components folder of your project. It has the name **ULDatabase1**.

2 Set the data source to be your reference database.

- In the Project Management Pane, open the Global Components folder and double-click ULDatabase1. The Properties tab for this component appears.
- Click the button next to the DataSource property. The UltraLite database setup dialog appears, open at the Data Source tab.



- From the Data Source dropdown list, choose **UltraLite 8.0 Sample**. This identifies the UltraLite sample database as your reference database.
- Enter **DBA** as user ID, and **SQL** as password. These entries are used during connection, and are also used by other pieces of the UltraLite database component.
- Click Connect to establish a connection to the reference database. An Adaptive Server Anywhere personal database server starts, and the UltraLite component connects to it.
- Leave the UltraLite database properties at their default settings. Ensure that Auto Connect is selected.
- Click OK to save the settings.

  You must save the Data Source settings the first time you add them, before setting other UltraLite properties.

Lesson 4: Define your UltraLite database schema

Any UltraLite application requires a database. You design your UltraLite database in a **reference database**: an Adaptive Server Anywhere database that holds the tables your UltraLite application needs.

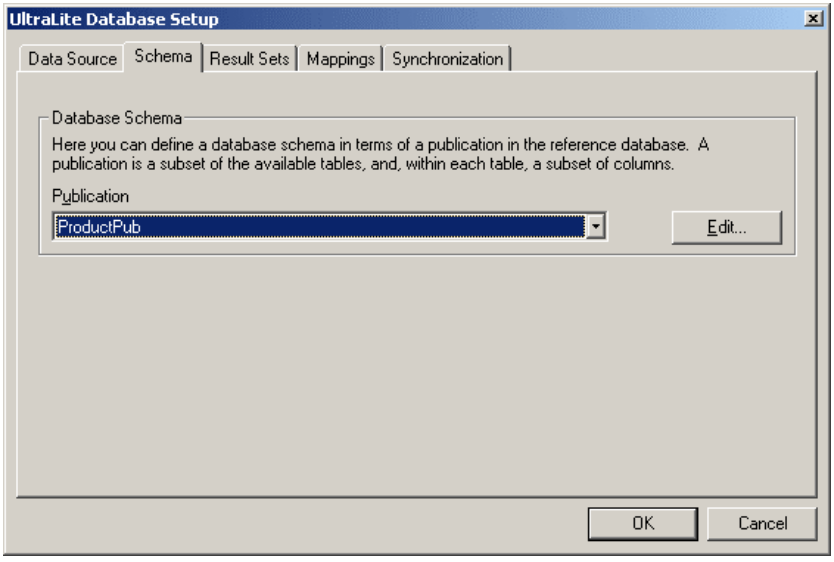
In this tutorial, you create an object called a **publication** in your reference database. The publication is a convenient device for assembling tables and column-based subsets of tables.

You can also define your UltraLite database by adding SQL statements to the reference database. SQL statements allow you to include joins and more advanced features in your UltraLite application.

❖ Create your UltraLite database schema:

1 Prepare to define the schema.

- In the Properties tab for the ULDatabase1 component, click the button by the Schema property. The UltraLite database setup dialog appears, open at the Schema tab.



2 Create a publication that reflects the data you wish to include in your UltraLite database.

- Click the Edit button. The Create Publication dialog appears.

The Table.Column list displays all columns of tables owned by the user ID you used in your connection.

- In the Publication Name field, enter the name **ProductPub**.
- Double-click the following columns to include them in the publication:
  - **ULProduct.prod_id**
  - **ULProduct.prod_name**
  - **ULProduct.price**
- Click OK to create the publication.
- In the UltraLite Database Setup dialog, ensure that **Product Pub** is selected in the Publication list, and click OK to complete the definition.

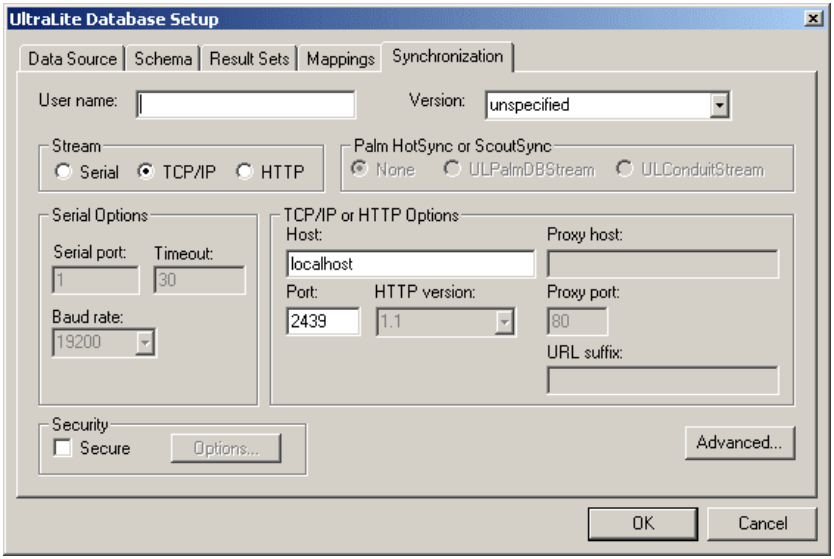
You have now finished designing your UltraLite database; in this case, just a single table. In the next lesson, you set up synchronization.

Lesson 5: Configure synchronization

There are several parameters that need to be selected in order to synchronize successfully. The meaning of the individual parameters is not described here. Instead, the section says which settings to use, and gives cross references for more information.

❖ **Configure the synchronization parameters:**

1 On the ULDatabase1 property tab, click the button next to the Synchronization property. The UltraLite Database Setup dialog appears, open at the Synchronization tab.



2 In the User name field, enter **50**. This value identifies the user for synchronization.

For more information, see "The MobiLink user" on page 22 of the book *MobiLink Synchronization User's Guide.*

3 In the version field, choose **custdb**. This identifies the particular instructions (already stored in the UltraLite Sample database) to be used for synchronization.

For more information, see "Script versions" on page 61 of the book *MobiLink Synchronization User's Guide*.

4 Leave the Stream setting at **TCP/IP**, and the Host setting at **localhost**, and the Port setting ad **2439**. Leave the Security option unchecked.

5 Click OK to complete the settings.

You have now configured the UltraLite database component. In this tutorial, you use default synchronization options. You set the Mappings properties in the next lesson.

Lesson 6: Design a MobileBuilder form

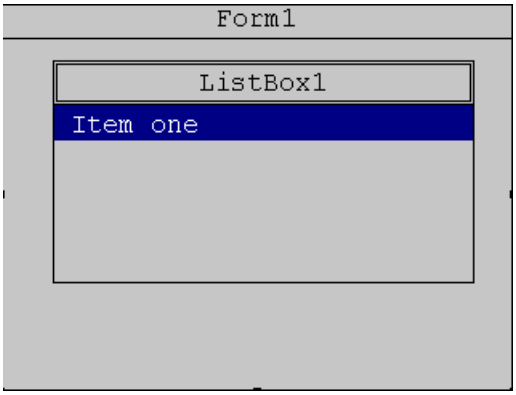
In this lesson you design a MobileBuilder form, and bind a list control on the form to the UltraLite database component.

❖ **Design the MobileBuilder form:**

1 Open the Project tab of the Project Manager Pane, and open the Forms folder.

2 Add a list box to the form.

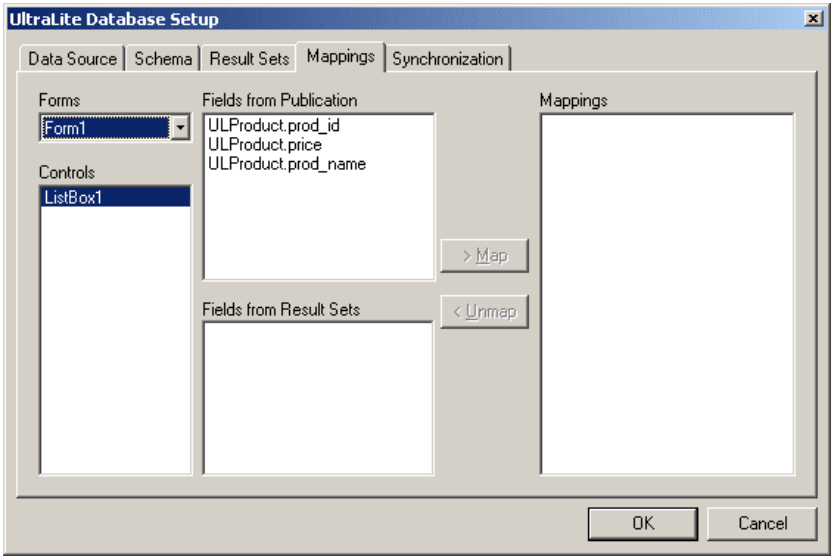
- In the Project Management pane, double-click Form1 to display it in the right pane.
- On the Component Palette, click Standard.
- Click the Listbox icon (TList component), and draw a list box on the form. The list box can take up most of the space on the form. By default, the listbox is named ListBox1.



3 Bind the listbox to a column of the UltraLite database.

You bind controls to UltraLite objects from the UltraLite database component.

- In the Project Management pane, double-click the **ULDatabase1** control.
- Click the button beside the Mappings item. The UltraLite database Setup dialog appears, open at the Mappings tab.



- Ensure that **Form1** is selected in the Forms list, and that **ListBox1** is selected in the Controls list.
- In the Fields From Publication list, click **ULProduct.prod_name**. Click Map to add this field to the mappings list.
- Click OK to complete the mapping.

4 Use the MobileBuilder Code Assistant to add code that opens the **ULProduct** table to the Form initialization event.

- In the MobileBuilder right-hand pane, right-click the form (not the list box) and choose Code Assistant from the popup menu. Make the following settings:

Field	Value
When this component	**Form1**
Dropdown box next to **Form1**	Empty
fires this event	**FrmInit**
then, with this component	**Application**
Dropdown box next to **Application**	**ULDatabase1**
do the following	**Open a table or result set**

- Click Add Code, and in the dialog that appears, choose **ULProduct**.

5 Use the MobileBuilder Code Assistant to add code that synchronizes data to the Form initialization event.

- In the Code Assistant, make the following settings:

Field	Value
When This Component	**Form1**
Dropdown box next to **Form1**	Empty
fires This Event	**FrmInit**
then, with this component	**Application**
Dropdown box next to Application	**ULDatabase1**
Do the following	**Synchronize the connection**

- Click Add Code to finish.

The code is added after the existing code for the **FrmInit** event.

6 Use the MobileBuilder Code Assistant to add code that sets the current item to be the first item in the table. Add this code to the Form initialization event.

- In the Code Assistant, make the following settings:

Field	Value
When This Component	**Form1**
Dropdown box next to **Form1**	Empty
fires This Event	**FrmInit**
then, with this component	**Application**
Dropdown box next to Application	**ULDatabase1**
Do the following	**Move to first record**

- Click Add Code, and in the dialog that appears, choose **ULProduct**.
- Click OK to finish.
- Click Close to close the Code Assistant.

You have now completed the application, which displays the product names in the database. You can look at the generated code in the code tab of the MobileBuilder right pane. Next, you compile and run your application.

Lesson 7: Build and run your application

You can compile and link your application in the development tool of your choice. In this section, we describe how to compile and link using Visual C++; if you are using one of the other supported development tools, modify the instructions to fit your tool.

❖ **Build and run your application:**

1 Save the application.

From the File menu, choose Save. Accept the default name and click Save.

2 From the MobileBuilder Project menu, choose Build.

When compiling is complete, the Output Pane at the bottom of the MobileBuilder workspace should display a line stating the following:

```
Tutorial.exe – 0 error(s), 0 warning(s)
```

> **Compiler failure**
> This tutorial assumes that you have a working installation of Visual C++. If the build fails, you should check that your Visual C++ installation is correct. If problems persist, consult your MobileBuilder documentation.

3 As you have configured the application to synchronize on startup, you must start the MobiLink synchronization server so that a connection can be established.

From the Windows Start button, choose Programs➤Sybase SQL Anywhere 8➤MobiLink➤Synchronization Server Sample.

The MobiLink synchronization server starts, and displays that it is ready to handle requests.

4 Run the application.

From the MobileBuilder Project menu, choose Run.

A MobileBuilder window appears and displays the form you designed, holding the product names from the product table.

If you look at the MobiLink synchronization server window, you will see a set of messages, finishing with Synchronization complete.

You have now built and run a simple UltraLite application using MobileBuilder. You can close MobileBuilder and the MobiLink synchronization server.

Restore the sample database

Now that you have completed the tutorial, you should restore the sample database so that it can be used again. You created a backup copy of the UltraLite Sample database in "Lesson 1: Getting started" on page 9. You can now replace the modified version of *custdb.db* with the backup copy, overwriting any changes.

❖ **Restore the sample database:**

1. Delete the database file *custdb.db*, and the transaction log file *custdb.log* in the *Samples**UltraLite**CustDB* subdirectory of your SQL Anywhere directory.

   From a command prompt, change to the *Samples**UltraLite**CustDB* subdirectory of your SQL Anywhere directory and enter the following command:

   ```
   dberase -y custdb.db
   ```

2. Copy the *custdb.db* file from your tutorial directory to the *Samples**UltraLite**CustDB* subdirectory of your SQL Anywhere directory.

Your sample database is now restored to its original state.

CHAPTER 3

Tutorial: Build an UltraLite Palm Application Using MobileBuilder

About this chapter

This chapter provides a tutorial that guides you through the process of developing an UltraLite application for the Palm Computing Platform using MobileBuilder. It describes how to build a very simple application, and how to add synchronization to your application.

Contents

Introduction

This tutorial describes how to create a simple UltraLite application for the Palm Computing Platform using MobileBuilder. It includes the following topics:

- How to create an UltraLite database.
- How to add the UltraLite database control to your MobileBuilder project.
- How to configure the UltraLite database control for your application.
- How to bind a user interface component to a column of an UltraLite table.
- How to use the MobileBuilder Code Assistant to add functionality to your application.
- How to build and preview your application.

The application carries out the following tasks.

1. Connects to an UltraLite database, consisting of a single table. The table is the *ULProduct* table of the UltraLite Sample database.
2. Synchronizes with the consolidated database.
3. Displays some of the data in text fields.

Lessons

The lessons in this tutorial carry out the following steps:

1. Create a directory to hold the tutorial files.
2. Create a MobileBuilder project.
3. Add the UltraLite component to your MobileBuilder project.
4. Design the UltraLite database for the application in an Adaptive Server Anywhere reference database.
5. Configure the UltraLite component.
6. Design a form and bind controls to the UltraLite database.
7. Compile, link, and run the application.

Lesson 1: Getting started

In this lesson you prepare for the remainder of the tutorial. You make a directory to hold the files you will be using during the tutorial. In addition, you should make a copy of the UltraLite sample database as a backup copy so that you can easily restore it to its original state for future use.

During the tutorial, we use the original UltraLite Sample database. At the end of the tutorial you can copy the untouched version from the *Tutorial* directory back into place.

❖ **Ensure you have the required software installed:**

- To carry out this tutorial, you need the following software:
  - SQL Anywhere Studio.
  - PenRight! MobileBuilder 2.0 or later. Build 889 or higher is required.
  - Palm OS SDK version 3.1 or version 3.5.
  - Palm emulator (included with MobileBuilder).
  - The PRC Tools development suite for the Palm Computing Platform.

    You need only the **cygwin** user tools, not the full PRC Tools install. You do not need to install PilRC as stated, because MobileBuilder includes a version of PilRC.

    When you have installed PRC Tools, you can confirm that MobileBuilder has detected the tools. In MobileBuilder, choose Tools➤Options➤Build Tools. Highlight PRC-Tools (do not check the check box) and click Detect.

    If the check box becomes checked, MobileBuilder has successfully detected PRC-Tools.

  For more information on PRC Tools and the Palm emulator, see the Palm developer zone at http://www.palmos.com/dev/.

❖ **Prepare a tutorial directory:**

- From a command prompt, enter the following command:

  ```
  mkdir c:\Tutorial
  ```

  This creates a directory to hold the tutorial files. You can choose your own name for the directory, but in the remainder of the tutorial, we assume that this directory is *c:\Tutorial*.

❖ **Copy the sample database:**

♦ From a command prompt, change to the *Samples\UltraLite\CustDB* subdirectory of your SQL Anywhere directory. This directory holds the UltraLite Sample database.

Enter the following command:

```
copy CustDB.db c:\Tutorial
```

This makes a copy of the UltraLite Sample database in the tutorial directory.

Lesson 2: Create a MobileBuilder project

In this lesson, you start building your MobileBuilder application. Here, you create a new MobileBuilder project and add the UltraLite database component to it.

❖ **Create a MobileBuilder project:**

1 Start MobileBuilder.

2 Create the project.

- From the File menu, choose New Project. The New Project dialog is displayed.
- Enter **Tutorial** as your project name
- Enter your tutorial directory (**c:\Tutorial**) as the location.
- Choose **C** as the language.
- Choose **Palm** as the target.
- Click OK to create the project. A Project Manager Panel is displayed on the left side of your MobileBuilder window and a Palm menu is added to the MobileBuilder interface.

You have created a standard MobileBuilder project. The next lesson describes how to add UltraLite features to this project.

Lesson 3: Add the UltraLite Component to your project

The UltraLite database component manages all the UltraLite-specific tasks within your MobileBuilder project. This lesson describes how to add the component to your project and configure it for use with your reference database.

The UltraLite database component is a global component. It is not visible, and you do not need to add it to any form in your project. You need only one instance of the component in any application, and the component holds application-wide properties.

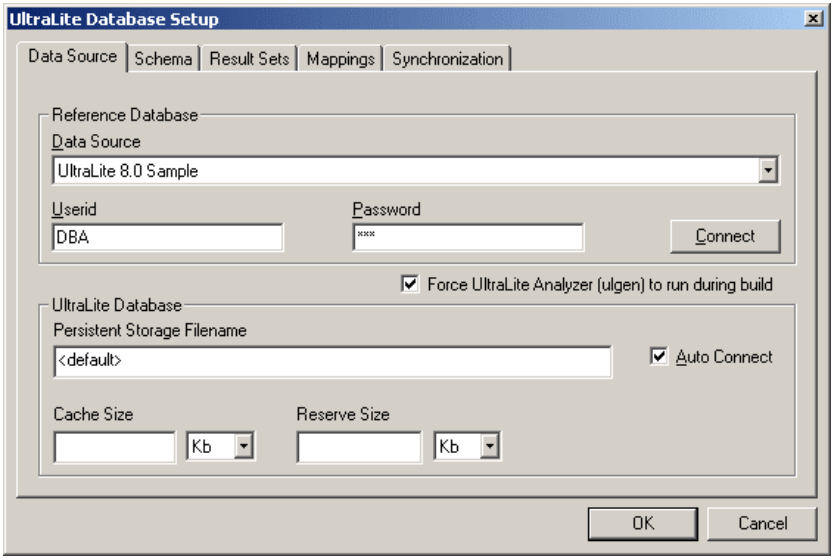
❖ **Add the UltraLite database component to your project:**

1 Add the UltraLite database component to your project.

- On the Component Palette at the top of your MobileBuilder window, click the UltraLite tab.
- Double-click the ULDatabase component. An UltraLite database component appears in the Global Components folder of your project. It has the name **ULDatabase1**.

2 Set the data source to be your reference database.

- In the Project Management Panel, open the Global Components folder and double-click ULDatabase1. The Properties tab for this component appears.
- Click the button next to the DataSource property. The UltraLite database setup dialog appears, open at the Data Source tab.



- From the Data Source drop-down list, choose **UltraLite 8.0 Sample**. This identifies the UltraLite sample database as your reference database.
- Enter **DBA** as user ID, and **SQL** as password. These entries are used during connection, and are also used by other pieces of the UltraLite database component.
- Click Connect to establish a connection to the reference database. An Adaptive Server Anywhere personal database server starts, and the UltraLite component connects to it.
- Leave the UltraLite database properties at their default settings. Ensure that Auto Connect is selected.
- Click OK to save the settings.

  You must save the Data Source settings the first time you add them, before setting other UltraLite properties.

Lesson 4: Define your UltraLite database schema

Any UltraLite application requires a database. You design your UltraLite database in a **reference database**: an Adaptive Server Anywhere database that holds the tables your UltraLite application needs.

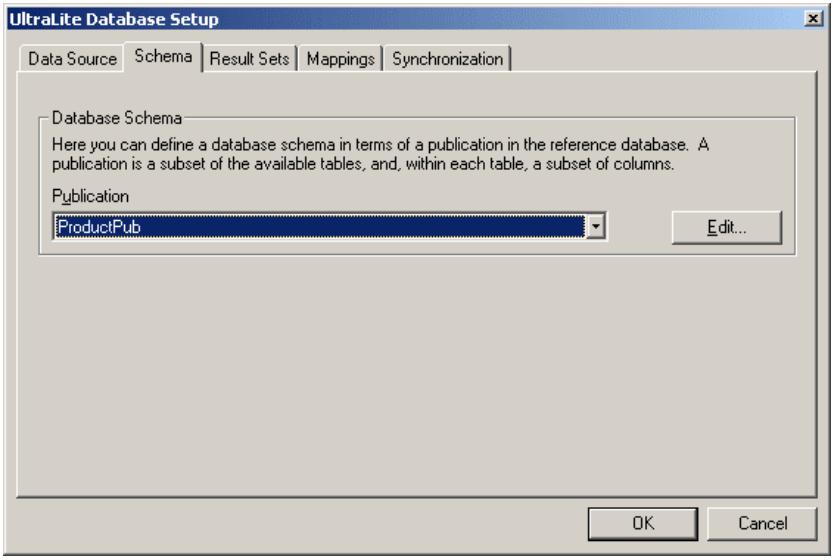
In this tutorial, you create an object called a **publication** in your reference database. The publication is a convenient device for assembling tables and column-based subsets of tables.

You can also define your UltraLite database by adding SQL statements to the reference database. SQL statements allow you to include joins and more advanced features in your UltraLite application.

❖ Create your UltraLite database schema:

1 Prepare to define the schema.

- In the Properties tab for the ULDatabase1 component, click the button by the Schema property. The UltraLite database setup dialog appears, open at the Schema tab.



2 Create a publication that reflects the data you wish to include in your UltraLite database.

- Click the Edit button. The Create Publication dialog appears.

The Table.Column list displays all columns of tables owned by the user ID you used in your connection.

- In the Publication Name field, enter the name **ProductPub**.
- Double-click the following columns to include them in the publication:
  - **ULProduct.prod_id**
  - **ULProduct.prod_name**
  - **ULProduct.price**
- Click OK to create the publication.
- In the UltraLite Database Setup dialog, ensure that **Product Pub** is selected in the Publication list, and click OK to complete the definition. Save your work.

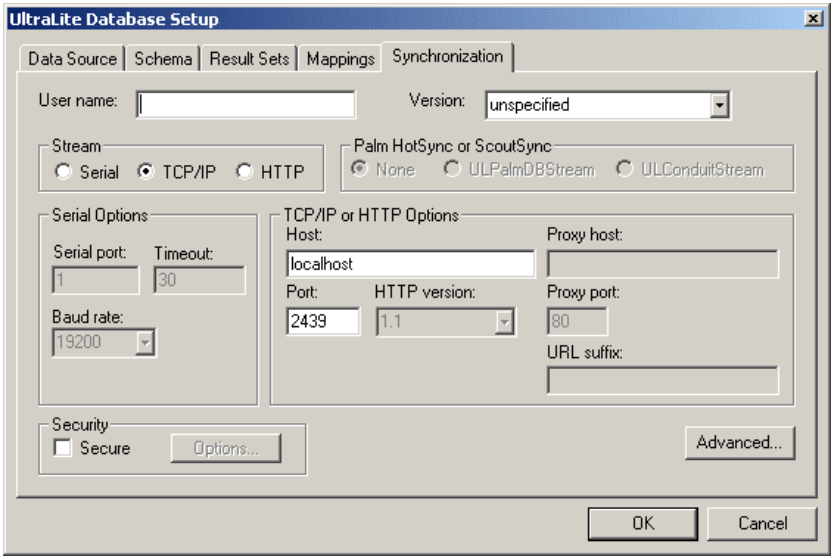
You have now finished designing your UltraLite database; in this case, just a single table. In the next lesson, you set up synchronization.

Lesson 5: Configure synchronization

There are several parameters that need to be selected in order to synchronize successfully. The meaning of the individual parameters is not described here. Instead, the section says which settings to use, and gives cross references for more information.

❖ **Configure the synchronization parameters:**

1 In the ULDatabase1 property tab, click the button next to the Synchronization property. The UltraLite Database Setup dialog appears, open at the Synchronization tab.



2 In the User name field, enter **50**. This value identifies the user for synchronization.

For more information, see "Authenticating MobiLink Users" on page 251 of the book *MobiLink Synchronization User's Guide*.

3 In the version field, choose **custdb**. This identifies the particular instructions (already stored in the UltraLite Sample database) to be used for synchronization.

For more information, see "Script versions" on page 61 of the book *MobiLink Synchronization User's Guide*.

4 Leave the Stream setting at **TCP/IP**, the Host setting at **localhost**, and the Port setting at **2439**. Leave the Security option unchecked.

It is more common to use HotSync or ScoutSync synchronization than TCP/IP for Palm devices, but here we use TCP/IP so that the application can be tested from the emulator without requiring a serial cable, which is needed for HotSync synchronization from the emulator.

At the end of the tutorial, settings are given for HotSync synchronization.

5 Click OK to complete the settings. Save your work.

You have now configured the UltraLite database component. In this tutorial, you use default synchronization options. You set the Mappings properties in the next lesson.

Lesson 6: Design a MobileBuilder form

In this lesson you design a MobileBuilder form, and bind a list control on the form to the UltraLite database component.

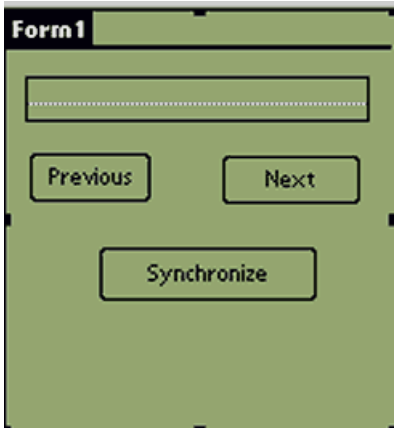
❖ **Design the MobileBuilder form:**

1 Open the Project tab of the Project Manager Panel, and open the Forms folder.

2 Add a text field to the form. This field is used to display the Product name of the current product as you scroll through a list of products.

- In the Project Management pane, double-click Form1 to display it in the right pane.
- On the Component Palette, click Standard.
- Click the text field icon (TField component), and draw a text field on the form. The text field can take up most of the space on the top half of the form. By default, the text field is named Field1.

  The following figure shows the complete form. At the moment, just the top text field is in place.



3 Add a Previous button to the form. This button is used to scroll backwards through a list of products.

- With Form1 displayed, and the Standard toolbar displayed, click the button icon (TButton component), and draw a button underneath the text field on the left side of the form. This will be the back scrolling button.This button is called Button1

- Right-click the button and select Properties from the drop down list. The Property sheet for the button is displayed in the left pane.
- Set the caption of the button to Previous.

4 Add a Next button to the form. This button is used to scroll forward through a list of products.

- Repeat the previous step. This time, draw the button on the right side of the form, underneath the text field. This will be the forward scrolling button. This button is called Button2
- Set the caption to Next.

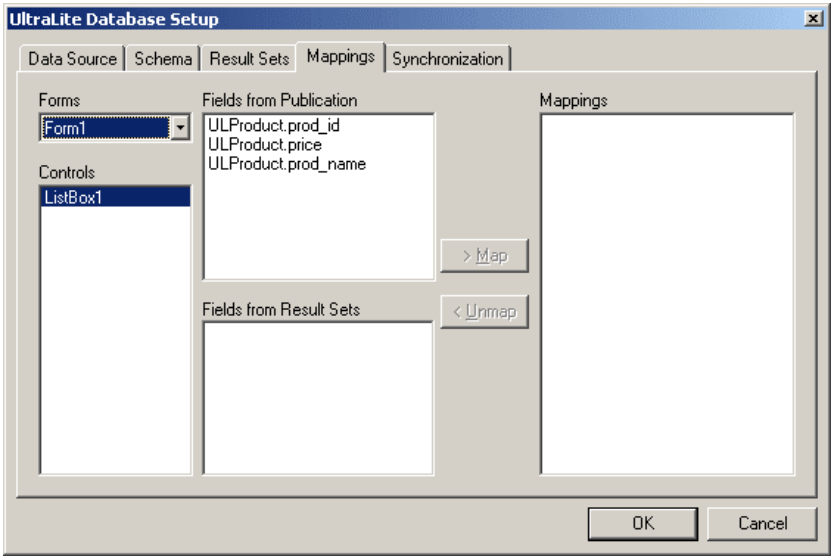
5 Add a Synchronize button to the form. This button is used to synchronize the application with the MobiLink synchronization server.

- Draw a button underneath the Previous and Next buttons. This button is called Button3
- Set the caption to Synchronize.

6 Bind the Text field to a column of the UltraLite database.

You bind controls to UltraLite objects from the UltraLite database component.

- In the Project Management pane, double-click the UltraLite database control.
- Click the button beside the Mappings item. The UltraLite database Setup dialog appears, open at the Mappings tab.



- Ensure that **Form1** is selected in the Forms list, and that **Field1** is selected in the Controls list.
- In the Fields From Publication list, click **ULProduct.prod_name**. Click Map to add this field to the mappings list.
- Click OK to complete the mapping.

7 Use the MobileBuilder Code Assistant to add code that synchronizes data to the Synchronize button.

- In the MobileBuilder right-hand pane, right-click the form and choose Code Assistant from the popup menu.
- In the Code Assistant, make the following settings:

Field	Value
When This Component	**Form1**
Drop down box next to **Form1**	**Button3**
fires This Event	**CtlSelect**
then, with this component	**Application**
Drop down box next to Application	**ULDatabase1**
Do the following	**Synchronize the connection**

- Click Add Code.
- Leave the Code Assistant open.

8 Use the MobileBuilder Code Assistant to add code that opens the table for use after synchronizing.

- In the Code Assistant, make the following settings:

Field	Value
When this component	**Form1**
Drop down box next to **Form1**	Empty
fires this event	**FrmInit**
then, with this component	**Application**
Drop down box next to **Application**	**ULDatabase1**
do the following	**Open a table or result set**

- Click Add Code, and in the dialog that appears, choose **ULProduct**.
- Click OK to finish. Leave the Code Assistant open.

9 Use the MobileBuilder Code Assistant to add code that scrolls through the data to the Previous and Next buttons.

- In the Code Assistant, make the following settings:

Field	Value
When This Component	**Form1**
Drop down box next to **Form1**	**Button1**
fires This Event	**CtlSelect**
then, with this component	**Application**
Drop down box next to Application	**ULDatabase1**
Do the following	**Move to previous record**

- Click Add Code, and in the dialog that appears, choose **ULProduct**.
- Click OK to finish. Leave the Code Assistant open.
- In the Code Assistant, make the following settings:

Field	Value
When This Component	**Form1**
Drop down box next to **Form1**	**Button2**
fires This Event	**CtlSelect**
then, with this component	**Application**
Drop down box next to Application	**ULDatabase1**
Do the following	**Move to next record**

- Click Add Code, and in the dialog that appears, choose **ULProduct**.
- Click OK to finish. Leave the Code Assistant open.

10 Click Close to close the Code Assistant.

You have now completed the application, which displays the product names in the database. You can look at the generated code in the code tab of the MobileBuilder right pane. Next, you compile and run your application.

Lesson 7: Build and run your application

This section describes how to compile and link using the PRC Tools.

❖ **Build and run your application:**

1 Save the application.

From the File menu, choose Save. Accept the default name and click Save.

2 From the MobileBuilder Project menu, choose Build.

When compiling is complete, the Output Panel at the bottom of the MobileBuilder workspace should display a line stating the following:

```
Tutorial.exe – 0 error(s),  0 warning(s)
```

> **Compiler failure**
> This tutorial assumes that you have a working installation of the PRC Tool chain for Palm development. If the build fails, you should check that your PRC Tools installation is correct. If problems persist, consult your MobileBuilder documentation.

3 Start the MobiLink synchronization server so that a connection can be established.

From the Windows Start button, choose Programs➤ Sybase SQL Anywhere 8 ➤MobiLink➤ Synchronization Server Sample.

The MobiLink synchronization server starts, and displays that it is ready to handle requests. In this case we are synchronizing with the UltraLite Sample database, which is serving as both reference and consolidated databases. In general, the reference and consolidated database are separate objects.

4 Start the application.

From the MobileBuilder Project menu, choose Run.

The Palm OS Emulator appears and displays the form you designed.

5 Prepare the emulator for TCP/IP synchronization.

- Right click the emulator, and choose Settings➤Properties from the popup menu.

- Check Redirect NetLib calls to host TCP/IP. This option sets the Emulator to use TCP/IP communications with applications on the current device.
- Leave other items in the dialog unchanged and click OK.

6 Synchronize your application.

- Click the Synchronize button.

  Inspect the MobiLink synchronization server window. It should display a set of messages confirming that Synchronization is complete. You can scroll back through the messages to see that a set of rows were added to the UltraLite database running in the Emulator.

  The current position is left before the first item of the result set, and so no item is yet displayed in the text field.

- Click the Next button.

  The first product name from the database appears in the text field.

You have now built and run a simple UltraLite application using MobileBuilder. The application is still incomplete, however. To see this, click Applications in the Emulator to switch away from your application, and then click App to switch back to it. Although the user interface displays, you will see that the application no longer functions. The next lesson describes how to fix that problem.

You can close MobileBuilder and the MobiLink synchronization server.

Lesson 8: Complete the application

In this lesson you add the application-switching features needed to handle switching away from and back to your application.

To add these features, which are specific to the Palm Computing Platform, you need to write code for both the **AppExit** and **AppEnter** events.

Each time the user switches away from an application, the MobileBuilder **AppExit** event is invoked. You need to add code for this event to close the table. The current position is lost during the switch away from and back to the application, but the data is kept.

❖ To close tables when the user exits the application:

1 Open the Code Assistant.

In the MobileBuilder right-hand pane, right-click the form and choose Code Assistant from the popup menu.

2 In the Code Assistant, make the following settings:

Field	Value
When This Component	**Application**
Drop down box next to **Form1**	Empty
fires This Event	**AppExit**
then, with this component	**Application**
Drop down box next to Application	**ULDatabase1**
Do the following	**Close a table or result set**

3 Click Add Code, and in the dialog that is displayed, choose **ULProduct**.

4 Click OK to add the code. Leave the Code Assistant open.

Each time the user switches back to an application, the MobileBuilder **AppEnter** event is invoked. You need to add code for this event to open the table. The operation is the same whether or not the user is switching to the application for the first time or at a subsequent time.

❖ To open tables when the user enters the application:

1 In the Code Assistant, make the following settings:

Field	Value
When This Component	**Application**
Drop down box next to **Form1**	Empty
fires This Event	**AppEnter**
then, with this component	**Application**
Drop down box next to Application	**ULDatabase1**
Do the following	**Open a table or result set**

2 Click Add Code, and in the dialog that is displayed, choose **ULProduct**.

3 Click OK to add the code. Leave the Code Assistant open.

4 Compile and run your application to test it.

Result sets and tables

This tutorial used a table object to hold the data. You can also use result set objects to hold more complex query result sets. The two objects have different behavior for application switching.

- Tables must be closed using **ULBClose** before exiting the application, and opened using **ULBOpen** when the user enters the application.
- Result sets must *not* be closed before exiting the application. You do not need to take any specific action for result sets when exiting the application. When the user enters the application for the first time, the result set must be opened using **ULBOpen**. When the user enters the application in subsequent times, the result set must be reopened using **ULBReopen**.

  The following code provides an example handler for the **AppEnter** event. The **ULDatabase1LaunchCode** function tests whether the application is being entered for the first time or not. The result set is opened on first entry, but is reopened on subsequent entries.

```
static BOOLEAN OnAppEnter(EVENTTYPE *pEvent, WORD
*pError)
{
   // perform default processing
   DefaultHandler(pEvent, pError);
   switch( ULDatabase1LaunchCode() ){
   case LAUNCH_SUCCESS_FIRST:
      if( !ULBOpen( ResultSet1,
           ULDatabase1connection) ){
          // initialization failed
          DisplayNotice ( "LAUNCH_SUCCESS_FIRST error",
                          "OK", "Cancel" );
           break;
       }
       break;
   case LAUNCH_SUCCESS:
       if( !ULBReopen(ResultSet1,
                    ULDatabase1connection) ){
           DisplayNotice ( "LAUNCH_SUCCESS error",
                           "OK", "Cancel" );
           break;
        }
        break;
   case LAUNCH_FAIL:
       // error
       break;
   }
   // return TRUE to continue event routing {{MB1}}
   return(TRUE);
}
```

The above code illustrates how to use a **DisplayNotice** function for rudimentary debugging.

Restore the sample database

Now that you have completed the tutorial, you should restore the sample database so that it can be used again. You created a backup copy of the UltraLite Sample database at the beginning of this tutorial. You can now replace the modified version of *custdb.db* with the backup copy, overwriting any changes.

❖ **Restore the sample database:**

1. Delete the database file *custdb.db*, and the transaction log file *custdb.log* in the *Samples\UltraLite\CustDB* subdirectory of your SQL Anywhere directory.

   From a command prompt, change to the *Samples\UltraLite\CustDB* subdirectory of your SQL Anywhere directory and enter the following command:

   ```
   dberase -y custdb.db
   ```

2. Copy the *custdb.db* file from your tutorial directory to the *Samples\UltraLite\CustDB* subdirectory of your SQL Anywhere directory.

Your sample database is now restored to its original state.

CHAPTER 4

Developing UltraLite MobileBuilder Applications

About this chapter

This chapter describes how to carry out tasks needed to build UltraLite applications in MobileBuilder. It describes how to use the UltraLite component, how to bind MobileBuilder controls to your data, and provides usage notes on how to use the MobileBuilder controls when building an UltraLite application.

Contents

Introduction

This chapter provides guidelines and notes for developers writing UltraLite applications with MobileBuilder.

What's in this chapter?

The chapter includes the following information:

- An overview of UltraLite application architecture which describes the major pieces that make up an UltraLite MobileBuilder application, and describes some essential functions any application must carry out.

  For information, see "UltraLite MobileBuilder application architecture" on page 47.

- How to use the UltraLite component, which makes an UltraLite database and SQL statements available to your application.

  For information, see "Working with the UltraLite component" on page 50.

- How to bind MobileBuilder controls to the UltraLite component, and how to work with those controls. For example, you may want to bind a list box or a text field to a column of a query result set.

  For information, see "Using the MobileBuilder controls" on page 57.

- How to use the MobileBuilder Code Assistant. The Code Assistant provides an easy way of adding common functionality to your application. For example, you can use the Code Assistant to implement the code for buttons to scroll through the data in your database, or to synchronize your data.

  For information, see "Using the MobileBuilder Code Assistant" on page 60.

- How to write UltraLite code for MobileBuilder. Although the Code Assistant makes many tasks easier, you will still need to write C code for most UltraLite applications. This section describes some considerations specific to UltraLite applications.

  For information, see "Writing UltraLite code for MobileBuilder applications" on page 60.

Before you begin

Some important concepts are introduced in "Tutorial: Build an UltraLite Application Using MobileBuilder" on page 7. This chapter assumes that you have worked through that tutorial.

This chapter also assumes the knowledge described in "About This Manual" on page v.

UltraLite MobileBuilder application architecture

This section describes tasks that must be carried out by any UltraLite MobileBuilder application, and also describes some tasks that are not universally essential, but which are very common.

The pieces of an UltraLite application

Any UltraLite application includes the following kinds of code:

- At the lowest level, an UltraLite runtime library contains platform-specific implementations of functions needed to execute queries and manipulate the data in the database. The library may be a static library that you have to link in to your application, or a DLL that you distribute with your application, depending on the target platform.

  MobileBuilder automatically ensures that the UltraLite runtime library functions are imported into your application.

- An API to manage the database and the data. This set of functions, which calls the runtime library as required, is generated from the UltraLite reference database as needed by MobileBuilder.

  Portions of the API are application-specific, tuned to the queries and data defined in your UltraLite database.

- The application-specific code that you write. The UltraLite-specific parts of your code make calls into the generated API.

  MobileBuilder makes the integration of UltraLite functions into your code easier, by enabling you to bind visual controls to database objects such as result set columns, and by providing the Code Assistant to walk you through adding commonly-used functionality to your application.

The UltraLite features of a MobileBuilder application are implemented by calling functions in an API. These functions in turn call functions in an UltraLite runtime library that must be compiled into your application or distributed with your application as a DLL.

The UltraLite runtime library contains the low-level code for working with the data in your application.

The UltraLite database objects

Any UltraLite application works with a database. The database itself is held in a file that is created automatically by the UltraLite runtime library the first time the application is run.

Before your application views or carries out any manipulation of data, you must open a set of database objects.

1 The first object to open is the database itself. Opening a database prepares the application to connect to the database.

   The functions provided for working with the database all start with **ULDatabase**. You open a database by calling the **ULDatabaseOpen** function. This function takes a single argument, which is a variable that represents the database in the rest of your application.

   If you select **AutoConnect** on the UltraLite component dialog, the **ULDatabase** object is opened for you.

2 The next object to open is a connection between the application and the database. All work done on a database must be done in the context of a connection. Transaction control and synchronization are also handled at the connection level.

   The functions provided for working with connections all start with **ULConnection**. You open a connection by calling the **ULConnectionOpen** function. This function takes two arguments: the database variable and a variable that represents the connection to the rest of your application.

   If you select **AutoConnect** on the UltraLite component dialog, the **ULConnection** object is opened for you.

3 Once you have opened a connection, you can open and then use any of the database objects within the database. A database object may represent a query result set or a base table, and is represented within the MobileBuilder application by an UltraLite Bound Object.

   The functions provided for working with database objects all begin with **ULB**. You open a table or result set by calling the **ULBOpen** function. This function takes two arguments: the connection variable and a variable that represents the table or result set.

   For more information, see "ULBOpen function" on page 77.

The UltraLite component

The UltraLite component provides MobileBuilder with access to UltraLite functionality. All UltraLite features are accessed through the UltraLite component.

The component provides you with access to the UltraLite database objects, enables you to bind these objects to controls on your MobileBuilder forms, and enables you to control synchronization and UltraLite database features.

For more information, see "Working with the UltraLite component" on page 50.

Opening UltraLite database objects

Before you can work with data in an UltraLite database, your application must open the following objects:

1 **The UltraLite database** To open a database, use the **ULDatabaseOpen** function.

2 **A connection to the database** To open a connection, use the **ULConnectionOpen** function.

3 **The table or result set holding the data** To open a table or result set, use the **ULBOpen** function.

If you select Auto Connect in the UltraLite database options of the UltraLite database Data Source properties, the code to open the database and to open the connection is automatically added to your application. If you do not select this option, you must add the code yourself.

You can use the Code Assistant to open the table or result set.

For more information, see "ULBOpen function" on page 77.

Building UltraLite MobileBuilder applications

You compile your UltraLite application from MobileBuilder by choosing Build from the Project menu. MobileBuilder invokes the compiler for the project's target platform, and displays any compilation messages in the Output Pane at the bottom of the MobileBuilder window.

Compilers used with MobileBuilder use a dependency model for compilation so that they only compile those files that have changed since the last compilation.

Working with the UltraLite component

You need to add the UltraLite component to your MobileBuilder project in order to add UltraLite features to your application.

This section describes how to add the UltraLite component to a project. It also describes how to configure the component to work with your UltraLite project and your reference database.

Adding the UltraLite component to your MobileBuilder project

The UltraLite component is a global component. You add it to your project, but unlike buttons and other user interface components, you do not place the control on a form.

❖ **To add the UltraLite component to a MobileBuilder project:**

1 Open a MobileBuilder project.

2 Double-click the ULDatabase component on the UltraLite tab of the Component Palette.

A ULDatabase component named **ULDatabase1** appears under the Global Components folder. The Global Components folder is on the Project tab of the Project Manager Pane.

Before you can use the UltraLite component, you must configure it to work with your reference database.

Configuring the UltraLite component

The UltraLite component has a number of properties that you must configure to suit your project.

❖ **To display the UltraLite component properties:**

1 On the Project tab of the Project Manager Pane, select the ULDatabase global component.

2 Click the Properties tab of the Project Manager Pane.

The UltraLite component properties appear.

The properties fall into the following categories:

- **Filename** The file that is to hold the generated UltraLite database code.
- **Name** The name of the control.
- **ASAVersion** The Adaptive Server Anywhere version that is being used for your reference database.
- **DataSource** Your reference database and also the filename and other characteristics of the UltraLite database.
- **Schema** Specifies a set of columns from tables in the reference database to include in your UltraLite application.
- **ResultSets** Specifies a query. The result set of the query is included in your UltraLite application.
- **Mappings** Maps fields from publications and result sets onto visual controls on your MobileBuilder forms.
- **Synchronization** Specifies how your UltraLite application is to synchronize with a consolidated database.

Configuring the UltraLite component DataSource property

The UltraLite component DataSource property sets the Adaptive Server Anywhere reference database used for your MobileBuilder project, and sets some characteristics of the UltraLite database that is to be generated.

Before you configure the DataSource property, you must have constructed a reference database that holds the tables you wish to include in your UltraLite database.

Reference database

The Data Source tab uses the following items to define your reference database:

- **Data Source** An ODBC data source name for the reference database. You can use Adaptive Server Anywhere ODBC data sources to define connections to a database on your own machine or on other machines, over a client/server connection.
- **User ID** The user ID as which you wish to connect. Some ODBC data source definitions include the user ID, but you need to enter a value in this field anyway, as it is used by the control itself.
- **Password** The password for the user ID. Some ODBC data source definitions include the user ID, but you need to enter a value in this field anyway, as it is used by the control itself.

Once you have chosen values for the required parameters, click Connect to test the connection to the reference database.

Once you have confirmed that the parameters are correct, click OK to save them in the UltraLite component.

You may need to save the connection information before you can configure the Schema, Result Sets, or Control Mappings.

UltraLite database

The Data Source tab uses the following items to define your UltraLite database:

- **Filename** UltraLite databases are stored in a persistent store. The filename for this store depends on the target platform. If you wish to use a name other than the default, enter a value here. If you wish to use the default name, you can leave this field blank.

  The default filename is *ul.udb* on Windows CE, and *ul_<projectname>.udb* on other target systems.

- **Cache Size** When your UltraLite application is working with its database, it uses a cache to hold some data and speed up repeated access to that data. You can configure the size of the cache in this entry. If you wish to use the default, leave this field blank.

  The default cache size for 16-bit architectures is 8 K. The default cache size for 32-bit architectures is 64 K. The minimum cache size is 4 K.

- **Reserve Size** Choose an amount of space to reserve on your target machine for the UltraLite database. If you wish to use the default, leave this field blank.

  By default, no space is reserved for the database (a default value of zero).

- **Auto Connect** If you leave this option checked, then your UltraLite application will connect to the database immediately upon starting.

  The UltraLite component adds code to open and connect to the UltraLite database.

Configuring the UltraLite component Schema property

The Schema tab provides you with a dropdown list of publication names in the reference database. You can select one of these publications to define tables in your UltraLite database, and to define bound objects for mapping to controls.

You can click Edit to change the definition of a publication in the reference database, or to define a new publication.

You must have saved reference database connection information to change some of these values.

For more information, see "Configuring the UltraLite component DataSource property" on page 51.

Publications

You can choose a set of tables from your reference database to include in your UltraLite database. For each table in the set, you can include all or some of the columns. This collection of columns is called a **publication**.

For more information on publications, see "Defining UltraLite tables" on page 123 of the book *UltraLite User's Guide*.

You can use an UltraLite bound object to access the rows of any table included in your UltraLite database. A set of bound object functions allows you to move backwards and forwards through the rows of the table, insert new rows, update rows, and delete rows.

For a full listing of bound object functions, see "ULBoundObject functions" on page 69.

❖ **To create a publication:**

1. In the MobileBuilder Project Manager Pane, open the Properties tab for the UltraLite component.
2. Click the button next to the Schema property. The UltraLite database Setup window appears, open at the Schema tab.
3. Click Edit. The Create Publication dialog appears.
4. Enter a name for the publication, and select the columns you wish to use for the publication.
5. Click OK to dismiss the dialogs and complete the operation.

As an alternative to using publications, you can define your database by creating an UltraLite project with a set of SQL statements. For more information, see "Configuring UltraLite component ResultSets" on page 53.

Configuring UltraLite component ResultSets

Instead of choosing tables from the reference database, you can define queries on your reference database, and the UltraLite generator produces a database containing all the tables needed to support the queries you define.

You must assign a name to each query or other SQL statement that you wish to use in an UltraLite application, and include it in an UltraLite **project**. A project is simply a collection of related statements.

☞ For more information on UltraLite projects, see "Creating an UltraLite project" on page 80 of the book *UltraLite User's Guide*, and "Defining SQL statements for your application" on page 80 of the book *UltraLite User's Guide*.

The Result Sets tab provides you with a dropdown list of existing UltraLite projects in the reference database. You can add projects, change the set of statements in a project, or edit individual statements, by clicking Edit.

If you have problems displaying information in the dialog, make sure that you have saved connection information for the reference database.

☞ For information on saving connection information, see "Configuring the UltraLite component DataSource property" on page 51.

❖ **To create an UltraLite project:**

1. In the MobileBuilder Project Manager Pane, open the Properties tab for the UltraLite component.
2. Click the button next to the ResultSets property. The UltraLite database Setup window appears, open at the Result Sets tab.
3. Click Edit. The Create Project dialog appears.
4. Enter a name for your new project. You can add SQL statements to the project now or at a later time.
5. Click OK to dismiss the dialogs and complete the operation.

❖ **To add a SQL statement to an UltraLite project:**

1. In the MobileBuilder Project Manager Pane, open the Properties tab for the UltraLite component.
2. Click the button next to the ResultSets property. The UltraLite database Setup window appears, open at the Result Sets tab.
3. Choose the project you wish to add SQL statements to, and click Edit. The Create Project dialog appears.
4. Assign a name and add a SQL statement to the project. You should first test your SQL statements using Interactive SQL or another application.
5. Click OK to dismiss the dialogs and complete the operation.

Configuring the UltraLite component Mappings property

On the Mappings tab, you can bind visual controls on the forms that make up your application to sets of values from the result sets or publications in your UltraLite database.

If you have problems displaying information in the dialog, make sure that you have saved connection information for the reference database.

For information on saving connection information, see "Configuring the UltraLite component DataSource property" on page 51.

Configuring the UltraLite component Synchronization property

On the Synchronization tab, you can set values that control the details of the synchronization process.

For more information on the synchronization in UltraLite, see "Adding synchronization" on page 71 of the book *UltraLite User's Guide*.

If you have problems displaying information in the dialog, make sure that you have saved connection information for the reference database.

For more information, see "Configuring the UltraLite component DataSource property" on page 51.

The values on the tab are as follows:

- **User name** The user name used by MobiLink synchronization. It is not a user ID for the database: user IDs are not used for UltraLite databases.
- **Version** The script version for the set of scripts to use to synchronize. You can choose which version to use from a dropdown list.

  For this to work, the synchronization scripts must be present in your reference database. In production, the scripts must be available in your consolidated database. If you have not added scripts to your reference database, you will not get a set of values to choose from, and you must enter a value yourself.
- **Stream** Choose whether you wish to synchronize using communication over a Serial, TCP/IP, or HTTP link. The available options that control the details of the communication depend on the stream you choose.
- **Serial options** If you choose to synchronize over a serial stream, you can set options to control the synchronization in these fields.

- **TCP/IP or HTTP options** If you choose to synchronize over a TCP/IP or HTTP stream, you can set options to control the synchronization in these fields.
- **Security** If you wish to use Certicom transport-layer security, you can do so by checking the Secure option, and entering values in the Options dialog.

Using the MobileBuilder controls

This section describes a number of aspects of working with MobileBuilder controls while developing UltraLite applications. It is not a general guide to using MobileBuilder controls.

Binding controls to UltraLite objects

You can use the Mapping properties of the UltraLite component to associate controls on your forms with columns in UltraLite tables or result sets.

By binding a control to a column in a table or result set, the control displays the values in that column. If the control is a text field or other single-entry field, it displays the value for the current row, and if the control is a list box, it displays the entire column, but with the current row selected.

You can navigate backwards and forwards among the items of the result set using **ULBNext**, **ULBPrevious**, and other UltraLite MobileBuilder functions. MobileBuilder bound controls get updated as you move through the result set. For example, you may place one button on your form that moves forward through the result set, and one that moves backwards.

Using MobileBuilder list boxes

In order to display the contents of a list box, you must carry out an action on the object to which it is bound. For example, if you open a table that has a column bound to a list box, the contents of the column do not automatically appear. Once you move the current position of the cursor to the next row (or any other location) the data appear in the list box.

MobileBuilder list boxes are intended for the display of lists with unique items. If you have duplicate items in a list box, you may find that scrolling behavior is inconsistent.

Configuring data entry controls

When you retrieve information from a data entry control such as a text field, you do so into a C variable in your MobileBuilder program. You can do this in either of the following ways:

- Get the information as a string, and cast it to the appropriate type explicitly.

- Set the **Type** property of the control to the correct data type, and get the information directly into a variable of that type.

❖ **To set the data type of a MobileBuilder control:**

1. Right-click the control and select Properties.
2. In the Project Manager Pane, click the Type entry, and select the appropriate type from the dropdown list.

Getting data from a control

MobileBuilder provides functions for obtaining data from a control. Here is sample code that uses the **FldGetIntegerValue** function into a variable named **prod_id**, and uses this value to set the current record value.

The constant **ULProduct_prod_id** is defined in the file *PRMaps.h*, which is generated during the build process and is held in the MobileBuilder project directory. It identifies the column number for the **prod_id** column of the **ULProduct** table.

```
long     prod_id ;
WORD     error_value;
FIELDTYPE * pField;
WindowHandle hWin;

// get form's window handle
hWin = WinGetActiveWindow();
if( ( pField = FrmGetFieldPtr( hWin, 1 ) ) != NULL ){
    prod_id = FldGetIntegerValue(
              pField, &error_value );
    if( !ULBSetSLong( ULDatabase1ULProduct,
                      ULProduct_prod_id, prod_id ) ) {
        DisplayNotice ( "Error on ULBSetString",
                        "OK", "");
    }
}
```

This step prepares a set of values ready for inserting or updating, but it does not actually update or insert any data into the table or result set.

Updating tables and result sets

Once you have set the values for each of the columns in your result set or table, as described above, you can call a function to insert those rows at the current position, or update the current row.

If you are working with a result set, it must be a single-table result set if you wish to insert or update data.

Here is some sample code that inserts a new row into the **ULDatabase1ULProduct** table, and commits the change:

```
if( !ULBInsert( ULDatabase1ULProduct, 0 ) ){
      // Error
      DisplayNotice ( "Error on Insert", "OK", "");
   }
   if( !ULConnectionCommit( ULDatabase1connection ) ){
      // Error
      DisplayNotice ( "Error on Commit", "OK", "");
   }
```

The **ULBInsert** function takes two arguments: the database object to be updated, and the form that holds the control from which to get values. As you have already set values into the current position using the ULBSet function, you can use a value of 0 to indicate that the data does not come from a form.

For more information, see "ULBInsert function" on page 75 and "ULBUpdate function" on page 82.

Writing UltraLite code for MobileBuilder applications

This section covers a variety of situations and tasks that you will come across during the development of UltraLite applications in MobileBuilder.

Using the MobileBuilder Code Assistant

The MobileBuilder Code Assistant provides an easy way to write code that handles some common events. In particular, you can use the MobileBuilder Code Assistant to add code for the following tasks:

- Synchronizing data.
- Scrolling through result sets and tables.
- Inserting, updating, and deleting rows from tables and result sets.
- Carrying out commits and rollbacks.
- Opening and closing UltraLite database, connection, and tables or result sets.

The Code Assistant provides an easy way of adding simple functionality, but it does not provide ways of carrying out other tasks.

Referencing database objects in the UltraLite API for MobileBuilder

To work on data in an UltraLite database from a MobileBuilder application, you must first open the database, open the connection, and then open the table or result set that you wish to work with.

If you choose the option to autoconnect when configuring the UltraLite component DataSource properties, the database and connection objects are automatically opened for you. You still have to open the table or result set explicitly. You can do this in a straightforward way using the MobileBuilder Code Assistant.

Working with databases and connections

The **ULDatabase** functions make the data in the database object available to your application. You need to call **ULDatabaseOpen** before you can connect to the UltraLite database or carry out any operations on the data.

The **ULDatabaseOpen** function can be called with parameters that define the storage and access parameters for the database (file name, cache size, reserved size).

Once the database is opened, you can open a connection on the database. You do that using the **ULConnectionOpen** function, supplying a reference to the database object, and a set of connection parameters to establish the connection.

Once the connection is established, you can open the result set or table, and use these objects to manipulate the data.

Palm Computing Platform developers

If you are developing an application for the Palm Computing Platform, there are some extra considerations for how to use these objects.

For more information, see "Developing Palm applications in MobileBuilder" on page 63.

Working with tables and result sets

Each table or result set is represented by a variable in your application which you supply as an argument when you open the object. The API for accessing and modifying the rows in the table or query is based on a SQL **cursor**: a pointer to a position in the table or query.

The cursor can have the following positions:

- **Before the first row** This position has value 0. This is the position of the cursor when the table or query opens.
- **On a row** If a table or query has *n* rows, positions 1 to *n* for the cursor correspond to the rows.
- **After the last row** This position has value (*n*+ 1)

You can move through the rows of the object using methods of the object, including **ULBNext** and **ULBPrevious** functions.

Palm Computing Platform developers

If you are developing an application for the Palm Computing Platform, there are some extra considerations for how to use these objects.

For more information, see "Developing Palm applications in MobileBuilder" on page 63.

Writing platform-specific code in MobileBuilder

MobileBuilder provides a way to write applications for multiple platforms. However, you may need to make some features of your application specific to one platform or another. For example, there are differences in architecture between the Palm Computing Platform and Windows CE that require different code to handle, as described in "Developing Palm applications in MobileBuilder" on page 63.

MobileBuilder provides a set of macros to identify platform-specific code. These macros are as follows:

- **PRWINDOWS** Code for Windows operating systems and Windows CE.
- **PRPALM** Code for the Palm Computing Platform.

Example

The following routine shows how to use platform-specific code to handle the startup process in a cross-platform application.

```
static BOOLEAN OnAppEnter( EVENTTYPE *pEvent,
                           WORD *pError)
{
    // perform default processing
    DefaultHandler(pEvent, pError);
#if defined(PRPALM)
    switch ( ULDatabase1LaunchCode() ) {
    case LAUNCH_SUCCESS_FIRST:
        ULBOpen(ULDatabase1ULProduct,
        ULDatabase1connection);
        break;
    case LAUNCH_SUCCESS:
        ULBReopen(ULDatabase1ULProduct,
        ULDatabase1connection);
        break;
    default:
        //
        break;
    }
#elif defined(PRWINDOWS)
      ULBOpen(ULDatabase1ULProduct,
      ULDatabase1connection);
#endif

    // return TRUE to continue event routing {{MB1}}
    return(TRUE);
}
```

Developing Palm applications in MobileBuilder

To develop MobileBuilder applications for the Palm Computing Platform, you must have the GCC PRC-Tools as compiler.

For information on writing Palm-specific code in your application, see "Writing platform-specific code in MobileBuilder" on page 62.

Applications for the Palm Computing Platform must have their code compiled in segments. You must manually enter code section names in the MobileBuilder Palm/Settings dialog. At minimum you must enter the names ULRT1 ... ULRT12, the section names for the UltraLite runtime.

For more information about HotSync, see "Adding TCP/IP, HTTP, or HTTPS synchronization to Palm applications" on page 283 of the book *UltraLite User's Guide*.

MobileBuilder expects the Palm SDK to be installed at *c:\PalmDev*. If you have the Palm SDK installed elsewhere, you must add the include paths in Project/Settings/Folders/Include Folders. For example, if you have installed the Palm SDK in *d:\PalmDev*:

- *d:\PalmDev\sdk-3.1\include*
- *d:\PalmDev\sdk-3.1\include\core*
- *d:\PalmDev\sdk-3.1\include\core\ui*
- *d:\PalmDev\sdk-3.1\include\core\system*
- *d:\PalmDev\sdk-3.1\include\core\hardware*

Compiler issues

When compiling source files that hold more than 32K of code partitioned into multiple segments, the compiler may generate **jsr** instructions unacceptable to the assembler. This happens when a function defined at the bottom of the source file makes an inter-segment call to a function defined at the top of the source file, with at least 32K of code in between. Normally, the offset of the **jsr** instruction is replaced during the relocation stage of the linker, but in this case, the error prevents the compilation from going any further.

The generated file for UltraLite is usually bigger than 32K, and almost every function is in a separate segment.

You can avoid this issue using the `-Wa,-J` switch when compiling. This instructs the assembler to ignore any signed overflow errors.

CHAPTER 5

UltraLite API Reference

About this chapter

This chapter describes the UltraLite API for MobileBuilder.

Contents

Introduction to the UltraLite API

The UltraLite API for MobileBuilder is a set of C functions, including the following:

- **Database functions** These functions are available for all UltraLite applications. These functions are called to initiate and terminate work with a database. Database functions are based on the **ULData** class of the C++ API, and their names start with the string **ULDatabase**.
- **Connection functions** These functions handle connecting to a database and disconnecting, as well as setting some properties of the connection. Synchronization is also initiated using a connection function. Connection functions are based on the **ULConnection** class of the C++ API, and their names start with the string **ULConnection**.
- **ULBoundObject functions** These are generated for each UltraLite application. The names of the functions are based on the names of the queries and tables included in your UltraLite database. Each query or table has a set of functions associated with it, which allows you to move through the rows of the table or query result set, make changes to the data, and so on.

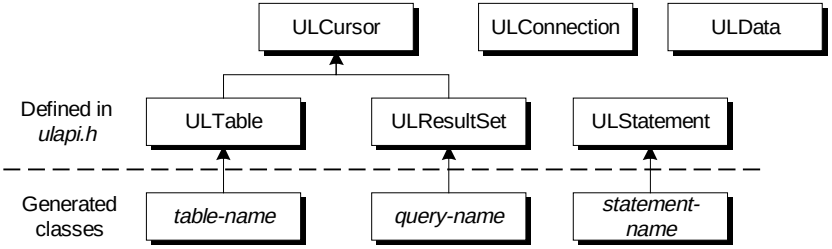
The functions described here are described in the file *ulbindc.h*, in the *h* subdirectory of your SQL Anywhere directory.

C++ API class hierarchy

The UltraLite API for MobileBuilder is a C wrapper around an underlying C++ interface. Although you can use the UltraLite API without understanding the underlying interface, doing so may help to orient you while working with MobileBuilder. You do not need to know C++ to understand the structure of the C++ API.

The C++ interface contains classes, and methods on those classes. The names of the C functions in the UltraLite API for MobileBuilder are based on the underlying class and method name. For example, the C++ API has a class named **ULConnection**, with a method named **Open**. The equivalent call in the UltraLite API for MobileBuilder is a function named **ULConnectionOpen**.

The classes in the underlying C++ API appear in the following diagram:



The classes are described in the following header files:

- **generated-name.hpp** The interface generated for a particular set of statements or tables is defined in the generated *.hpp* file.
- **ulapi.h** The base classes are defined in *ulapi.h*, in the *h* subdirectory of your SQL Anywhere directory.
- **ulglobal.h** You may want to look at *ulglobal.h*, in the *h* subdirectory of your SQL Anywhere installation directory, for some of the data types and other definitions used in *ulapi.h*.

Language elements

The UltraLite API functions and variables are described in terms of a set of UltraLite data types. These data types are described in this section.

UltraLite data types

- **an_SQL_code** A data type for holding SQL error codes.
- **ul_bool** A data type for holding boolean values.
- **ul_char** A data type representing a character. If the operating system uses Unicode, **ul_char** uses two bytes per character. For single-byte character sets, **ul_char** uses a single byte per character.
- **ul_column_num** A data type for holding a number indicating a column of a table or query. The first column in the table or query is number one.
- **ul_fetch_offset** A data type for holding a relative number in a ULCursor object.
- **ul_length** A data type for holding the length of a data type.
- **DECL_DATETIME** A type for holding date and time information in a SQLDATETIME structure, which is defined as follows:

```
typedef struct sqldatetime {
   unsigned short year; /* e.g. 1999 */
   unsigned char month; /* 0-11 */
   unsigned char day_of_week; /* 0-6 0=Sunday */
   unsigned short day_of_year; /* 0-365 */
   unsigned char day; /* 1-31 */
   unsigned char hour; /* 0-23 */
   unsigned char minute; /* 0-59 */
   unsigned char second; /* 0-59 */
   unsigned long microsecond; /* 0-999999 */
} SQLDATETIME;
```

  DECL_DATETIME is also used in embedded SQL programming. Other embedded SQL data types with named DECL_*type* are not needed in C++ API programming.

- **UL_NULL** A constant representing a SQL NULL.

ULBoundObject functions

The **ULBoundObject** functions manipulate data in the UltraLite database.

ULBAfterLast function

Syntax ul_bool **ULBAfterLast**(void **tablePtr*, WORD *formToDraw*)

Description Changes a cursor position to after the last row in a table or result set.

tablePtr The table or result set on which the cursor is defined.

formToDraw Redraw the named form, so that any controls on that form reflect the cursor movement.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULBBeforeFirst function" on page 69
"ULBLast function" on page 75

ULBBeforeFirst function

Syntax ul_bool **ULBBeforeFirst**(void **tablePtr*, WORD *formToDraw*)

Description Changes a cursor position to before the first row in a table or result set.

tablePtr The table or result set on which the cursor is defined.

formToDraw Redraw the named form, so that any controls on that form reflect the cursor movement.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULBAfterLast function" on page 69
"ULBFirst function" on page 73

ULBClose function

Syntax ul_bool **ULBClose**(void **tablePtr*)

Description

Frees resources associated with the table or result set in your application. This function must be called after all processing involving the table or result set is complete, and before the ULConnection and ULData objects are closed.

Any uncommitted operations are rolled back when the **ULBClose** function is called.

tablePtr The table or result set on which the cursor is defined.

Return

true (1) if successful.

false (0) if unsuccessful.

ULBDelete function

Syntax

ul_bool **ULBDelete(** void **tablePtr* **)**

Description

Deletes the current row from a table or result set.

tablePtr The table or result set on which the cursor is defined.

Return

true (1) if successful.

false (0) if unsuccessful. For example, if you attempt to use the function on a SQL statement that represents more than one table.

See also

"ULBInsert function" on page 75
"ULBUpdate function" on page 82

ULBDeleteAllRows function

Prototype

ul_ret_void **DeleteAllRows(**void **tablePtr* **)**

Description

The function deletes all rows in the table.

In some applications, it can be useful to delete all rows from tables before downloading a new set of data into the table. Rows can be deleted from the UltraLite database without being deleted from the consolidated database using the **ULConnection::StartSynchronizationDelete** method.

tablePtr The table or result set on which the cursor is defined.

ULBFind function

Equivalent to the ULBFindNext function.

See "ULBFindNext function" on page 72.

ULBFindFirst function

Prototype ul_bool **ULBFindFirst(** void * *tablePtr*, ul_column_num *ncols* **)**

Description Move forwards through the table from the beginning, looking for a row that exactly matches a value or set of values in the current index.

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure the cursor position is **ULBAfterLast**.

Parameters **tablePtr** The table or result set on which the cursor is defined.

ncols For composite indexes, the number of columns to use in the lookup. For example, if there is a three column index, and you want to lookup a value that matches based on the first column only, you should **Set** the value for the first column, and then supply an *ncols* value of 1.

Returns **true** (1) if successful.

false (0) if unsuccessful.

See also "ULBLookupBackward function" on page 76
"ULBLookupForward function" on page 76

ULBFindLast function

Prototype bool **ULBFindLast(** void * *tablePtr*, ul_column_num *ncols* **)**

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

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure the cursor position is **ULBBeforeFirst**.

Parameters **tablePtr** The table or result set on which the cursor is defined.

ncols For composite indexes, the number of columns to use in the lookup. For example, if there is a three column index, and you want to lookup a value that matches based on the first column only, you should **Set** the value for the first column, and then supply an *ncols* value of 1.

Returns **true** (1) if successful.

false (0) if unsuccessful.

See also

"ULBLookupBackward function" on page 76
"ULBLookupForward function" on page 76

ULBFindNext function

Prototype

bool **ULBFindNext(**void * *tablePtr*, ul_column_num *ncols* **)**

Description

Move forwards through the table from the current position, looking for a row that exactly matches a value or set of values in the current index.

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure, the cursor position is **ULBAfterLast**.

Parameters

tablePtr The table or result set on which the cursor is defined.

ncols For composite indexes, the number of columns to use in the lookup. For example, if there is a three column index, and you want to lookup a value that matches based on the first column only, you should **Set** the value for the first column, and then supply an *ncols* value of 1.

Returns

true (1) if successful.

false (0) if unsuccessful.

See also

"ULBLookupBackward function" on page 76
"ULBLookupForward function" on page 76

ULBFindPrevious function

Prototype

bool **ULBFindPrevious(**void * *tablePtr*, ul_column_num *ncols* **)**

Description

Move backwards through the table from the current position, looking for a row that exactly matches a value or set of values in the current index.

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure the cursor position is **ULBBeforeFirst**.

Parameters

tablePtr The table or result set on which the cursor is defined.

ncols For composite indexes, the number of columns to use in the lookup. For example, if there is a three column index, and you want to lookup a value that matches based on the first column only, you should **Set** the value for the first column, and then supply an *ncols* value of 1.

Returns

true (1) if successful.

false (0) if unsuccessful.

See also

"ULBLookupBackward function" on page 76
"ULBLookupForward function" on page 76

ULBFirst function

Syntax

ul_bool **ULBFirst**(void *_tablePtr_, WORD _formToDraw_)

Description

Changes a cursor position to be on the first row in a table or result set.

tablePtr The table or result set on which the cursor is defined.

formToDraw Redraw the named form, so that any controls on that form reflect the cursor movement.

Return

true (1) if successful.

false (0) if unsuccessful. For example, the function fails if there are no rows.

See also

"ULBBeforeFirst function" on page 69
"ULBLast function" on page 75

ULBGet functions

Syntax

ul_bool *function-name*(void * *tablePtr*, ul_column_num *column-index*, *value-declaration*, ul_bool *isNULL*)

Function-name	value-declaration
ULBGetString	ul_char * *ptr*, ul_length *length*
ULBGetBinary	p_ul_binary *name* , ul_length *length*
ULBGetDateTime	DECL_DATETIME *&date-value*
ULBGetSBig	DECL_BIGINT *&bigint-value*
ULBGetUBig	DECL_UNSIGNED_BIGINT *&bigint-value*
ULBGetSLong	long *&integer-value*
ULBGetULong	unsigned long *&integer-value*
ULBGetBit	unsigned char *&char-value*
ULBGetDouble	double *& double-value*
ULBGetReal	float *& float-value*
ULBGetSShort	short *&short-value*
ULBGetUShort	unsigned short *&short-value*

Description Gets a value from the specified column.

tablePtr The table or result set on which the cursor is defined.

column-index A 2-byte integer. The first column is column 1. The file *PRMaps.h*, which is generated during the build process and is held in the MobileBuilder project directory, defines a set of constants that you can use to make your code more readable. For example:

```
// column identifiers
#define ULProduct_prod_id    1
#define ULProduct_price    2
#define ULProduct_prod_name    3
```

value declaration The arguments required to specify the value depend on the data type. Character and binary data must be mapped into buffers, with the buffer name and length specified in the call. For other data types, a pointer to a variable of the proper type is needed.

isNULL If a value in a column is NULL, **isNull** is set to **true**. In this case, the **value** argument is not meaningful.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULBSet functions" on page 79

ULBGetEmptyFieldIsNull function

Syntax ul_bool **ULBGetEmptyFieldIsNull**(void **tablePtr*)

Description This function returns the value of the **EmptyFieldIsNull** property of the table or result set. If the value is true, then empty text controls are interpreted as NULL when mapping text controls to or from database columns. Otherwise, they are interpreted as empty strings.

tablePtr The table or result set on which the cursor is defined.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULBSetEmptyFieldIsNull function" on page 81

ULBGetOffset function

Syntax ul_fetch_offset **ULBGetOffset**(void *_tablePtr_)

Description Returns the current offset in the table or result set.

tablePtr The table or result set on which the cursor is defined.

Return **true** (1) if successful.

false (0) if unsuccessful.

ULBGetRowCount function

Prototype ul_ul_long **ULBGetRowCount(** void * _tablePtr_ **)**

Description The function returns the number of rows in the table.

tablePtr The table or result set on which the cursor is defined.

Returns The number of rows in the table.

ULBInsert function

Syntax ul_bool **ULBInsert**(void *_tablePtr_ , WORD _formToGetFrom_)

Description Inserts a row in a table or single-table result set.

tablePtr The table or result set on which the cursor is defined.

formToGetFrom The form that holds the data you are inserting. If your new values have been set using **ULBSet** functions, supply a value of 0.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULBDelete function" on page 70
"ULBUpdate function" on page 82

ULBLast function

Syntax ul_bool **ULBLast**(void *_tablePtr_, WORD _formToDraw_)

Description Changes a cursor position to be on the last row in a table or result set.

tablePtr The table or result set on which the cursor is defined.

formToDraw Redraw the named form, so that any controls on that form reflect the cursor movement.

Return

true (1) if successful.

false (0) if unsuccessful.

See also

"ULBAfterLast function" on page 69
"ULBFirst function" on page 73

ULBLookupBackward function

Syntax

ul_bool **ULBLookupBackward**(void * *tablePtr*, ul_column_num *ncols*)

Description

Move backward through a table or result set, looking for the first row that matches a value or set of values in the current index.

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches the index value. On failure, the cursor position is before the first row.

tablePtr The table or result set on which the cursor is defined.

ncols For composite indexes, the number of columns to use in the lookup. For example, if there is a three column index, and you want to lookup a value that matches based on the first column only, you should set the value for the first column, and then supply an *ncols* value of 1.

Return

true (1) if successful.

false (0) if unsuccessful.

See also

"ULBLookupForward function" on page 76

ULBLookupForward function

Syntax

ul_bool **ULBLookupForward**(void * *tablePtr*, ul_column_num *ncols*)

Description

Move forward through a table or result set, looking for the first row that matches a value or set of values in the current index.

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches the index value. On failure, the cursor position is after the last row.

tablePtr The table or result set on which the cursor is defined.

ncols For composite indexes, the number of columns to use in the lookup. For example, if there is a three column index, and you want to lookup a value that matches based on the first column only, you should set the value for the first column, and then supply an *ncols* value of 1.

Return

true (1) if successful.

false (0) if unsuccessful.

See also

"ULBLookupBackward function" on page 76

ULBNext function

Syntax

ul_bool **ULBNext**(void **tablePtr*, WORD *formToDraw*)

Description

Moves a cursor one position forward in a table or result set.

tablePtr The table or result set on which the cursor is defined.

formToDraw Redraw the named form, so that any controls on that form reflect the cursor movement.

Return

true (1) if successful.

false (0) if unsuccessful.

See also

"ULBPrevious function" on page 78
"ULBRelative function" on page 78

ULBOpen function

Syntax

ul_bool **ULBOpen**(void * *tablePtr*, void * *connPtr*)

Description

Opens a cursor on the specified connection. If the object is a result set with parameters, you must set the parameters before opening the result set using the **ULBSetParameter** functions.

tablePtr The table or result set on which the cursor is defined.

connPtr The connection.

Return

true (1) if successful.

false (0) if unsuccessful.

See also

"ULBClose function" on page 69

ULBPrevious function

Syntax ul_bool **ULBPrevious**(void **tablePtr*, WORD *formToDraw*)

Description Moves a cursor one position backward in a table or result set.

tablePtr The table or result set on which the cursor is defined.

formToDraw Redraw the named form, so that any controls on that form reflect the cursor movement.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULBNext function" on page 77
"ULBRelative function" on page 78

ULBRefresh function

Syntax ul_bool **ULBRefresh(** void **tablePtr*, WORD *formToDraw* **)**

Description This function forces a form to be redrawn, and changes in a bound object to be reflected in the display.

tablePtr The table or result set.

formToDraw The form to be refreshed.

Return **true** (1) if successful.

false (0) if unsuccessful.

ULBRelative function

Syntax ul_bool **ULBRelative**(void **tablePtr*,
ul_fetch_offset *offset* , WORD *formToDraw*)

Description Moves the cursor position relative to the current position. If the row does not exist, the method returns false, and the cursor is left at **AfterLast()** if *offset* is positive, and **BeforeFirst()** if *offset* is negative.

tablePtr The table or result set on which the cursor is defined.

offset The number of rows to move. Negative values correspond to moving backwards.

formToDraw Redraw the named form, so that any controls on that form reflect the cursor movement.

Return

true (1) if the row exists.

false (0) if the row does not exist.

See also

"ULBNext function" on page 77
"ULBPrevious function" on page 78

ULBReopen function

Syntax

ul_bool **ULBReopen**(void *_tablePtr_, void * _connPtr_)

Description

This method is available for the Palm Computing Platform only. The database must be reopened for this call to succeed.

When developing Palm applications, you should never close the connection object. Instead, you should call **Reopen** when the user switches to the UltraLite application. The method prepares the data in use by the database object for use by the application.

tablePtr The table or result set on which the cursor is defined.

connPtr The connection on which the cursor is defined.

Return

true (1) if successful.

false (0) if unsuccessful.

See also

"ULConnectionOpen function" on page 89

ULBSet functions

Syntax

ul_bool *function-name*(void * *tablePtr*, ul_column_num *column-index*, *value-declaration*)

function-name	Value-declaration
ULBSetString	Ul_char * *ptr*, ul_length *length*
ULBSetBinary	P_ul_binary *name* , ul_length *length*
ULBSetDateTime	DECL_DATETIME *date-value*
ULBSetSBig	DECL_BIGINT *bigint-value*
ULBSetUBig	DECL_UNSIGNED_BIGINT *bigint-value*
ULBSetSLong	Long *&integer-value*
ULBSetULong	Unsigned long *&integer-value*

function-name	Value-declaration
ULBSetBit	Unsigned char *&char-value*
ULBSetDouble	Double *double-value*
ULBSetReal	Float *float-value*
ULBSetSShort	Short *short-value*
ULBSetUShort	Unsigned short *short-value*

Description

Gets a value from the specified column.

tablePtr The table or result set on which the cursor is defined.

column-index A 2-byte integer. The first column is column 1. The file *PRMaps.h*, which is generated during the build process and is held in the MobileBuilder project directory, defines a set of constants that you can use to make your code more readable. For example:

```
// column identifiers
#define ULProduct_prod_id    1
#define ULProduct_price    2
#define ULProduct_prod_name    3
```

value declaration The arguments required to specify the value depend on the data type. Character and binary data must be mapped into buffers, with the buffer name and length specified in the call. For other data types, a pointer to a variable of the proper type is needed.

The BIGINT data type is not supported on DOS

Return

true (1) if successful.

false (0) if unsuccessful.

See also

"ULBGet functions" on page 73

ULBSetColumnNull function

Syntax

int ULBSetColumNull(void **tablePtr*, ul_column_num *column-index*)

Description

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

tablePtr The table or result set on which the cursor is defined.

column-index The number of the column. The first column in the table has a value of one.

Return **true** (1) if successful.

false (0) if unsuccessful.

ULBSetEmptyFieldIsNull function

Syntax void **ULBGetEmptyFieldIsNull**(void **tablePtr* , ul_bool *value*)

Description This function sets the value of the **EmptyFieldIsNull** property of the table or result set. If the value is true, then empty text controls are interpreted as NULL when mapping text controls to or from database columns. Otherwise, they are interpreted as empty strings.

tablePtr The table or result set on which the cursor is defined.

value True or false.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULBGetEmptyFieldIsNull function" on page 74

ULBSetParameter functions

Syntax ul_bool *function-name***(** void * *tablePtr*, int *argnum*,
type * *value-reference* **)**

function-name	Type
ULBSetParameterSLong	Long
ULBSetParameterBinary	ul_binary
ULBSetParameterBit	Unsigned char
ULBSetParameterString	ul_char
ULBSetParameterDouble	Double
ULBSetParameterReal	Float
ULBSetParameterSShort	Short

function-name	Type
ULBSetParameterULong	Unsigned long
ULBSetParameterUShort	Unsigned short
ULBSetParameterDateTime	DECL_DATETIME
ULBSetParameterSBig	DECL_BIGINT
ULBSetParameterUBig	DECL_UNSIGNED_BIGINT

Description

A separate function is provided for each data type, with names as listed in the table.

Some result sets are defined by queries that have parameters. You must set the value of the parameter before opening the generated result set object.

tablePtr The table or result set for which the parameter is to be set.

argnum An identifier for the argument to be set. The first argument is 1, the second 2, and so on.

value-reference A pointer to the parameter value.

Return

true (1) if successful.

false (0) if unsuccessful. If you supply a parameter of the wrong data type, the method fails.

Example

The following query has a parameter, indicated by a question mark:

```
SELECT * FROM ULProduct WHERE price > ?
```

If this query is opened as a bound object named **query1**, you can set the value of this parameter before opening it as follows:

```
min_price = 23;
ULBSetParameterSLong( &query1, 1, &min_price );
```

ULBUpdate function

Syntax

ul_bool **ULBUpdate**(void **tablePtr*, WORD *formToGetFrom*)

Description

Updates a row in a table or single-table result set.

tablePtr The table or result set on which the cursor is defined.

formToGetFrom The form that holds the data you are inserting. If your new values have been set using **ULBSet** functions, supply a value of 0.

Return

true (1) if successful.

false (0) if unsuccessful.

See also

"ULBDelete function" on page 70
"ULBInsert function" on page 75

ULConnection functions

Function

The ULConnection functions handle tasks associated with an individual connection to the database. This includes transaction and synchronization control.

ULConnectionClose function

Syntax

ul_bool **ULConnectionClose** (void * *connPtr*)

Description

Disconnects your application from the database, and frees resources associated with the connection. Once you have closed the connection, your application can no longer manipulate data.

Closing a connection rolls back any outstanding changes.

If you select the AutoConnect option when setting up your UltraLite database DataSource property, you do not need to explicitly close your connection.

Return

true (1) if successful.

false (0) if unsuccessful.

Example

The following example closes a ULConnection object:

```
ULConnectionClose( &conn );
```

See also

"ULConnectionOpen function" on page 89

ULConnectionCommit function

Syntax

ul_bool **ULConnectionCommit**(void * *connPtr*)

Description

Commits outstanding changes to the database.

Return

true (1) if successful.

false (0) if unsuccessful.

See also

"ULConnectionRollback function" on page 90

ULConnectionCountUploadRows function

Prototype

ul_u_long **ULConnectionCountUploadRows(**
void * *connPtr* ,
ul_publication_mask *mask*,
ul_u_long *threshold* **)**

Description

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

You can use this function to determine if a synchronization is needed.

Parameters

publication-mask A set of publications to check. A value of 0 corresponds to the entire database. The set is supplied as a mask. For example, the following mask corresponds to publications *PUB1* and *PUB2*.:

```
UL_PUB_PUB1 | UL_PUB_PUB2
```

For more information on publication masks, see "publication synchronization parameter" on page 386 of the book *UltraLite User's Guide*.

threshold A value that determines the maximum number of rows to count, and so limits the amount of time taken by the call. A value of 0 corresponds to no limit. A value of 1 determines if any rows need to be synchronized.

Returns

The number of rows to be uploaded.

ULConnectionGetLastIdentity function

Prototype

ul_u_big **ULConnectionGetLastIdentity(** void * *connPtr* **)**

Description

Returns the most recent identity value used. This function is equivalent to the following SQL statement:

```
SELECT @@identity
```

The function is particularly useful in the context of global autoincrement columns.

Returns

The last identity value.

ULConnectionGetSQLCode function

Syntax

an_SQL_code ULConnectionGetSQLCode(void * *connPtr*)

Description

Obtains the SQLCODE value for the most recent statement on the named connection

Return A SQLCODE value. For more information, see "Error messages indexed by Adaptive Server Anywhere SQLCODE" on page 2 of the book *ASA Errors Manual*.

Example The following code illustrates how to use **ULConnectionGetSQLCode**.

```
ULConnectionInitSynchInfo(
       ULDatabase1Connection, &synch_info );
ULConnectionSynchronize(
       ULDatabase1Connection, &synch_info );
```

See also "ULConnectionLastCodeOK function" on page 87
"ULConnectionLastFetchOK function" on page 88

ULConnectionGlobalAutoincUsage function

Prototype ul_u_short **ULConnectionGlobalAutoIncUsage(** void * *connPtr* **)**

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

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

Returns The percent usage of the available global autoincrement values.

ULConnectionGrantConnectTo function

Prototype ul_bool **ULConnectionGrantConnectTo(**
void * *connPtr*, ul_char * *userid*, ul_char * *password* **)**

Parameters

userid Character array holding the user ID. The maximum length is 16 characters.

password Character array holding the password for *userid*. The maximum length is 16 characters.

Description Grant access to an UltraLite database for a user ID with a specified password. If an existing user ID is specified, this function updates the password for the user.

ULConnectionInitSynchInfo function

Syntax ul_bool **ULConnectionInitSynchInfo**(void * *connPtr*, ul_synch_info * *synch_info*)

Description Initializes the *synch_info* structure used for synchronization.

Return **true** (1) if successful.

false (0) if unsuccessful.

Example The following code illustrates where **ULConnectionInitSynchInfo** function is used in the sequence of calls that synchronize data in a UltraLite application.

```
ULConnectionInitSynchInfo(
       ULDatabase1Connection, &synch_info );
ULConnectionSynchronize(
       ULDatabase1Connection, &synch_info );
```

See also "ULConnectionSynchronize function" on page 91

ULConnectionIsOpen function

Syntax **ul_bool ULConnectionIsOpen** (void * *connPtr*)

Description Checks whether the database connection is currently open.

Return **true** (1) if the connection is open.

false (0) if the connection is not open.

See also "ULConnectionOpen function" on page 89

ULConnectionLastCodeOK function

Syntax **ul_bool ULConnectionLastCodeOK** (void * *connPtr*)

Description Checks whether the most recent SQLCODE returned was zero (**true**) or non-zero (**false**).

This method provides a convenient way of checking for the success (SQLCODE zero) or potential failure (non-zero) of operations. You can use ULConnectionGetSQLCode to obtain the numerical value.

Return **true** (1) if the previous SQLCode was zero.

false (0) if the previous SQLCode was non-zero.

See also "ULConnectionGetSQLCode function" on page 85

ULConnectionGetLastDownloadTime function

Prototype

bool **ULConnectionGetLastDownloadTime(**
void * *connPtr*
ul_publication_mask *mask*,
DECL_DATETIME **value* **)**

Description

Provides error checking capabilities by checking the SQLCODE value for the success or failure of a database operation. The SQLCODE is the standard Adaptive Server Anywhere code.

Parameters

publication-mask A set of publications for which the last download time is retrieved. A value of 0 corresponds to the entire database. The set is supplied as a mask. For example, the following mask corresponds to publications *PUB1* and *PUB2*.:

```
UL_PUB_PUB1 | UL_PUB_PUB2
```

For more information on publication masks, see "publication synchronization parameter" on page 386 of the book *UltraLite User's Guide*.

value A pointer to the DECL_DATETIME structure to be populated.

A value of **January 1, 1990** indicates that the publication has yet to be synchronized.

Returns

- **true** Indicates that *value* is successfully populated by the last download time of the publication specified by *publication-mask*.
- **false** Indicates that *publication-mask* specifies more than one publication or that the publication is undefined. If the return value is false, the contents of *value* are not meaningful.

ULConnectionLastFetchOK function

Syntax

ul_bool **ULConnectionLastFetchOK(** void * *connPtr* **)**

Description

Provides a convenient way of checking that the most recent fetch of a row succeeded (**true**) or failed (**false**).

Return

true (1) if successful.

false (0) if unsuccessful.

See also

"ULBAfterLast function" on page 69
"ULBFirst function" on page 73
"ULConnectionGetSQLCode function" on page 85

ULConnectionOpen function

Syntax

ul_bool ULConnectionOpen (void * *connPtr*, void * *dbPtr*,
ul_char * *userid*, ul_char * *password*, ul_char * *name*)

Description

Open a connection to a database. The database referenced by *dbPtr* must be open for this call to succeed.

connPtr An identifier for the current connection. The default name for this variable if you use the UltraLite database control to open your connection is **ULDatabase1connection**, where **ULDatabase1** is the default name of the UltraLite database component.

dbPtr A pointer to the database on which the connection is made. This argument is set when calling ULDatabaseOpen.

userid The user ID argument is a placeholder reserved for possible future use. It is ignored.

password The password argument is a placeholder reserved for possible future use. It is ignored.

name An optional name for the connection. This is needed only if you have multiple connections from a single application to the same database.

Return

true (1) if successful.

false (0) if unsuccessful.

See also

"ULConnectionClose function" on page 84

ULConnectionReopen function

Syntax

ul_bool ULConnectionReopen (void * *connPtr*, void * *dbPtr*,
ul_char **name*)

Description

This method is available for the Palm Computing Platform only. The database must be reopened for this call to succeed.

When developing Palm applications, you should never close the connection object. Instead, you should call **ULConnectionReopen** when the user switches to the UltraLite application. The method prepares the data in use by the database for use by the application.

connPtr A pointer to the connection.

db A pointer to the **ULData** object on which the connection is made. This argument is usually the address of the **ULData** object opened prior to reopening the connection.

name An optional name for the connection. This is needed only if you have multiple connections from a single application to the same database.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULConnectionOpen function" on page 89

ULConnectionRevokeConnectFrom function

Prototype bool **ULConnectionRevokeConnectFrom(** ul_char * *userid* **)**

Description Revoke access from an UltraLite database for a user ID.

Parameters **userid** Character array holding the user ID to be excluded from database access. The maximum length is 16 characters.

ULConnectionRollback function

Syntax **ul_bool ULConnectionRollback**(void * *connPtr*)

Description Rolls back outstanding changes to the database.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULConnectionCommit function" on page 84

ULConnectionSetDatabaseID function

Prototype bool **ULConnectionSetDatabaseID(** void * *connPtr*, ul_u_long *value* **)**

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

Parameters **value** The value to use for generating global autoincrement values.

Returns **true** (1) if successful.

false (0) if unsuccessful.

ULConnectionStartSynchronizationDelete function

Prototype ul_bool **ULConnectionStartSynchronizationDelete(** void * *connPtr* **)**

Description Once this function is called, all delete operations are again synchronized.

Returns **true** (1) if successful.

false (0) if unsuccessful.

See also "ULConnectionStopSynchronizationDelete function" on page 91
"ULBDeleteAllRows function" on page 70

ULConnectionStopSynchronizationDelete function

Prototype ul_bool **ULConnectionStopSynchronizationDelete(**void * *connPtr* **)**

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

Returns **true** (1) if successful.

false (0) if unsuccessful.

See also "ULConnectionStartSynchronizationDelete function" on page 90
"ULBDeleteAllRows function" on page 70

ULConnectionSynchronize function

Syntax **ul_bool ULConnectionSynchronize (** void * *connPtr*,
ul_synch_info * *info* **)**

Description Synchronizes an UltraLite connection.

connPtr The identifier used when the connection was opened.

synch_info The structure holding the synchronization information. This structure is set by calling **ULConnectionInitSynchInfo**.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULConnectionInitSynchInfo function" on page 86

ULDatabase functions

Function Represents an UltraLite database.

Description The ULDatabase functions make an UltraLite database available to your application. They provide ways to open and close a database, and to check whether a database is open.

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

For its position in the API hierarchy, see "Introduction to the UltraLite API" on page 66.

ULDatabaseClose function

Syntax **bool ULDatabaseClose** (void * *dbPtr*)

Description Frees resources associated with a database, before you terminate your application. Once you have closed the database, you cannot execute any other operations on that database without reopening.

You should not close a database object in a Palm Computing Platform application. Instead, use the **Reopen** method when the application is reactivated.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULDatabaseOpen function" on page 93

ULDatabaseIsOpen function

Syntax **bool ULDatabaseIsOpen** (void * *dbPtr*)

Description Checks whether the database is currently open.

Return **true** (1) if the ULData object is open.

false (0) if the ULData object is not open.

See also "ULDatabaseOpen function" on page 93

ULDatabaseOpen function

Syntax **ul_bool ULDatabaseOpen** (void * *dbPtr*)

Description Prepares your application to work with a database. You must open the database before carrying out any other operations on the database.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULDatabaseClose function" on page 92

ULDatabaseOpenParms function

Syntax **ul_bool ULDatabaseOpenParms** (void * *dbPtr* , ul_char * *parms*)

Description Prepares your application to work with a database. You must open the database before carrying out any other operations on the database.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULDatabaseClose function" on page 92

ULDatabaseReopen function

Syntax **bool ULDatabaseReopen** (void * *dbPtr*)

Description This method is available for the Palm Computing Platform only.

When developing Palm applications, you should never close the database object. Instead, you should call **ULDatabaseReopen** when the user switches to the UltraLite application. The method prepares the data in use by the database object for use by the application.

Return **true** (1) if successful.

false (0) if unsuccessful.

See also "ULDatabaseOpen function" on page 93

Index

I

L

M

N

P

R

S

T

U