UltraLite™ for MobileVB User's Guide

Part number: 36292-01-0900-01

Last modified: June 2003

Contents

About This Manual **v**
SQL Anywhere Studio documentation vi
Documentation conventions ix
The CustDB sample database xi
Finding out more and providing feedback xii

1 Introduction **1**
System requirements and supported platforms 2
UltraLite for MobileVB architecture 4

2 Tutorial: An UltraLite for MobileVB Application for Palm OS **7**
Introduction 8
Lesson 1: Create a project architecture 9
Lesson 2: Create a form interface 11
Lesson 3: Write connection, synchronization, and table code 13
Lesson 4: Deploy the application to a device 21
Summary 22

3 Tutorial: An UltraLite Application for PocketPC **23**
Introduction 24
Lesson 1: Create a project architecture 25
Lesson 2: Create a form interface 27
Lesson 3: Write the sample code 29
Lesson 4: Deploy to a device 37
Summary 38

4 Tutorial: Using Dynamic SQL in an UltraLite Application for PocketPC **39**
Introduction 40
Lesson 1: Create a project architecture 41
Lesson 2: Create a form interface 43
Lesson 3: Write connection, synchronization, and table code 44
Lesson 4: Deploy the application to a device 50
Summary 51

5 Understanding UltraLite for MobileVB Development **53**
Connecting to the UltraLite database 54
Accessing data using dynamic SQL 58
Accessing data using the table-based API 63
Transaction processing in UltraLite 69

Accessing schema information 70
Error handling 71
Synchronization 72
Component samples, demonstrations and code fragments 74
Maintaining database state on Palm OS 75

6 UltraLite for MobileVB API Reference 79
ULAuthStatusCode 81
ULColumn class 82
ULColumnSchema class 88
ULConnection class 89
ULConnectionParms class 97
ULDatabaseManager class 100
ULDatabaseSchema class 106
ULIndexSchema class 109
ULPreparedStatement class 111
ULPublicationSchema class 116
ULResultSet class 117
ULResultSetSchema class 123
ULSQLCode enumeration 124
ULSQLType enumeration 128
ULStreamErrorCode enumeration 129
ULStreamErrorContext enumeration 132
ULStreamErrorID enumeration 133
ULStreamType enumeration 134
ULSyncParms class 135
ULSyncResult class 138
ULSyncState enumeration 139
ULTable class 140
ULTableSchema class 149

Index 151

About This Manual

Subject	This manual describes UltraLite for MobileVB, which is part of the UltraLite Component Suite. With UltraLite for MobileVB you can develop and deploy database applications to handheld, mobile, or embedded devices, including devices running the Palm Computing Platform and Windows CE.
Audience	This manual is intended for MobileVB application developers who wish to take advantage of the performance, resource efficiency, robustness, and security of an UltraLite relational database for data storage and synchronization.

SQL Anywhere Studio documentation

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

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

- **Introducing SQL Anywhere Studio** This book provides an overview of the SQL Anywhere Studio database management and synchronization technologies. It includes tutorials to introduce you to each of the pieces that make up SQL Anywhere Studio.
- **What's New in SQL Anywhere Studio** This book is for users of previous versions of the software. It lists new features in this and previous releases of the product and describes upgrade procedures.
- **Adaptive Server Anywhere 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 and database servers.
- **Adaptive Server Anywhere SQL User's Guide** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- **Adaptive Server Anywhere SQL Reference Manual** This book provides a complete reference for the SQL language used by Adaptive Server Anywhere. It also describes the Adaptive Server Anywhere system tables and procedures.
- **Adaptive Server Anywhere Programming Guide** This book describes how to build and deploy database applications using the C, C++, and Java programming languages. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by those tools. It also describes the Adaptive Server Anywhere ADO.NET data provider.

- **Adaptive Server Anywhere Error Messages** This book provides a complete listing of Adaptive Server Anywhere error messages together with diagnostic information.
- **SQL Anywhere Studio Security Guide** This book provides information about security features in Adaptive Server Anywhere databases. Adaptive Server Anywhere 7.0 was awarded a TCSEC (Trusted Computer System Evaluation Criteria) C2 security rating from the U.S. Government. This book may be of interest to those who wish to run the current version of Adaptive Server Anywhere in a manner equivalent to the C2-certified environment.
- **MobiLink Synchronization User's Guide** This book describes how to use the MobiLink data synchronization system for mobile computing, which enables sharing of data between a single Oracle, Sybase, Microsoft or IBM database and many Adaptive Server Anywhere or UltraLite databases.
- **MobiLink Synchronization Reference** This book is a reference guide to MobiLink command line options, synchronization scripts, SQL statements, stored procedures, utilities, system tables, and error messages.
- **iAnywhere Solutions ODBC Drivers** This book describes how to set up ODBC drivers to access consolidated databases other than Adaptive Server Anywhere from the MobiLink synchronization server and from Adaptive Server Anywhere remote data access.
- **SQL Remote User's Guide** This book describes all aspects of the SQL Remote data replication system for mobile computing, which enables sharing of data between a single Adaptive Server Anywhere or Adaptive Server Enterprise database and many Adaptive Server Anywhere databases using an indirect link such as e-mail or file transfer.
- **SQL Anywhere Studio Help** This book includes the context-sensitive help for Sybase Central, Interactive SQL, and other graphical tools. It is not included in the printed documentation set.
- **UltraLite Database User's Guide** This book is intended for all UltraLite developers. It introduces the UltraLite database system and provides information common to all UltraLite programming interfaces.
- **UltraLite Interface Guides** A separate book is provided for each UltraLite programming interface. Some of these interfaces are provided as UltraLite components for rapid application development, and others are provided as static interfaces for C, C++, and Java development.

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

Documentation formats

SQL Anywhere Studio provides documentation in the following formats:

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

  To access the online documentation on Windows operating systems, choose Start ➤ Programs ➤ SQL Anywhere 9 ➤ Online Books. You can navigate the online documentation using the HTML Help table of contents, index, and search facility in the left pane, as well as using the links and menus in the right pane.

  To access the online documentation on UNIX operating systems, see the HTML documentation under your SQL Anywhere installation.

- **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 complete set of books is available from Sybase sales or from eShop, the Sybase online store. You can access eShop by clicking How to Buy ➤ eShop at *http://www.ianywhere.com*.

Documentation conventions

This section lists the typographic and graphical conventions used in this documentation.

Syntax conventions

The following conventions are used in the SQL syntax descriptions:

- **Keywords** All SQL keywords appear in upper case, like the words ALTER TABLE in the following example:

  ALTER TABLE [*owner*.]*table-name*

- **Placeholders** Items that must be replaced with appropriate identifiers or expressions are shown like the words *owner* and *table-name* in the following example:

  ALTER TABLE [*owner*.]*table-name*

- **Repeating items** Lists of repeating items are shown with an element of the list followed by an ellipsis (three dots), like *column-constraint* in the following example:

  ADD *column-definition* [*column-constraint*, . . .]

  One or more list elements are allowed. In this example, if more than one is specified, they must be separated by commas.

- **Optional portions** Optional portions of a statement are enclosed by square brackets.

  RELEASE SAVEPOINT [*savepoint-name*]

  These square brackets indicate that the *savepoint-name* is optional. The square brackets should not be typed.

- **Options** When none or only one of a list of items can be chosen, vertical bars separate the items and the list is enclosed in square brackets.

  [**ASC** | **DESC**]

  For example, you can choose one of ASC, DESC, or neither. The square brackets should not be typed.

- **Alternatives** When precisely one of the options must be chosen, the alternatives are enclosed in curly braces and a bar is used to separate the options.

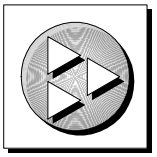
  [**QUOTES** { **ON** | **OFF** }]

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

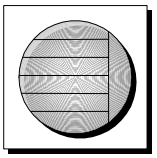
Graphic icons

The following icons are used in this documentation.

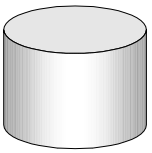
- A client application.



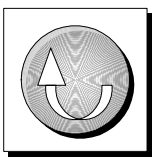
- A database server, such as Sybase Adaptive Server Anywhere.



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



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



- A programming interface.

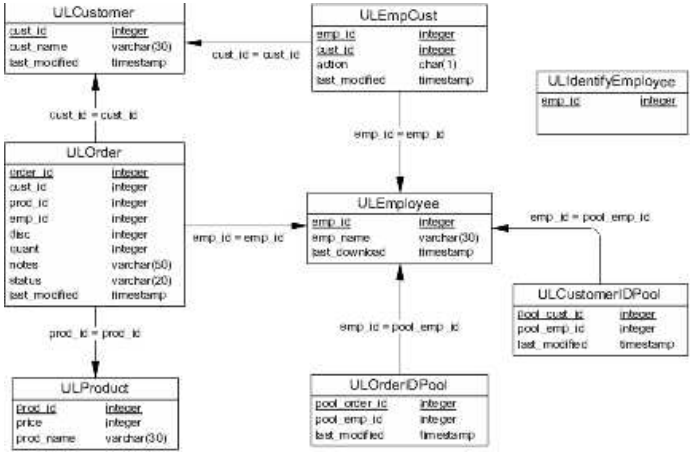
The CustDB sample database

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

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

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

The following 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

Introduction

About this chapter

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

☞ For hands-on tutorials introducing UltraLite for MobileVB, see the following:

- "Tutorial: An UltraLite for MobileVB Application for Palm OS" on page 7
- "Tutorial: An UltraLite Application for PocketPC" on page 23
- "Tutorial: Using Dynamic SQL in an UltraLite Application for PocketPC" on page 39

Contents

Topic:	page
System requirements and supported platforms	2
UltraLite for MobileVB architecture	4

System requirements and supported platforms

Development platforms

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

- Microsoft Windows NT/2000/XP.
- Microsoft Visual Basic 6.

  You must install a service pack that meets the requirements for the version of AppForge MobileVB that you are using. For more information, see the AppForge web site. It is recommended that you install at least service pack 5.

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

- AppForge 2.11 or AppForge MobileVB Version 3.x.

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

☞ For more information, see “UltraLite host platforms” [*Introducing SQL Anywhere Studio*, page 126].

Target platforms

UltraLite supports the following target platforms:

- Windows CE 3.0 and higher, with Pocket PC on the ARM and MIPS processors.
- Palm OS version 3.5 and higher. For more information about Palm-specific memory requirements, see “UltraLite target platforms” [*Introducing SQL Anywhere Studio*, page 136].

☞ For more information, see “UltraLite target platforms” [*Introducing SQL Anywhere Studio*, page 136].

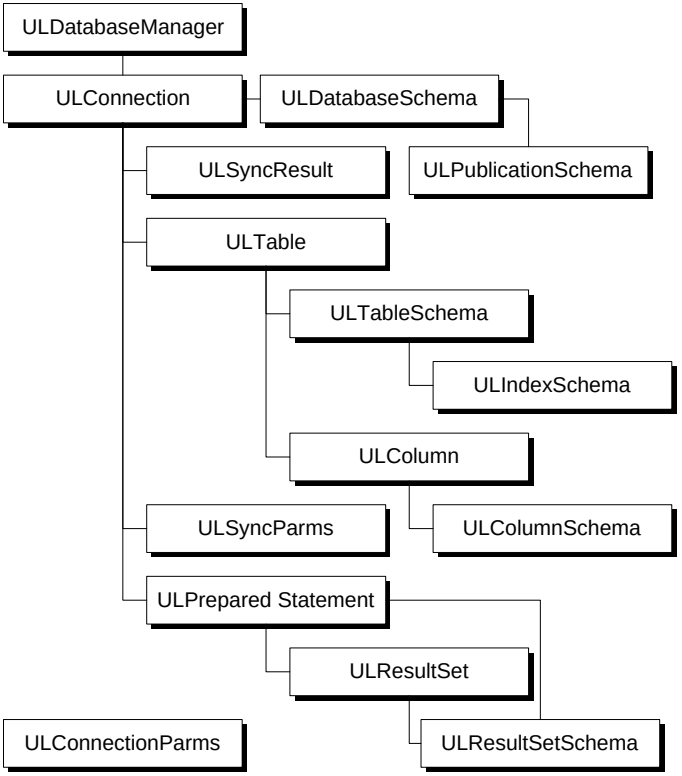
SQL Anywhere Studio

You can use SQL Anywhere Studio to add the following capabilities to your applications:

- **Synchronization** SQL Anywhere users can synchronize the data in UltraLite applications with a central database.
- **Reference database** SQL Anywhere users can use the *ulinit* utility to model an UltraLite schema file after an Adaptive Server Anywhere database.

UltraLite for MobileVB architecture

UltraLite for MobileVB provides a database engine for the Palm Computing Platform and Windows CE. It provides a MobileVB control that exposes a set of objects for data manipulation using an UltraLite database.



Notably, there is a set of high-level objects you should know about:

- **ULDatabaseManager** allows you to open connections and set an active listener. The ULDatabaseManager is the starting point for your MobileVB application because it is through this class that you first open a connection to database.

  ☞ For more information about the ULDatabaseManager class, see "ULDatabaseManager class" on page 100.

- **ULConnectionParms** UltraLite for MobileVB gives you a convenient ULConnectionParms object interface, so you can add connection parameters directly to an easy to use Visual Basic property sheet.

  ☞ For more information about ULConnectionParms, see "ConnectionParms" on page 97.

- **ULConnection** represents a database connection, and governs transactions.

  ☞ For more information about ULConnection, see “ULConnection class” on page 89.

- **Dynamic SQL objects - ULPreparedStatement, ULResultSet, and ULResultSetSchema** objects allow you to create Dynamic SQL statements, make queries and execute INSERT, UPDATE and DELETE statements, and attain programmatic control over database result sets.

  ☞ For more information about the ULPreparedStatement, ULResultSet, and ULResultSetSchema objects, see “PreparedStatement” on page 111, “ResultSet” on page 117, and “ResultSetSchema” on page 123.

- **Table API objects - ULTable, ULColumn, and ULIndexSchema and ULColumnSchema** objects allow programmatic control over database tables, columns and indexes.

  ☞ For more information about the ULTable, ULColumn, and ULIndexSchema objects, see “ULTable class” on page 140, “ULColumn” on page 82, and “ULIndexSchema” on page 109.

- **Synchronization** objects allow you to control synchronization through the MobiLink synchronization server, providing you have the SQL Anywhere Studio suite.

  ☞ For more information about synchronization with MobiLink, see “Understanding Synchronization for UltraLite Applications” [*UltraLite Database User’s Guide*, page 143].

CHAPTER 2

Tutorial: An UltraLite for MobileVB Application for Palm OS

About this chapter

This chapter walks you through all the steps of building your first UltraLite for MobileVB application. The application synchronizes data with a database on your desktop computer.

Contents

Topic:	page
Introduction	8
Lesson 1: Create a project architecture	9
Lesson 2: Create a form interface	11
Lesson 3: Write connection, synchronization, and table code	13
Lesson 4: Deploy the application to a device	21
Summary	22

Introduction

This tutorial walks you through building an UltraLite for MobileVB application. At the end of the tutorial you will have an application and small database on the Palm emulator that synchronizes with a larger database running on your desktop machine. If you have a device set up to use TCP/IP, you can also run the application on the device.

Timing

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

Requirements, competencies and experience

This tutorial assumes:

- you have MobileVB and Microsoft Visual Basic 6 installed on your system.
- you are familiar with MobileVB and Microsoft Visual Basic 6
  - you can write, test, and troubleshoot a Visual Basic 6 application
  - you can add references and components as needed
  - you can use the Visual Basic Object Browser and navigate the Visual Basic 6 environment.

Note

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

To develop applications using the UltraLite for AppForge MobileVB component, you will need the AppForge Booster. If you are missing Booster, you can get it from *http://www.appforge.com/booster.html.*

Goals

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

Lesson 1: Create a project architecture

The first step is to create a directory to hold your work, and an UltraLite schema file that holds the database schema.

❖ Create a directory and a schema file

1. Create a directory to hold the files you create in this tutorial.

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

2. Design a database schema using the UltraLite Schema Painter:

   To start the UltraLite Schema Painter, choost Start ➤ Programs ➤ SQL Anywhere Studio 9 ➤ UltraLite ➤ UltraLite Schema Painter.

   ☞ For more information on creating database schemas with the UltraLite Schema Painter, see the "UltraLite Schema Painter Tutorial" [*UltraLite Database User's Guide*, page 83].

   - **Schema filename** tutcustomer.usm
   - **Table name** customer
   - **Columns in customer**

Column Name	Data Type (Size)	Column allows NULL values?	Default value
id	integer	No	autoincrement
fname	char(15)	No	None
lname	char(20)	No	None
city	char(20)	Yes	None
phone	char(12)	Yes	555-1234

   - **Primary key** ascending id

3. Export the schema to Palm with a creator id of *Syb3*.
   (a) From the File menu, choose Export Schema for Palm.
   (b) Enter Syb3 as the creator ID.
   (c) Save the file as *tutcustomer.pdb* in your tutorial directory.

> **A note on Palm Creator ID's**
> A Palm creator ID is assigned to you by Palm. You can use Syb3 as your creator ID when you make sample applications for your own learning. However, when you create your commercial application, you should use your own creator ID.

Configure MobileVB for UltraLite development

The next step is to create an UltraLite for MobileVB project for your application. You can use a MobileVB project to get MobileVB controls that are suitable for small devices. On the Visual Basic toolbar on the left of the Visual Basic environment, MobileVB tools appear in addition to the standard Visual Basic tools.

❖ To create an UltraLite for MobileVB project

1. Start MobileVB

   Click Start ➤ Programs ➤ AppForge MobileVB ➤ Start MobileVB.

2. Create a new project.

3. Choose a design target for your application. When asked for the Design Target, choose Palm OS.

4. Create a Visual Basic reference to the UltraLite for MobileVB component:
   - Click Project ➤ References
   - Ensure the box beside the item iAnywhere Solutions, UltraLite for MobileVB 9.0 is checked.

     If the control does not appear, for example if you installed MobileVB after installing SQL Anywhere Studio, run the following command to register the component:

     ```
     ulmvbreg -r
     ```
   - Ensure the box beside the item UltraLite Connection Parameters 9.0 is checked.
   - Click OK.

5. Give the Project a name.
   - Click Project ➤ MobileVBProject1 Properties
   - Under Project Name, type **UltraLiteTutorial** (all one word). This is the name of the application as it will appear on your device.
   - Click OK

6. Save the Project:
   - Choose File ➤ Save Project
   - For the Form filename, type **c:\tutorial\mvb\Form1.frm**.
   - Click Save.
   - For the Project filename, type **c:\tutorial\mvb\UltraLiteTutorial.vbp**.
   - Click Save.

You are now ready to proceed to the next step in the tutorial.

Lesson 2: Create a form interface

You are now ready to design your application form. The project should have a single form named Form1 displayed.

❖ To add controls to your project

1. Add the AppForge MobileVB controls and properties given in the table below to Form1:

Type	Name	Caption or text
AFTextBox	txtFname	
AFTextBox	txtLname	
AFTextBox	txtCity	
AFTextBox	txtPhone	
AFLabel	lblID	
AFButton	btnInsert	Insert
AFButton	btnUpdate	Update
AFButton	btnDelete	Delete
AFButton	btnNext	Next
AFButton	btnPrevious	Previous
AFButton	btnSync	Synchronize
AFButton	btnDone	End

Your form should look like the figure below:

2. Compile and validate the application.
   - Choose MobileVB ➤ Compile and Validate.

Lesson 3: Write connection, synchronization, and table code

The first step in developing your application is to write UltraLite code to connect to the database.

Write code for connecting to your database

In this application, you connect to the database in the Form Load event.

❖ Write code to connect to the UltraLite database

1. Specify the connection parameters.
   (a) Add a ULConnectionParms control to your form. The ULConnectionParms control is a database icon on the toolbox.
   (b) In the Properties window, specify the following ULConnectionParms properties:

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

2. Declare global UltraLite objects.

   Enter the following code at the top of the form in the declarations area. This code declares the UltraLite objects you will need in this sample.

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

   These variables are used through the application. Note that the DatabaseMgr variable is also allocated as a new object.

3. Add the code to connect to the database in the Form Load event.

   The code below opens the connection to the database and if the database is new, it assigns a schema to it.

```
Sub Form_Load()
    On Error Resume Next
    Set Connection = _
        DatabaseMgr.OpenConnectionWithParms( _
            ULConnectionParms1)
    If Err.Number = ULSQLCode.ulSQLE_NOERROR Then
        MsgBox "Connected to an existing database"
    ElseIf Err.Number = _
       ULSQLCode.ulSQLE_ULTRALITE_DATABASE_NOT_FOUND _
    Then
        Err.Clear
        Set Connection = _
        DatabaseMgr.CreateDatabaseWithParms( _
           ULConnectionParms1)
        If Err.Number = ULSQLCode.ulSQLE_NOERROR _
        Then
            MsgBox "Connected to a new database"
        Else
            MsgBox Err.Description
        End If
    Else
        MsgBox Err.Description
    End If
End Sub
```

This code attempts to connect to an existing database. If the database does not exist, it creates a new database from the schema file and establishes a connection.

4. Write the code that ends the application and closes the connection when the End button is clicked:

```
Sub btnDone_Click()
    Connection.Close
    End
End Sub
```

5. Run the application in the development environment.
   - Choose Run ➤ Start.
   - The first time you run the application, a message box is displayed with the message `Connected to a new database`. On subsequent runs the message is `Connected to an existing database`. The Form then loads.
   - Click End to terminate the application.

You have now written a routine to establish a connection to a database. The next lesson describes how to access data.

Write code for data manipulation

The next step is to write code for data manipulation and navigation.

❖ To open the table

1. Write code that initializes the table and moves to the first row.

   Add the following code to the Form_Load routine, just before the End Sub instruction:

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

   This code assigns the CustomerTable variable and opens the table so data can be read or manipulated. The call to MoveBeforeFirst positions the application before the first row of data in the table - but note that it is not strictly speaking, required, because after you call open, you are already positioned before the first row. There are no rows in the table at the moment.

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

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

   If the table has no rows, the application displays empty controls.

Otherwise, it displays the values stored in each of the columns of the current row of the database.

3. Call this function from the Form's Activate function.

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

This call ensures the fields get updated when the application starts.

At this stage you may wish to run the application to check that you have entered the code correctly. As there are no rows in the table, the controls are all empty.

❖ To insert rows into the table

1. Implement the code for the Insert button.

Add the following routine to the form:

```
Private Sub btnInsert_Click()
    Dim fname As String
    Dim lname As String
    Dim city As String
    Dim phone As String

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

    On Error GoTo InsertError
    CustomerTable.InsertBegin
    CustomerTable.Column("Fname").StringValue = _
         fname
    CustomerTable.Column("Lname").StringValue = _
         lname
    If Len(city) > 0 Then
        CustomerTable.Column("City").StringValue = _
         city
    End If
    If Len(phone) > 0 Then
       CustomerTable.Column("Phone").StringValue = _
       phone
    End If
    CustomerTable.Insert
    CustomerTable.MoveLast
    DisplayCurrentRow
    Exit Sub

InsertError:
    MsgBox "Error:  " & CStr(Err.Description)
End Sub
```

The call to InsertBegin puts the application into insert mode and sets all the values in the row to their defaults (for example, the ID column receives the next autoincrement value). The column values are set and then the new row is inserted. Note that if an error occurs during the insert, a message box will display the error number.

2. Run the application.

   After the initial message box, the form is displayed.

   - Enter a first name of Jane in the top text box and a last name of Doe in the second.
   - Click the Insert button. A row is added to the table with these values. The application moves to the last row of the table and displays the row. The label displays the autoincremented value of the ID column that UltraLite assigned to the row.
   - Enter a first name of John in the top text box and a last name of Smith in the second.
   - Click Insert to add this row to the table.
   - Click End to end the program.

With two rows in the table, it is now time to implement the code to scroll through the rows and display each.

❖ To move through the rows of the table

1. Implement the code for the Next and Previous buttons:

   Add the following routines to the form:

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

2. Run the application.

   When the form is first displayed, the controls are empty as the current position is before the first row.

   After the form is displayed, click Next and Previous to move through the rows of the table.

The next step is to modify the data in a row by updating or deleting it.

❖ To update and delete rows in the table

1. Implement the code for the Update button.

   Add the following routine to the form:

```
Private Sub btnUpdate_Click()
    Dim fname As String
    Dim lname As String
    Dim city As String
    Dim phone As String

    fname = txtFname.Text
    lname = txtLname.Text
    city = txtCity.Text
    phone = txtPhone.Text
    On Error GoTo UpdateError
    CustomerTable.UpdateBegin
    CustomerTable.Column("Fname").StringValue = _
        fname
    CustomerTable.Column("Lname").StringValue = _
        lname
    If Len(city) > 0 Then
        CustomerTable.Column("City").StringValue = _
        city
    Else
        CustomerTable.Column("City").SetNull
    End If
    If Len(phone) > 0 Then
       CustomerTable.Column("Phone").StringValue = _
        phone
    End If
    CustomerTable.Update
    DisplayCurrentRow
    Exit Sub

UpdateError:
    MsgBox "Error:  " & CStr(Err.Description)
End Sub
```

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

2. Implement the code for the Delete button.

   Add the following routine to the form:

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

The call to Delete deletes the current row on which the application is positioned.

3. Run the application.

   The data manipulation and display part of the application is now complete. Try inserting, updating, and deleting rows. Also, use the Next and Previous buttons to move through the rows. Check the label to see which row you are on.

Note
You can now run this application as a stand-alone application without SQL Anywhere Studio. If you wish to synchronize your UltraLite database with an Adaptive Server Anywhere database, please complete the next lesson in the tutorial.

Write code to synchronize

The final step is to write synchronization code. This step requires SQL Anywhere.

❖ To write code for the synchronize button

1. Implement the code for the Synchronize button.

   Add the following routine to the form:

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

   The SyncParms object contains the synchronization parameters. For this simple example, we start MobiLink so that it will add new users. Also, we send the column names to MobiLink so it can generate proper upload and download scripts.

   The code uses TCP/IP synchronization, and not HotSync synchronization. It works on a Palm OS device only as long as it is set up for TCP/IP synchronization.

2. From a command prompt, start the MobiLink synchronization server with the following command line:

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

The ASA 9.0 Sample database has a Customer table that matches the columns in the UltraLite database you have created. You can synchronize your UltraLite application with the ASA 9 Sample database.

The -zu+ and -za command line options provide automatic addition of users and generation of synchronization scripts. For more information on these options, see the *MobiLink Synchronization User's Guide.*

3. Start the UltraLite application.
4. Delete all the rows in your table.

   Any rows in the table would be uploaded to the ASA 9.0 Sample database.
5. Synchronize your application.
   - Click the Synchronize button.

     The MobiLink synchronization server window should scroll messages displaying the synchronization progress.
   - When the synchronization is complete, click Next and Previous to move through the rows of the table.

Lesson 4: Deploy the application to a device

Now that you are convinced the application runs properly, you can deploy it to the device.

❖ To deploy to the Palm device

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

Summary

Learning accomplishments

During this tutorial, you:

- created a database schema
- created an UltraLite for MobileVB application
- synchronized a remote database with an Adaptive Server Anywhere consolidated database using UltraLite
- increased your familiarity with MobileVB for UltraLite as an integrated system
- gained competence with the process of developing an UltraLite for MobileVB application

.

CHAPTER 3

Tutorial: An UltraLite Application for PocketPC

About this chapter

This chapter provides a tutorial to guide you through the process of building your first UltraLite for MobileVB application for CE. The application then synchronizes with a database.

Contents

Topic:	page
Introduction	24
Lesson 1: Create a project architecture	25
Lesson 2: Create a form interface	27
Lesson 3: Write the sample code	29
Lesson 4: Deploy to a device	37
Summary	38

Introduction

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

Timing

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

Competencies and experience

This tutorial assumes:

- you have MobileVB and Microsoft Visual Basic 6 installed on your system
- you can write, test, and troubleshoot a MobileVB application

> **Note**
> You can perform most of this tutorial without SQL Anywhere Studio. The synchronization sections of the tutorial require SQL Anywhere Studio.
>
> To develop applications using the UltraLite for AppForge MobileVB component, you will need the AppForge Booster. If you are missing Booster, you can get it from *http://www.appforge.com/booster.html.*

The synchronization section of this tutorial requires that you can use command line options and parameters.

Goals

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

Lesson 1: Create a project architecture

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

☞ For more information about database schemas, see "Creating UltraLite database schema files" [*UltraLite ActiveX User's Guide*, page 64].

❖ To create an UltraLite database schema

1. Create a directory for this tutorial.

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

2. Create a database schema using the UltraLite Schema Painter.
   - **Schema filename** tutcustomer.usm
   - **Table name** customer
   - **Columns in customer**

Column Name	Data Type (Size)	Column allows NULL values?	Default value
id	integer	No	autoincrement
fname	char(15)	No	None
lname	char(20)	No	None
city	char(20)	Yes	None
phone	char(12)	Yes	555-1234

   - **Primary key** ascending id

☞ For more information about creating a database schema, see the "UltraLite Schema Painter Tutorial" [*UltraLite Database User's Guide*, page 83].

Create a MobileVB project

The following procedure creates a MobileVB project for your application. It adds the UltraLite for MobileVB controls to a MobileVB project. On the Visual Basic toolbar on the left of the Visual Basic environment, MobileVB tools appear in addition to the standard Visual Basic tools.

❖ To create an UltraLite for MobileVB reference

1. Start MobileVB
   - Choose Start ➤ Programs ➤ AppForge MobileVB ➤ Start MobileVB.

   Choose a target and click OK.
2. Create a new project.
3. Choose a design target for your application. When asked for the Design Target, choose PocketPC.
4. Create a Visual Basic reference to the UltraLite for MobileVB component:
   - Choose Project ➤ References
   - Check the box beside the item iAnywhere Solutions, UltraLite for MobileVB 9.0.

     If the control does not appear, for example if you installed MobileVB after installing SQL Anywhere Studio, run the following command to register the component:

     ```
     ulmvbreg -r
     ```

     Ensure the box beside the item UltraLite Connection Parameters 9.0 is checked.
   - Click OK.
5. Give the Project a name.
   - Click Project ➤ MobileVBProject1 Properties
   - Under Project Name, type **UltraLiteTutorial**. This is the name of the application as it will appear on your device.
   - Click OK.
6. Save the Project:
   - Choose File ➤ Save Project.
   - Save the form as *c:\tutorial\mvb\Form1.frm*.
   - Save the project as *c:\tutorial\mvb\UltraLiteTutorial.vbp*.

Lesson 2: Create a form interface

After completing the steps in "Lesson 1: Create a project architecture" on page 25, the project should have a single form displayed.

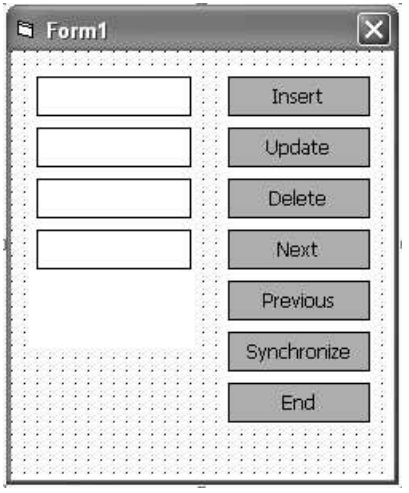
❖ To add controls to your project

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

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

2. Check the application.
   - ♦ Choose MobileVB ➤ Compile and Validate.

Your form should look something like this:


Form1
Insert
Update
Delete
Next
Previous
Synchronize
End

Lesson 3: Write the sample code

The first step in developing your application is to write UltraLite code to connect to the database.

Write code to connect to your database

In this application, you connect to the database using the Form Load event.

❖ Write code to connect to the UltraLite database

1. Specify the connection parameters.
   (a) Add a ULConnectionParms control to your form. The ULConnectionParms control is a database icon on the toolbox.
   (b) In the Properties window, specify the following ULConnectionParms properties:

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

2. Declare global UltraLite objects.
   - Double-click the form to open the Code window.
   - Enter the following code at the top of the form in the declarations area. This code declares a database manager, a connection and a table:

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

   These variables will be used through the application. Note that the DatabaseMgr variable is also allocated as a new object.

3. Add the code to connect to the database in the Form Load event.

   The code below attempts to connect to an existing database. If the database does not exist, it creates a new database from the schema file and establishes a connection.

```
Sub Form_Load()
    On Error Resume Next
    Set Connection = _
        DatabaseMgr.OpenConnectionWithParms( _
            ULConnectionParms1)
    If Err.Number = ULSQLCode.ulSQLE_NOERROR Then
        MsgBox "Connected to an existing database"
    ElseIf Err.Number = _
       ULSQLCode.ulSQLE_ULTRALITE_DATABASE_NOT_FOUND _
    Then
        Err.Clear
        Set Connection = _
        DatabaseMgr.CreateDatabaseWithParms( _
           ULConnectionParms1)
        If Err.Number = ULSQLCode.ulSQLE_NOERROR _
        Then
            MsgBox "Connected to a new database"
        Else
            MsgBox Err.Description
        End If
    Else
        MsgBox Err.Description
    End If
End Sub
```

4. Write the code that ends the application and closes the connection when the End button is clicked:

```
Sub btnDone_Click()
    Connection.Close
    End
End Sub
```

5. Run the application in the development environment.
   - Choose Run ➤ Start.
   - The first time you run the application, a message box is displayed with the message `Connected to a new database`. On subsequent runs the message is `Connected to an existing database`.

     The form loads.
   - Click End to terminate the application.

Write code for navigation and data manipulation

The following procedure implements data manipulation and navigation.

❖ To open the table

1. Write code that initializes the table and moves to the first row.

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

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

   This code assigns the customer table to CustomerTable and opens the table so data can be read or manipulated. The call to MoveBeforeFirst positions the application before the first row of data in the table. This call is not necessarily required because the call to open positions the application before the first row. There are currently no rows in the table.

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

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

   If the table has no rows, the application displays empty controls. Otherwise, it displays the values stored in each of the columns of the

current row of the database.

3. Call this function from the Form's Activate function.

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

This call ensures the fields get updated when the application starts.

At this stage you may want to run the application to verify that you have entered the code correctly. As there are no rows in the table, the controls are all empty.

❖ To insert rows into the table

1. Implement the code for the Insert button.

Add the following procedure to the form:

```
Private Sub btnInsert_Click()
    Dim fname As String
    Dim lname As String
    Dim city As String
    Dim phone As String

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

    On Error GoTo InsertError
    CustomerTable.InsertBegin
    CustomerTable.Column("Fname").StringValue = _
      fname
    CustomerTable.Column("Lname").StringValue = _
      lname
    If Len(city) > 0 Then
        CustomerTable.Column("City").StringValue = _
      city
    End If
    If Len(phone) > 0 Then
       CustomerTable.Column("Phone").StringValue = _
       phone
    End If
    CustomerTable.Insert
    CustomerTable.MoveLast
    DisplayCurrentRow
    Exit Sub

InsertError:
    MsgBox "Error:  " & CStr(Err.Description)
End Sub
```

The call to InsertBegin puts the application into insert mode and sets all the values in the row to their defaults. For example, the ID column

receives the next autoincrement value. The column values are set and then the new row is inserted. Note that if an error occurs during the insert, a message box will display the error number.

2. Run the application.

   After the initial message box, the form is displayed.

   - Enter a first name of Jane in the top text box and a last name of Doe in the second.
   - Click the Insert button. A row is added to the table with these values. The application moves to the last row of the table and displays the row. The label displays the autoincremented value of the ID column that UltraLite assigned to the row.
   - Enter a first name of John in the top text box and a last name of Smith in the second.
   - Click Insert to add this row to the table.
   - Click End to end the program.

The following procedure implements the code to scroll through the rows and display each.

❖ To move through the rows of the table

1. Implement the code for the Next and Previous buttons.

   Add the following procedures to the form:

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

2. Run the application.

   When the form is first displayed, the controls are empty as the current position is before the first row.

   Click Next and Previous to move through the rows of the table.

The following procedure modifies the data in a row by updating or deleting it.

❖ To update and delete rows in the table

1. Implement the code for the Update button.

   Add the following procedure to the form:

```
Private Sub btnUpdate_Click()
    Dim fname As String
    Dim lname As String
    Dim city As String
    Dim phone As String

    fname = txtFname.Text
    lname = txtLname.Text
    city = txtCity.Text
    phone = txtPhone.Text
    On Error GoTo UpdateError
    CustomerTable.UpdateBegin
    CustomerTable.Column("Fname").StringValue = fname
    CustomerTable.Column("Lname").StringValue = lname
    If Len(city) > 0 Then
        CustomerTable.Column("City").StringValue = city
    Else
        CustomerTable.Column("City").SetNull
    End If
    If Len(phone) > 0 Then
       CustomerTable.Column("Phone").StringValue = phone
    End If
    CustomerTable.Update
    DisplayCurrentRow
    Exit Sub

UpdateError:
    MsgBox "Error:  " & CStr(Err.Description)
End Sub
```

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

2. Implement the code for the Delete button.

   Add the following procedure to the form:

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

   The call to Delete deletes the current row on which the application is positioned.

3. Run the application.

   The data manipulation and display portion of the application is now complete.

Note
You can now run this application as a stand-alone application without SQL Anywhere Studio. To synchronize your UltraLite database with an Adaptive Server Anywhere database, complete the remainder of this lesson.

Write code to synchronize

The following procedure implements synchronization. This procedure requires SQL Anywhere.

❖ To write code for the synchronize button

1. Implement the code for the Synchronize button.

   Add the following procedure to the form:

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

   The SyncParms object contains the synchronization parameters. For this simple example, we start MobiLink so that it will add new users. Also, we send the column names to MobiLink so it can generate proper upload and download scripts.

2. From a command prompt, start the MobiLink synchronization server with the following command line:

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

   The ASA 9.0 Sample database has a Customer table with columns matching those in the UltraLite database you have created. You can synchronize your UltraLite application with the ASA 9.0 Sample database.

   The -zu+ and -za command line options provide automatic addition of users and generation of synchronization scripts. For more information on these options, see the *MobiLink Synchronization User's Guide.*

3. Start the UltraLite application.
4. Delete all the rows in your table.

   Any rows in the table would be uploaded to the ASA 9.0 Sample database.
5. Synchronize your application.
   - Click the Synchronize button.

     The MobiLink synchronization server window displays the synchronization progress.
   - When the synchronization is complete, click Next and Previous to move through the rows of the table.

Lesson 4: Deploy to a device

The following procedure deploys your application to a mobile device.

❖ To deploy to a PocketPC device

1. Configure the application settings.
   - From the MobileVB menu, choose MobileVB Settings.

     A dialog appears.
   - In the left pane, choose Dependencies and click the User Dependencies tab.
   - Click the Add button and select the *c:\tutorial\mvb\tutCustomer.usm*. This indicates to MobileVB that the file should be included in the deployment.
   - Choose the PocketPC Settings item in the left pane
   - Enter *\tutorial\mvb* for the Device Installation Path.
   - Click OK to close the dialog.
2. From the MobileVB menu, choose Deploy to Device, and make sure you select the PocketPC device. If a dialog appears asking if you want to save the project, choose Yes.
3. If you are running a version of MobileVB that is older than 3.0, you will also need to copy the UltraLite control to the device.

   Copy the file *ulmvb9.dll* to *\Program Files\AppForge* on your device. The file is located in one of the following platform-specific subdirectories of your SQL Anywhere 9 installation: *\UltraLite\UltraLiteForMobileVB\ce\arm* or *\UltraLite\UltraLiteForMobileVB\ce\mips*. This step only needs to be performed once per device.
4. On your device, tap Start ➤ Programs.
5. Tap UltraLiteTutorial to run your application.

Summary

Learning accomplishments

During this tutorial, you:

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

CHAPTER 4

Tutorial: Using Dynamic SQL in an UltraLite Application for PocketPC

About this chapter

This chapter provides a tutorial to guide you through the process of building an UltraLite for MobileVB application. This tutorial differs from "Tutorial: An UltraLite Application for PocketPC" on page 23 in that you use dynamic SQL to access the UltraLite database.

Contents

Topic:	page
Introduction	40
Lesson 1: Create a project architecture	41
Lesson 2: Create a form interface	43
Lesson 3: Write connection, synchronization, and table code	44
Lesson 4: Deploy the application to a device	50
Summary	51

Introduction

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

Timing

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

Competencies and experience

This tutorial assumes:

- you have MobileVB and Microsoft Visual Basic 6 installed on your computer
- you are familiar with AppForge MobileVB and Microsoft Visual Basic 6
  - you can write, test, and troubleshoot a MobileVB application
  - you can add references and components as needed
- you know how to create an UltraLite schema using the UltraLite Schema Painter.

Note

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

To develop applications using the UltraLite for AppForge MobileVB component, you will need the AppForge Booster. If you are missing Booster, you can get it from http://www.appforge.com/booster.html.

Goals

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

Lesson 1: Create a project architecture

The first step is to create an UltraLite for MobileVB project for your application. You can use a MobileVB project to get MobileVB controls that are suitable for small devices. On the Visual Basic toolbar on the left of the Visual Basic environment, MobileVB tools appear in addition to the standard Visual Basic tools.

❖ To create an UltraLite database schema

1. Create a directory to hold the files you create in this tutorial.

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

2. Create a database schema using the UltraLite Schema Painter.

   For more information about creating a database schema using the UltraLite Schema Painter, see the "UltraLite Schema Painter Tutorial" [*UltraLite Database User's Guide*, page 83].

   - **Schema filename** tutorial.usm
   - **Table name** names
   - **Columns in names**

Column name	Data type (size)	Allow NULL?	Default value
id	integer	No	autoincrement
name	char(15)	No	None

   - **Primary key** ascending id

❖ To create an UltraLite for MobileVB reference

1. Start MobileVB.

   Click Start ➤ Programs ➤ AppForge MobileVB ➤ Start MobileVB.

   The Project Manager appears.

2. Click New Project.

3. Choose a target and click OK.

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

- Click OK.

5. Save the Project:
   - Choose File ➤ Save Project.
   - Save the form as *c:\tutorial\mvb\Form1.frm.*
   - Save the Project as *c:\tutorial\mvb\UltraLiteTutorial.vbp.*

Lesson 2: Create a form interface

After completing the steps in "Lesson 1: Create a project architecture" on page 41, your project should have a single form displayed.

❖ To add a controls to your project

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

   Add the text box and label to the left side of the form. Add the buttons down the right side of the form.

Type	Name	Caption or text
AFTextBox	txtName	
AFLabel	lblID	
AFButton	btnInsert	Insert
AFButton	btnUpdate	Update
AFButton	btnDelete	Delete
AFButton	btnNext	Next
AFButton	btnPrevious	Previous
AFButton	btnDone	End

2. Check the application.
   - Choose MobileVB ➤ Compile and Validate.

Lesson 3: Write connection, synchronization, and table code

The first step in developing your application is to write UltraLite code to connect to the database.

Write code for connecting to your database

In this application, you connect to the database in the Form Load event.

❖ Write code to connect to the UltraLite database

1. Specify the connection parameters.
   (a) Add a ULConnectionParms control to your form. The ULConnectionParms control is a database icon on the toolbox.
   (b) In the Properties window, specify the following ULConnectionParms properties:

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

2. Declare the UltraLite objects you need.
   - Double-click the form to open the Code window.
   - Enter the following code at the top of the form in the declarations area. This code declares the DatabaseManager and Connection objects that you will need in this sample. It also creates two objects, a PreparedStatement and ResultSet, that you will use for data manipulation:

```
Dim DatabaseMgr As New ULDatabaseManager
Dim MyConnection As ULConnection
Dim MyPrepStmt As ULPreparedStatement
Dim MyResultSet As ULResultSet
```

   These variables will be used through the application. Note that the DatabaseMgr variable is also allocated as a new object. This is the only object that can be instantiated.

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

   The code below opens the connection to the database and if the database is new, it assigns a schema to it. Note that we have not added extensive

error handling to our code fragments. You should implement error handling in your own applications.

```
Sub Form_Load()
    On Error Resume Next
    Set MyConnection = _
      DatabaseMgr.OpenConnectionWithParms(ULConnectionParms1
         )
    If Err.Number = ULSQLCode.ulSQLE_NOERROR Then
        MsgBox "Connected to an existing database"
    ElseIf Err.Number = _
        ULSQLCode.ulSQLE_ULTRALITE_DATABASE_NOT_FOUND Then
        Err.Clear
        Set MyConnection = _

         DatabaseMgr.CreateDatabaseWithParms(ULConnectionPar
         ms1)
                        MsgBox "Connected to a new database"
    Else
       MsgBox "Connect error:" & Err.Description
    End If
End Sub
```

This code attempts to connect to an existing database. If the database does not exist, it creates a new database from the schema file and establishes a connection.

☞ For more information, see "ULConnection class" on page 89.

4. Write the code that ends the application and closes the connection when the End button is clicked:

```
Sub btnDone_Click()
    MyConnection.Close
    End
End Sub
```

5. Run the application in the development environment.
   - Choose Run ➤ Start.
   - After an initial message box, the form loads.

     The first time you run the application, the message is Connected to a new database.

     Subsequent times, the message is Connected to an existing database.
   - Click End to terminate the application.
   - Once you have successfully connected, you can comment out the message box if you wish.

Write code for data manipulation

The following procedure implements data manipulation and navigation.

❖ To open the table

1. Write code that initializes the table and moves to the first row.

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

```
On Error GoTo 0
Set MyPrepStmt = _
  MyConnection.PrepareStatement("SELECT ID, Name FROM
         Names")
Set MyResultSet = MyPrepStmt.ExecuteQuery
If MyResultSet.MoveFirst then
   DisplayCurrentRow
End If
```

   Setting the prepared statement gives you a result set from a SELECT statement.

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

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

   If the table has no rows, the application displays empty controls. Otherwise, it displays the values stored in each of the columns of the current row of the database. Note how the type for each Get statement is specific to the column data type.

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

3. At this stage you may want to run the application to check that you have entered the code correctly. As there are no rows in the table, the controls are all empty.

❖ To insert rows into the table

1. Implement the code for the Insert button.

   Add the following procedure to the form:

   ```
   Private Sub btnInsert_Click()
     Dim InsertStmt As ULPreparedStatement
     Set InsertStmt = MyConnection.PrepareStatement( _
          "INSERT INTO names(name) VALUES(?)")
     InsertStmt.SetStringParameter 1, txtName.Text
     InsertStmt.ExecuteStatement
   End Sub
   ```

   The column values are set and then the new row is inserted.

2. Run the application.

   After the initial message box, the form is displayed.

   - Enter a name in the text box.
   - Click the Insert button. A row is added to the table with this value.
   - Enter another name in the text box.
   - Click Insert to add this row to the table.
   - Click End to end the program.

The following procedure implements the code to scroll through and display the rows.

❖ To move through the rows of the table

1. Implement the code for the Next and Previous buttons.

   Add the following procedures to the form:

   ```
   Private Sub btnNext_Click()
       MyResultSet.MoveNext
       DisplayCurrentRow
   End Sub

   Private Sub btnPrevious_Click()
       MyResultSet.MovePrevious
       DisplayCurrentRow
   End Sub
   ```

2. Run the application.

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

The following procedure modifies the data in a row by updating or deleting it.

❖ To update and delete rows in the table

1. Implement the code for the Update button.

   Add the following procedure to the form:

```
Private Sub btnUpdate_Click()
  Dim MyUpdateStmt as ULPreparedStatement
  Set MyUpdateStmt = MyConnection.PrepareStatement("UPDATE
         Names SET Name = ? WHERE ID = ?")
  MyUpdateStmt.SetStringParameter 1, txtName.Text
  MyUpdateStmt.SetIntegerParameter 2, CLng(lblID.Caption)
  MyUpdateStmt.ExecuteStatement

  MsgBox "Row updated"
  DisplayCurrentRow
End Sub
```

   The column values are updated.

2. Implement the code for the Delete button.

   Add the following procedure to the form:

```
Private Sub btnDelete_Click()
  Dim MyDeleteStmt As ULPreparedStatement
  Set MyDeleteStmt = MyConnection.PrepareStatement( _
    "DELETE FROM Names WHERE ID = ?")
  MyDeleteStmt.SetIntegerParameter 1, CLng( lblID.Caption )
  MyDeleteStmt.ExecuteStatement
  DisplayCurrentRow
End Sub
```

   The call to Delete deletes the current row according to the value set in the WHERE clause.

3. Run the application.

   The data manipulation and display portion of the application is now complete.

Note

You can now run this application as a stand-alone application without SQL Anywhere Studio. To synchronize your UltraLite database with an Adaptive Server Anywhere database, complete the remainder of this lesson.

Write code to synchronize

The following procedure implements synchronization. This procedure requires SQL Anywhere.

❖ To write code for the synchronize button

1. Add a button named btnSync to your form.
2. Add the following procedure to the form:

```
Private Sub btnSync_Click()
    With MyConnection.SyncParms
        .UserName = "afsample"
        .Stream = ulTCPIP
        .Version = "ul_default"
        .SendColumnNames = True
    End With
    MyConnection.Synchronize
    MyResultSet.Close
    Set MyResultSet = MyPrepStmt.ExecuteQuery
    MyResultSet.MoveFirst
    DisplayCurrentRow
End Sub
```

   The SyncParms object contains the synchronization parameters. Setting its SendColumnNames property to true sends the column names to MobiLink so that it can generate proper upload and download scripts.

3. Create the Adaptive Server Anywhere consolidated database by running *Samples\UltraLite\Names\build.bat.*
4. From a command prompt, start the MobiLink synchronization server with the following command line:

```
dbmlsrv9 -c "dsn=ASAConsolidated" -v+ -zu+ -za
```

   The ASAConsolidated database has a table that matches the columns in the UltraLite database you have created. You can synchronize your UltraLite application with this database.

   The -zu+ and -za command line options provide automatic addition of users and generation of synchronization scripts. For more information about these options, see “MobiLink Synchronization Server Options” [*MobiLink Synchronization Reference*, page 3].

5. Delete the existing .udb file to avoid any conflicts.
6. Start the UltraLite application.
7. Synchronize your application.

   Click Synchronize.

   The MobiLink synchronization server window displays the synchronization progress.

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

Lesson 4: Deploy the application to a device

The final step is to deploy the application to a device.

❖ To deploy to the PocketPC device

1. Configure the application settings.
   - From the MobileVB menu, choose MobileVB Settings.
   - In the dialog that appears, choose Dependencies in the left pane and click the User Dependencies tab.
   - Click the Add button and select the *c:\tutorial\mvb\tutorial.usm*. This indicates to MobileVB that the file should be included in the deployment.
   - In the left pane, Choose PocketPC Settings.
   - Enter *\Tutorial\mvb* for the Device Installation Path.
   - Click OK to close the dialog.
2. From the MobileVB menu, choose Deploy to Device, and make sure you select the PocketPC device. If a dialog appears asking if you want to save the project, choose Yes. If you have not yet done so, be sure to name the project.
3. If you are running a version of MobileVB that is older than 3.0, you will also need to copy the UltraLite control to the device. Copy from your desktop, the file *\UltraLite\UltraLiteForMobileVB\ce\arm\ulmvb9.dll* or *\UltraLite\UltraLiteForMobileVB\ce\mips\ulmvb9.dll* located in the installation path of SQL Anywhere Studio to *\Program Files\AppForge* on your device. This step only needs to be performed once per device.
4. On your device, go to your Programs.
5. Choose UltraLiteTutorial. You are now running your application.

Summary

Learning accomplishments

During this tutorial, you:

- created a database schema
- created an UltraLite for MobileVB application
- synchronized an UltraLite remote database with an Adaptive Server Anywhere consolidated database
- increased your familiarity with the UltraLite Component Suite as an integrated system
- gained competence with the process of developing an UltraLite application

Samples

For more code samples, see the following project group. Paths are relative to your SQL Anywhere installation:

- *Samples\UltraLiteForMobileVB\custdb\custdb.vbg*
- *Samples\UltraLiteForMobileVB\grid\gridsample.vbg*

CHAPTER 5

Understanding UltraLite for MobileVB Development

About this chapter

This chapter describes how to develop applications with the UltraLite for MobileVB component.

Contents

Topic:	page
Connecting to the UltraLite database	54
Accessing data using dynamic SQL	58
Accessing data using the table-based API	63
Transaction processing in UltraLite	69
Accessing schema information	70
Error handling	71
Synchronization	72
Component samples, demonstrations and code fragments	74
Maintaining database state on Palm OS	75

Connecting to the UltraLite database

Any UltraLite application must connect to its database before it can carry out any operation on the data, including applying a schema to the database.

Using ULConnectionParms to connect

❖ To connect to an UltraLite database using ULConnectionParms

1. Place the ULConnectionParms object on your form and ensure all of the database and schema parameters are complete.

   You need one ULConnectionParms object per application. The ULConnectionParms object is located on the MobileVB tool palette.

   - Double click the ULConnectionParms object. The ULConnectionParms object appears on your form.
   - In the ULConnectionParms properties palette, type in the location of the database, schema files and username and password for your database.

2. Create your ULDatabaseManager object.

   The following code declares a ULDatabaseManager object named dbMgr

   ```
   Public dbMgr As New ULDatabaseManager
   ```

3. Create and open a connection to the database.

   The ULDatabaseManager CreateDatabaseWithParms and OpenConnectionWithParms methods create a database and open a connection. Each takes a single ULConnectionParms object as its argument. The ULConnectionParms object is is composed of a set of parameters that you created when you placed the ULConnectionParms object on your form and filled in the vital properties. A schema file must be specified for CreateDatabaseWithParms and a database file must be specified for OpenConnectionWithParms.

   The following properties, shown in the ULConnectionParms property palette, are mandatory for CreateDatabaseWithParms:

Keyword	Description
SchemaOnDesktop	The path and filename of the UltraLite schema. The default extension for UltraLite schema files is *.usm*. SchemaOnDesktop is required when using CreateDatabaseWithParms on Windows desktop operating systems. SchemaOnCE has precedence over SchemaOnDesktop. Required for CreateDatabaseWithParms.
SchemaOnCE	The path and filename of the UltraLite schema on Windows CE. The default extension for UltraLite schema files is *.usm*. This is a required parameter when using CreateDatabaseWithParms for CE.
SchemaOnPalm	If using Palm, the name of the UltraLite schema for Palm. SchemaOnPalm is a required parameter when using CreateDatabaseWithParms on Palm devices. The Palm file extension is *.pdb*. Do *not* specify the extension in the SchemaOnPalm parameter.

☞ For more information on connection parameters used WithParms, see "DatabaseManager" on page 100.

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

Using a connection string to connect

❖ To connect to an UltraLite database

1. Create a ULDatabaseManager object.

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

   The following code creates a ULDatabaseManager object named dbMgr

   ```
   Public dbMgr As ULDatabaseManager
   ...
   Set dbMgr = New ULDatabaseManager
   ```

2. Create and open a connection to the database.

   The ULDatabaseManager CreateDatabase and OpenConnection methods are used to Create a database and Open a connection. Each takes a single

string as its argument. The string is composed of a set of keyword-value pairs separated by semicolons. A schema file must be specified for CreateDatabase and a database file must be specified for OpenConnection.

The following are mandatory connection parameters for CreateDatabase:

Keyword	Description
schema_file	The path and filename of the UltraLite schema. The default extension for UltraLite schema files is *.usm*. SCHEMA_FILE is a required parameter when using CreateDatabase on Windows desktop operating systems. CE_SCHEMA has precedence over SCHEMA_FILE. Required for CreateDatabase.
ce_schema	The path and filename of the UltraLite schema on Windows CE. The default extension for UltraLite schema files is *.usm*. CE_SCHEMA is a required parameter when using CreateDatabase for CE.
palm_schema	If using Palm, the name of the UltraLite schema for Palm. PALM_SCHEMA is a required parameter when using CreateDatabase on Palm devices. The Palm file extension is *.pdb*. Do *not* specify the extension in the palm_schema parameter.

☞ For more information on connection parameters, see "Connection Parameters" [*UltraLite Database User's Guide*, page 49].

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

The following code opens a connection to an UltraLite database named *mydata.udb* (assuming the file exists).

```
Public conn As ULConnection
Dim conParms as String
Dim filePath as String
filepath="c:\tutorial"
conParms = "uid=dba;pwd=sql;dbf=" + filepath +
   "\mydata.udb"
Set conn = dbMgr.OpenConnection(conParms)
```

Using the ULConnection object

Properties of the ULConnection object govern global application behavior, including the following:

- **Commit behavior** By default, UltraLite applications are in AutoCommit mode. Each Insert, Update, or Delete statement or action is committed to the database immediately. You can also set ULConnection.AutoCommit to False to build explicit transaction handling into your application.

  ☞ For more information, see "Transaction processing in UltraLite" on page 69.

- **User authentication** You can change the user ID and password for the application from the default values of DBA and SQL by using the GrantConnectTo and RevokeConnectFrom methods.

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

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

- **Prepared statements** SQL statements are accessed using the PreparedStatement method.

Encryption and obfuscation

☞ For background information about database encryption, see "Encrypting UltraLite databases" [*UltraLite Database User's Guide*, page 36].

You can encrypt or obfuscate your UltraLite database when using UltraLite for MobileVB. When you call CreateDatabaseWithParms and pass in the ConnectionParms object, with a value in place for EncryptionKey, the database is created with encryption. A way to change the encryption key is by specifying the new encryption key on the Connection object. In this example, "apricot" is the key.

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

Connections to the database, such as OpenConnectionWithParms, must, after the database is encrypted, specify apricot as the EncryptionKey property. Otherwise, the connection fails.

To obfuscate the database, obfuscate=1 as a creation parameter.

Accessing data using dynamic SQL

UltraLite applications can access data in an Ultralite database using Dynamic SQL. This section covers the following topics:

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

Data manipulation: INSERT, UPDATE and DELETE

UltraLite can perform numerous common SQL Data Manipulation Language operations using the ExecuteStatement method, a member of the ULPreparedStatement class. With UltraLite, you can perform INSERT, UPDATE and DELETE operations just as you can with any SQL database.

UltraLite handles variable values using the ? character.

Using input parameters (?) in your prepared statements
For any INSERT, UPDATE or DELETE, each ? is referred to by its ordinal position in the prepared statement. The first ? is referred to as 1, the second 2, and so on.

❖ To perform INSERT operations using ExecuteStatement:

1. Declare a variable as ULPreparedStatement

```
Dim PrepStmt As ULPreparedStatement
Dim NewValue As String
Dim RowCount As Long
```

2. Assign a statement to the ULPreparedStatement object.

```
Set PrepStmt = Connection. _
PrepareStatement("INSERT MyTable(MyColumn) VALUES (?)")
```

3. Assign input parameter values for the statement

```
PrepStmt.SetStringParameter 1, NewValue
```

4. Execute the statement

```
RowCount = PrepStmt.ExecuteStatement
```

❖ **To perform UPDATE operations using ExecuteStatement:**

1. Declare variables needed for the operation:

```
Dim PrepStmt As ULPreparedStatement
Dim NewValue as String
Dim RowCount As Long
Dim ID As Integer
```

2. Assign a prepared statement to your ULPreparedStatement object.

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

3. Assign input parameter values for the statement, using a variable named *ID* you created.

```
PrepStmt.SetStringParameter 1, NewValue
 PrepStmt.SetIntParameter 2, ID
```

4. Execute the statement

```
RowCount = PrepStmt.ExecuteStatement
```

❖ **To perform DELETE operations using ExecuteStatement:**

1. Declare a variable as ULPreparedStatement

```
Dim PrepStmt As ULPreparedStatement
Dim RowCount As Long
Dim ID As Integer
```

2. Assign a statement to the ULPreparedStatement object.

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

3. Assign input parameter values to the statement

```
PrepStmt.SetIntParameter 1, ID
```

4. Execute the statement

```
RowCount = PrepStmt.ExecuteStatement
```

Data retrieval: SELECT

Use the SELECT statement to retrieve information from the database.

❖ **To execute a SELECT query using ExecuteQuery:**

1. Declare variables required for the operation:

```
Dim PrepStmt As ULPreparedStatement
Dim MyResultSet As ULResultSet
```

2. Assign a prepared statement to your ULPreparedStatement object.

```
Set PrepStmt = Connection.PrepareStatement( _
  "SELECT Name FROM customer")
```

3. Execute the statement. In the following code, a MobileVB listbox captures the result of the SELECT query.

```
Set MyResultSet = PrepStmt.ExecuteQuery
While MyResultSet.MoveNext
  aflistbox1.AddItem MyResultSet.GetString(1)
Wend
```

For more information on moving through result sets, see "Navigating through Dynamic SQL result sets" on page 61

Get column values from your database

UltraLite for MobileVB provides you with a number of methods to get data of particular types from the UltraLite database into a result set. MobileVB does not permit the use of *Variant* data types, and because of this, UltraLite for MobileVB is equipped to handle all types of data, but you must use a specific method suitable to each data type contained in your UltraLite database. To call your method, use the following to guide your writing: **MyResultSetName.MethodName(***Index***)**, where *Index* is the ordinal position of the column name in your **SELECT** statement.

Numerous Get types are provided including those for retrieving SQLTypes.

Consider a **SELECT** query used in the code below. Here, a result set called MyResultSet is created.

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

The example below illustrates the use of these methods. This example uses GetInteger and GetString to call Integer and String values from the database into a control on the form:

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

GetInteger retrieves integer data from the first column returned by the SELECT query, and the method GetString retrieves string data from the second column returned by the SELECT query.

Navigating through dynamic SQL result sets

You can navigate through a result set in MobileVB using methods associated with the ULResultSet object.

Move through a result set

UltraLite for MobileVB provides you with a number of methods to navigate a result set. To call your method, use the following model to guide your writing: MyResultSetName.MethodName.

The following methods allow you to navigate your result set:

- **MoveAfterLast** moves to a position after the last row.
- **MoveBeforeFirst** moves to a position before the first row.
- **MoveFirst** moves to the first row.
- **MoveLast** moves to the last row.
- **MoveNext** moves to the next row.
- **MovePrevious** moves to the previous row.
- **MoveRelative** moves a certain number of rows relative to the current row, as specified in the index. Relative to the current position of the cursor in the result set, positive index values move forward in the result set, negative index values move backward in the result set and zero does not move the cursor. Zero is useful if you want to repopulate a row buffer.

The following example shows you how you can use **MoveFirst** to navigate within a result set.

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

Result set schema description

The ULResultSet.Schema property allows you to retrieve result set schema properties. Available properties include ColumnName, ColumnCount, ColumnPrecision, ColumnScale, ColumnSize and ColumnSQLType. The following example shows how you can use ULResultSet.Schema to display schema information in a MobileVB grid. The example assumes you have a ResultSet named *MyResultSet* and a MobileVB grid named *grdSchema*.

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

Accessing data using the table-based API

UltraLite applications can access data in tables in a row-by-row fashion. This section covers the following topics:

- Scrolling through the rows of a table.
- Accessing the values of the current row.
- Using Find and Lookup methods to locate rows in a table.
- Inserting, deleting, and updating rows.

The section also provides a lower-level description of the way that UltraLite operates on the underlying data to help you understand how it handles transactions, and how changes are made to the data in your database.

Data manipulation internals

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

When your application changes its row (by a ULTable.MoveNext method or other method on the ULTable object) UltraLite copies the row to a buffer. There is one buffer per table. Any operations using ULColumn properties to get or set values affect only the copy of data in this buffer. They do not affect the data in the database. For example, the following statement changes the value of the ID column in the buffer to 3.

```
TCustomer.GetColumn( "ID" ).IntegerValue = 3
```

Using UltraLite modes

UltraLite uses the values in the buffer for a variety of purposes, depending on the kind of operation you are carrying out. UltraLite has four different modes of operation, in addition to a default mode, and in each mode the buffer is used for a different purpose.

- **Insert mode** The data in the buffer is added to the table as a new row when the ULTable.Insert method is called. The buffer is initially empty.
- **Update mode** The data in the buffer replaces the current row when the ULTable.Update method is called.
- **Find mode** The data in the buffer is used to locate rows when one of the ULTable.Find methods is called.
- **Lookup mode** The data in the buffer is used to locate rows when one of the ULTable.Lookup methods is called.

Whichever mode you are using, there is a similar sequence of operations:

1. Enter the mode.

   The ULTable InsertBegin, UpdateBegin, FindBegin, and LookupBegin methods set UltraLite into the mode.

2. Set the values in the buffer.

   Use the ULColumn object to set values in the buffer.

3. Carry out the operation.

   Use a ULTable method such as Insert, Update, FindFirst, or LookupForward to carry out the operation, using the values in the buffer. In most cases the UltraLite mode is set back to the default method and you must enter a new mode before performing another data manipulation or searching operation. An exception is that Delete does not affect the Find mode.

Scrolling through the rows of a table

The following code opens the customer table and scrolls through its rows, displaying a message box with the value of the lname column for each row.

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

You expose the rows of the table to the application when you open the table object. By default, the rows are exposed in order by primary key value, but you can specify an index to access the rows in a particular order. The following code moves to the first row of the customer table as ordered by the ix_name index.

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

Accessing the values of the current row

At any time, a ULTable object is positioned at one of the following positions:

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

If the ULTable object is positioned on a row, you can use the Column method together with a method appropriate for the data type of that column to access the value of that row. For example, the following expression represents the value of the lname column, as a character string:

```
TCustomer.Column( "lname" ).StringValue
```

The following expression represents the value of the ID column, an integer:

```
TCustomer.Column( "ID" ).IntegerValue
```

You can assign values to the properties even if you are before the first row or after the last row of the table. You cannot, however, get values from the column.

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

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

Casting values

The method you choose on the ULColumn object must batch the Visual Basic data type you wish to assign. UltraLite automatically casts data types where they are compatible, so that you could use the StringValue method to fetch an integer value into a string variable, and so on.

☞ For more information on accessing values of the current row, see the methods and properties of "Column" on page 82.

Fetching BLOB data

You can fetch BLOB data for columns declared BINARY or LONG BINARY. The following example illustrates how you can fetch BLOB data:

```
Dim table as ULTable
Dim col As ULColumn
Dim data(1 to 1024) As Byte
Dim data_fit As Boolean
Dim size As Long

Set table = Conn.GetTable("image")

table.Open
size = 1024
Set col = table.GetColumn("img_data")
data_fit = col.GetBytes(VarPtr(data(1)), size)
If data_fit Then
  'No truncation
Else
  'data truncated at 1024
End if
table.Close
```

Searching for rows with Find and Lookup

UltraLite has several modes of operation when working with data. The ULTable object has two sets of methods for locating particular rows in a table:

- **Find methods** These move to the first row that exactly matches a specified search value, under the index specified when the ULTable object was opened. If the search method cannot be found you are positioned before the first or after the last row.
- **Lookup methods** These move to the first row that matches or is greater than a specified search value, under the index specified when the ULTable object was opened.

Both sets are used in a similar manner:

1. Enter Find or Lookup mode.

   The mode is entered by calling the FindBegin or LookupBegin method, respectively. For example.

   ```
   TCustomer.FindBegin
   ```

2. Set the search values.

   You do this by setting values in the current row. Only values in the columns of the index are relevant to the search.

   Setting these values affects the buffer holding the current row only, and does not affect the database. For example:

   ```
   TCustomer.Column( "lname" ).StringValue = "Kaminski"
   ```

3. Search for the row.

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

   ```
   TCustomer.FindFirst
   ```

   For multi-column indexes, a value for the first column is always used, but you can omit the other columns and you can specify the number of columns as a parameter to FindFirst.

   ☞ For a list of methods, see “Table” on page 140.

Inserting updating, and deleting rows

To update a row in a table, use the following sequence of instructions:

1. Move to the row you wish to update.

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

2. Enter update mode.

   For example, the following instruction enters update mode on TCustomer:

   ```
   TCustomer.UpdateBegin
   ```

3. Set the new values for the row to be updated. For example:

   ```
   TCustomer.Column( "LName" ).StringValue = "Smith"
   ```

4. Execute the Update.

   ```
   TCustomer.Update
   ```

   The update is not carried out until the Update method is called.

After the update operation the current row is the row that was just updated. If you changed the value of a column in the index specified when the ULTable object was opened, the current row is undefined. For more information, see "Update method" on page 148

By default, UltraLite operates in AutoCommit mode, so that the Update is immediately applied to the row in permanent storage. If you have disabled AutoCommit mode, the Update is not made permanent until you execute a Commit operation. For more information, see "Transaction processing in UltraLite" on page 69.

> ***Caution***
> *Updating primary key values can interfere with synchronization. Do not update the primary key of a row: delete the row and add a new row instead.*

Inserting rows

The steps to insert a row are very similar to those for updating rows, except that there is no need to locate any particular row in the table before carrying out the Insert operation. The order of rows in the table has no significance.

Note: The location of the cursor's current row is not defined after an insert. So you should not rely on the current row position after an insert.

The following sequence of instructions inserts a new row:

```
TCustomer.InsertBegin
TCustomer.Column( "Id" ).IntegerValue = 3
TCustomer.Column( "LName" ).StringValue = "Carlo"
TCustomer.Insert
```

If you do not set a value for one of the columns, and that column has a default, the default value is used. If the column has no default, the following entries are added:

- For numeric columns, zero.
- For character columns, an empty string.
- To set a value to NULL, use the ULColumn.SetNull method.

As for Update operations, after calling Insert it is possible to see the newly inserted row, but an Insert is applied to the database in permanent storage itself only when a Commit is carried out. In AutoCommit mode, a Commit is carried out as part of the Insert method.

Deleting rows

The steps to delete a row are simpler than to insert or update rows. There is no Delete mode corresponding to the Insert or Update modes. The steps are as follows:

1. Move to the row you wish to delete.
2. Execute the ULTable.Delete method.

Transaction processing in UltraLite

UltraLite provides transaction processing to ensure the correctness of the data in your database. A transaction is a logical unit of work: it is either all executed or none of it is executed.

By default, UltraLite operates in AutoCommit mode, so that each Insert, Update, or Delete is executed as a separate transaction. Once the operation is completed, the change is made to the database. If you set the ULConnection.AutoCommit property to False, you can use multi-statement transactions. For example, if your application transfers money between two accounts, either both the deduction from the source account and the addition to the destination account must be completed, or neither must be completed.

If AutoCommit is set to False, you must execute a ULConnection.Commit statement to complete a transaction and make changes to your database permanent, or you must execute a ULConnection.Rollback statement to cancel all the operations of a transaction.

Accessing schema information

Objects in the API represent tables, columns, indexes, result sets, and synchronization publications. Each object has a Schema property that provides access to information about the structure of that object.

Here is a summary of the information you can access through the Schema objects.

- **ULDatabaseSchema** The number and names of the tables in the database, as well as global properties such as the format of dates and times.

  To obtain a ULDatabaseSchema object, call the ULConnection.Schema property.

- **ULTableSchema** The number and names of the columns and indexes for this table.

  To obtain a ULTableSchema object, call the ULTable.Schema property.

- **ULColumnSchema** The SQL data type, default value, and other characteristics of the column, such as whether it accepts NULL.

  To obtain a ULTableSchema object, call the ULColumn.Schema property.

- **ULIndexSchema** Information about the type of index and the columns in it. As an index has no data directly associated with it (only that which is in the columns of the index) there is no separate ULIndex object, just a ULIndexSchema object.

  To obtain a ULIndexSchema object, call the ULTableSchema.GetIndex method.

- **ULPublicationSchema** Tables contained in a publication. Publications are also comprised of schema only, and so there is a ULPublicationSchema object rather than a ULPublication object.

  To obtain a ULPublicationSchema object, call the ULDatabaseSchema.GetPublicationSchema method.

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

You cannot modify the schema through the API. You can only retrieve information about the schema. For information about modifying the schema, see “Altering the schema of UltraLite databases” [*UltraLite Database User’s Guide*, page 30].

Error handling

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

☞ For more information, see "SQLCode" on page 124.

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

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

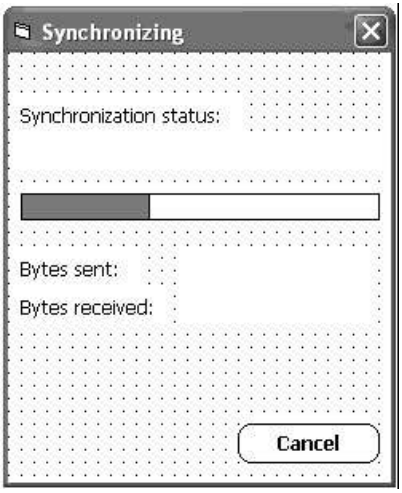
You can then easily access UltraLite errors using the GetError function.

Synchronization

You can synchronize your data if you have SQL Anywhere Studio.

Adding the synchronization template

UltraLite for MobileVB includes a template form that can be used to monitor the status of a synchronization session. A version of this form is included for both Palm OS and Pocket PC. You can use these templates in your application, you can customize them, or you can simply examine them to learn how UltraLite synchronization events work.



❖ **To add one of these templates to your application**

1. From the project menu, select Add Form
2. Select either UltraLite for MobileVB Sync Form (Windows CE) or UltraLite for MobileVB Sync Form (Palm)
3. Click Open

A copy of the form will then be added to your application.

Writing code to use the synchronization form

Call the the InitSyncForm function, passing it your ULConnection object. This must be done before each synchronization. For example, if your synchronization status form is named Form_Sync and your ULConnection object is named Connection:

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

Now, every time your application synchronizes, the synchronization status form appears. As synchronization progresses, your user can observe the progress bar and byte count. When synchronization completes, the form is dismissed. The Cancel button instructs UltraLite to abort the current synchronization.

For more details, see the CustDB example.

Component samples, demonstrations and code fragments

The following are samples, demonstrations and code fragments that you can use.

A data grid control

The following data grid control allows you to use a data grid to work with database information.

- *Samples\UltraLiteForMobileVB\grid\gridsample.vbg*

A customer application

The following application, called custdb, shows you a fully-functional customer application.

- *Samples\UltraLiteForMobileVB\custdb\custdb.vbg*

Maintaining database state on Palm OS

Palm Computing Platform devices allow one application to run at a time. It is common to want to make your application appear as if they never exited when a user switches to another application. For some programs, this can be accomplished simply by storing the current settings; however in a database application, it can be challenging to restart an application and re-open and re-position within previously open result sets. UltraLite provides a way for you to obtain this functionality.

This section describes how you can restore positions within tables so that applications appear to suspend instead of terminate when a user switches to another application. This is accomplished by providing a value for the persistent name parameter in the Open method of the ULTable object.

Databases on the Palm Computing Platform

It is important to distinguish between the Palm database and relational databases such as UltraLite. In this documentation, the term **PDB** means a Palm database and **database** refers to an UltraLite relational database.

UltraLite databases on the Palm Computing Platform

When writing UltraLite applications for the Palm Computing platform, you identify the UltraLite database by the Palm creator ID. UltraLite actually stores database information in multiple PDBs, whose names are constructed using the given creator ID. For example, a database created with a creator ID of ABCD causes the following files to be created:

- *ul_state_ABCD*
- *ul_udb_ABCD*

UltraLite uses the *ul_state_ABCD* PDB to maintain state information about any open cursors or tables when the application exits. **State information** means the current row on which the user is positioned. This is how UltraLite allows you to write your application so that when it is launched, users can resume where they left off.

The state PDB stores a name for the table whose state is being preserved, as well as enough information to restore the table to that state. The name associated with the table may be, but is not required to be, the name of the table. It is called the **persistent name** since it is the name that persists in the PDB.

Using the persistent name

When a request is made to open a table, the user can pass in a persistent name. UltraLite looks up the persistent name in the state PDB to see if there

is a table associated with it. If so, the table is opened and positioned to the proper row. If not, the table is opened and positioned before the first row.

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

Suppose the Connection object is of type ULConnection and a table called ULCustomer exists in the database.

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

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

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

Persistent name notes

- If the persistent name is empty, UltraLite does not store state information for this table, or attempt to look it up when opening the table.

  If you do not need to store the state of your tables, supply an empty persistent name. The state is then not looked up in the state database.

- HotSync synchronization does not affect the state of your open tables. When a user presses the HotSync button on a device, the operating system closes the application in the same way it closes the application when any other application is started. Consequently, the state of the open tables is recorded in the state PDB and when the user returns to the application and the tables are re-opened, the user is positioned on the expected row. If that row has been deleted as part of the synchronization, the user is positioned on the next row (or after the last row if it was the last row).

- Applications with auto-commit turned off could terminate with uncommitted transactions. UltraLite maintains these transactions so that when the application restarts, they are not lost.

- If UltraLite finds a table in the state PDB that matches the persistent name you have provided, it checks that the table and index are the same as the table and index used when the position information was recorded. If they are not, then the call to Open fails.

- The use of the persistent name is unique to the Palm OS. If you create UltraLite for MobileVB applications for Windows CE, they do not use the persistent name. Applications on Windows CE run more like they do on a desktop machine.

Why a separate persistent name is needed

It is possible with UltraLite to have the same table open multiple times and at the same time in your application. In this case, the table name is not unique enough to store in the state PDB. In an application that does this, the code would look something like the following:

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

Closing connections and tables

If you explicitly close a ULTable object (by calling its Close method), the current row is not maintained. If you explicitly close a ULConnection object, all the ULTable objects are implicitly closed, so the current row for all the tables is not maintained.

The current state is only stored in the state PDB for tables that have not been closed, and when the ULConnection object's Close method is not called. This means that the application should terminate without ever calling Close on the ULConnection object; or, the routine that defined the ULConnection object is exited; or, the variable for the ULConnection object is assigned to Nothing.

For applications with auto-commit turned off, closing a connection rolls back any uncommitted transactions. By not closing the ULConnection object, the outstanding transactions are saved (not committed), so that when the application restarts, those transactions will appear and can be committed or rolled back. Also, uncommitted changes are not synchronized.

Example: Using the persistent name to maintain state information

The PersistentName example program is a relatively simple program that shows how to use maintained state information. It is available at *http://www.sybase.com/detail?id=1022734*. Here are some highlights from the sample:

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

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

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

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

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

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

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

CHAPTER 6

UltraLite for MobileVB API Reference

About this chapter

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

Contents

Topic:	page
ULAuthStatusCode	81
ULColumn class	82
ULColumnSchema class	88
ULConnection class	89
ULConnectionParms class	97
ULDatabaseManager class	100
ULDatabaseSchema class	106
ULIndexSchema class	109
ULPreparedStatement class	111
ULPublicationSchema class	116
ULResultSet class	117
ULResultSetSchema class	123
ULSQLCode enumeration	124
ULSQLType enumeration	128
ULStreamErrorCode enumeration	129
ULStreamErrorContext enumeration	132
ULStreamErrorID enumeration	133
ULStreamType enumeration	134
ULSyncParms class	135

Topic:	page
ULSyncResult class	138
ULSyncState enumeration	139
ULTable class	140
ULTableSchema class	149

ULAuthStatusCode

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

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

ULColumn class

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

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

☞ For information about the table object, see "ULTable class" on page 140.

Properties

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

Prototype	Description
UUIDValue As String	Gets or sets the value of this column as a UUID. UltraLite stores universally unique identifiers (UUID's) as binary values. The column must be of binary type and able to store at least 16 bytes. When getting this property, UltraLite converts the column value to a string representation of the UUID. If the value is not a valid UUID, a SQLE_CONVERSION_ERROR is raised. When setting this property, UltraLite converts the string form of the UUID to a binary value before storing it in the database.

AppendByteChunk method

Prototype

AppendByteChunk(_
data As Long**, _**
data_len As Long **_**
)
Member of **UltraLiteAFLib.ULColumn**

Description

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

Parameters

data A pointer to an array of bytes. To get the pointer to the array of bytes, use the Visual Basic VarPtr() function.

data_len The number of bytes from the array to append.

Errors set

Error	Description
ulSQLE_INVALID_PARAMETER	The error occurs if data length is less than 0
ulSQLE_CONVERSION_ERROR	The error occurs if the column data type is not LONG BINARY

Example

In the following example, 512 bytes of data are appended to the edata column.

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

AppendStringChunk method

Prototype

AppendStringChunk(*chunk* As String **)**
Member of **UltraLiteAFLib.ULColumn**

Description

Appends the string to the column if the type is TypeLongString or TypeString.

Parameters

data A string to append to the existing string in a table.

Errors set

Error	Description
ulSQLE_CONVERSION_ERROR	The error occurs if the column data type is not CHAR or LONG VARCHAR.

GetByteChunk method

Prototype

GetByteChunk (_
offset As Long**, _**
data As Long**, _**
data_len As Long**, _**
filled_len As Long _
) As Boolean
Member of **UltraLiteAFLib.ULColumn**

Description

Gets data from a TypeBinary or TypeLongBinary column.

Parameters

offset The offset into the underlying array of bytes. The source offset must be greater than or equal to 0, otherwise a ulSQLE_INVALID_PARAMETER error will be raised.

data A pointer to an array of bytes. To get the pointer to the array of bytes, use the Visual Basic VarPtr() function.

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

filled_len This is an OUT parameter. After the method is called, it indicates how many bytes were fetched with valid data. If the size of BLOB data is unknown in advance, it is fetched using a fixed-length chunk - one chunk at a time. The last chunk fetched can be smaller than chunk size, so filled_len informs how many bytes of valid data exist in the buffer.

Returns

True if this column value contains more data

False if there is no more data for this column in the database.

Errors set

Error	Description
ulSQLE_CONVERSION_-ERROR	The error occurs if the column data type isn't BINARY or LONG BINARY.
ulSQLE_INVALID_-PARAMETER	The error occurs if the column data type is BINARY and the offset is not 0 or 1, or, the data length is less than 0. The error also occurs if the column data type is LONG BINARY and the offset is less than 1 .

Example

In the following example, edata is a column name. If the *data_len* parameter passed in is larger than the actually array size, a General Protection Fault occurs.

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

GetStringChunk method

Prototype

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

Description

Gets data from a TypeString or TypeLongString column.

Parameters

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

data The variable to receive the string data.

string_length The length of the String you want returned.

filled_len The length of the String fetched.

Returns

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

False if there is no more data.

Errors

Error	Description
ulSQLE_CONVERSION_-ERROR	The error occurs if the column data type isn't CHAR or LONG VARCHAR.
ulSQLE_INVALID_-PARAMETER	The error occurs if the column data type is CHAR and the src_offset is greater than 64K.
ulSQLE_INVALID_-PARAMETER	The error occurs if src_offset is less than 0 or string length is less than 0.

SetByteChunk method

Prototype

SetByteChunk (_
data As Long**, _**
length As Long _
)
Member of **UltraLiteAFLib.ULColumn**

Description

Sets data in a TypeBinary or TypeLongBinary column.

Parameters

data A pointer to an array of bytes. To get the pointer to the array of bytes, use the Visual Basic VarPtr() function.

length The length of the array.

Errors set

Error	Description
ulSQLE_CONVERSION_ERROR	The error occurs if the column data type is not BINARY or LONG BINARY.
ulSQLE_INVALID_PARAMETER	The error occurs if the data length is less than 0.
ulSQLE_INVALID_PARAMETER	The error occurs if the data length is greater than 64K.

Example

In the following example, edata is a column name and the first 232 bytes of the data variable are stored in the database.

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

SetNull method

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

SetToDefault method

Prototype	**SetToDefault( )** Member of **UltraLiteAFLib.ULColumn**
Description	Sets the current column to its default value as defined by the database schema. For example, an autoincrement column will be assigned the next available value.

ULColumnSchema class

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

Properties

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

ULConnection class

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

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

Example

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

```
Public WithEvents Connection As ULConnection
```

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

Properties

The following are properties of ULConnection:

Prototype	Description
AutoCommit As Boolean	Indicates the AutoCommit value. If true, all data changes are committed immediately after they are made. Otherwise, changes are not committed to the database until Commit is called. By default, this property is True.
DatabaseID As Long (write-only)	Sets the identification number for the connected database. When you write the DatabaseID, you set the database ID value to be used for global autoincrement columns.
GlobalAutoIncrementUsage As Integer (read-only)	Gets the percentage of available global autoincrement values that have been used.
LastIdentity As Long (read-only)	Gets the most recent value inserted into a column with a default of autoincrement or global autoincrement.
OpenParms As String (read-only)	Gets the string used to open the connection to the database.

Prototype	Description
Schema As ULDatabaseSchema (read-only)	Gets the ULDatabaseSchema object which represents the definition of the database.
SQLErrorOffset As Integer (read-only)	If PrepareStatement raises an error, indicates the 1-based offset in the SQL statement where the error was noted. If this value is less than or equal to 0, no offset information is available.

CancelSynchronize method

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

Description
When called during synchronization, the method cancels the synchronization. The user can only call this method during one of the synchronization events.

To allow this the ULConnection object must be declared **WithEvents**.

ChangeEncryptionKey method

Prototype
ChangeEncryptionKey(*newkey* As String **)**
Member of **UltraLiteAFLib.ULConnection**

Description
Encrypt the database with the specified key.

Parameters
newkey The new encryption key value for the database.

Example
When you call CreateDatabaseWithParms and pass in the parms object, with a value in place for EncryptionKey, the database is created with encryption. Another way to change the encryption key is by specifying the new encryption key on the ULConnection object. In this example, “apricot” is the key.

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

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

Close method

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

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

Commit method

Prototype — **Commit( )**
Member of **UltraLiteAFLib.ULConnection**

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

For more information, see Autocommit under ULConnection "Properties." on page 89

CountUploadRows method

Prototype —
```
CountUploadRows(
  [ mask As Long = 0 ], _
  [ threshold As Long = -1 ] _
) As Long
```
Member of **UltraLiteAFLib.ULConnection**

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

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

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

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

GetNewUUID method

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

Description — Returns a new universally unique identifier in a string format. This string is of the form *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*.

Returns — Each call returns a new UUID.

GetTable method

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

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

Parameters
name The name of the table sought.

Returns
Returns the ULTable object.

GrantConnectTo method

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

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

Parameters
userid The user ID being granted authority to connect.

password The password the user ID must specify for connecting.

LastDownloadTime method

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

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

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

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

OnReceive event

Prototype
OnReceive(
nBytes As Long**,** _
nInserts As Long**,** _
nUpdates As Long**,** _
nDeletes As Long _
)
Member of **UltraLiteAFLib.ULConnection**

Description
Reports download information to the application from the consolidated

database via MobiLink. This event may be called several times.

Parameters

nBytes Cumulative count of bytes received at the remote application from the consolidated database.

nInserts Cumulative count of inserts received at the remote application from the consolidated database.

nUpdates Cumulative count of updates received at the remote application from the consolidated database.

nDeletes Cumulative count of deletes received at the remote application from the consolidated database.

Example

See the CustDB application for an example of this method.

OnSend event

Prototype

```
OnSend(
  nBytes As Long, _
  nInserts As Long, _
  nUpdates As Long, _
  nDeletes As Long _
)
```

Member of **UltraLiteAFLib.ULConnection**

Description

Reports upload information from the remote database via MobiLink to the consolidated database. This event may be called several times.

Parameters

nBytes Cumulative count of bytes sent by the remote application to the consolidated database via MobiLink.

nInserts Cumulative count of inserts sent by the remote application to the consolidated database via MobiLink.

nUpdates Cumulative count of updates sent by the remote application to the consolidated database via MobiLink.

nDeletes Cumulative count of deletes sent by the remote application to the consolidated database via MobiLink.

Example

See the CustDB application for an example of this method.

OnStateChange event

Prototype

```
OnStateChange(
  newState As ULSyncState, _
  oldState As ULSyncState _
)
```

Member of **UltraLiteAFLib.ULConnection**

Description	This event is called whenever the state of the synchronization changes. For more information, see "ULSyncState enum" on page 139.
Parameters	**newState** The state that the synchronization process is about to enter.
	oldState The state that the synchronization process just completed.
Example	See the CustDB application for an example of this method.

OnTableChange event

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

PrepareStatement method

Prototype	**PrepareStatement(** *sqlStatement* As String**,** _ *persistent_name* As String _ **)** As ULPreparedStatement Member of **UltraLiteAFLib.ULConnection**
Description	Prepares a SQL statement for execution.
Parameters	**sqlStatement** The SQL statement to prepare.
	persistent_name For Palm applications, the persistent name of the statement.
Returns	Returns a ULPreparedStatement. If there was a problem preparing the statement, an error will be raised. The offset into the statement where the error occurred can be determined from the SQLErrorOffset property.

ResetLastDownloadTime method

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

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

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

RevokeConnectFrom method

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

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

Parameters — **userid** The user ID for the user to be revoked.

Rollback method

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

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

StartSynchronizationDelete method

Prototype — **StartSynchronizationDelete( )**
Member of **UltraLiteAFLib.ULConnection**

Description — Once StartSynchronizationDelete is called, all delete operations are again synchronized.

StopSynchronizationDelete method

Prototype — **StopSynchronizationDelete( )**
Member of **UltraLiteAFLib.ULConnection**

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

StringToUUID method

Prototype —
StringToUUID(
s_uuid As String**, _**
buffer_16_bytes As Long _
)
Member of **UltraLiteAFLib.ULConnection**

Description — Converts the universally unique identifier represented as a String in the form xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx to a Byte array of 16 bytes. In a

MobileVB application, it may be useful to refer to them in their string format. Consequently, the UUIDValue property on the ULColumn object converts from string to binary(16) and vice versa. The StringToUUID function is provided as an easy way to convert a MobileVB String to a Byte array. It does not reference the UltraLite database in any way.

> **A note on the pointer to the buffer:**
> The pointer to the buffer must be declared as at least16 bytes. Since Visual Basic does not provide bounds checking, memory could be overwritten if the buffer is too small. Use the VarPtr() function to get the pointer to the buffer. See also ULColumn.UUIDValue property

Parameters

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

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

Example

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

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

Synchronize method

Prototype

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

Description

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

UUIDToString method

Prototype

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

Description

Converts a UUID from a byte array to a string of the form *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*.

Parameters

buffer_16_bytes An array of 16 bytes containing a UUID.

Returns

Each call returns a string of the form *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*

ULConnectionParms class

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

Properties

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

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

> **Note**
> Databases are created with a single authenticated user, DBA, whose initial password is SQL. By default, connections are opened using the user ID DBA and password SQL.

☞ For more information about the meaning of these parameters, see "Connection Parameters" [*UltraLite Database User's Guide*, page 49].

Prototype	Description
AdditionalParms As String (read-write)	Additional parameters specified as **name**=*value* pairs separated with semi-colons. ☞ See "Additional Parms connection parameter" [*UltraLite Database User's Guide*, page 65].
CacheSize As Integer (read-write)	The size of the cache. CacheSize values are specified in bytes. Use the suffix k or K for kilobytes and use the suffix m or M for megabytes. The default cache size is sixteen pages. Given a default page size of 4 KB, the default cache size is 64 KB. ☞ See "Cache Size connection parameter" [*UltraLite Database User's Guide*, page 66].

Prototype	Description
ConnectionName As String (read-write)	A name for the connection. This is needed only if you create more than one connection to the database. ☞ See "Connection Name connection parameter" [*UltraLite Database User's Guide*, page 60].
DatabaseOnCE As String (read-write)	The filename of the database deployed to PocketPC. ☞ See "Database On CE connection parameter" [*UltraLite Database User's Guide*, page 54].
DatabaseOnDesktop As String (read-write)	The filename of the database during development. ☞ See "Database On Desktop connection parameter" [*UltraLite Database User's Guide*, page 55].
DatabaseOnPalm As String (read-write)	The UltraLite database on the Palm device. ☞ See "Database On Palm connection parameter" [*UltraLite Database User's Guide*, page 56].
EncryptionKey As String (read-write)	A key for encrypting the database. OpenConnection and OpenConnectionWithParms must use the same key as specified during database creation. Suggestions for keys are: 1. Select an arbitrary, lengthy string 2. Select strings with a variety of numbers, letters and special characters, so as to decrease the chances of key penetration. ☞ See "Encryption Key connection parameter" [*UltraLite Database User's Guide*, page 63].
PageSize As Integer (read-write)	The page size for the database. ☞ See "Page Size connection parameter" [*UltraLite Database User's Guide*, page 67].
ParmsUsed As String (read-only)	The parameters used by the ULDatabaseManager. Useful for debugging purposes.

Prototype	Description
Password As String (read-write)	The password for an authenticated user. Databases are initially created with one authenticated user password *SQL*. Passwords are case-insensitive if the database is case-insensitive and case-sensitive if the database is case-sensitive. The default value is *SQL*. ☞ See "Password connection parameter" [*UltraLite Database User's Guide*, page 58].
ReserveSize As Integer (read-write)	The amount of file system space to reserve for storage of UltraLite persistent data. ☞ See "Reserve Size connection parameter" [*UltraLite Database User's Guide*, page 68].
SchemaOnCE As String (read-write)	The schema filename deployed to PocketPC. ☞ See "Schema On CE connection parameter" [*UltraLite Database User's Guide*, page 61].
SchemaOnDesktop As String (read-write)	The schema filename during development. ☞ See "Schema On Desktop connection parameter" [*UltraLite Database User's Guide*, page 62].
SchemaOnPalm As String (read-write)	The schema PDB on the Palm device. ☞ See "Schema On Palm connection parameter" [*UltraLite Database User's Guide*, page 63].
UserID As String (read-write)	The authenticated user for the database. Databases are initially created with one authenticated user DBA. The UserID is case-insensitive if the database is case-insensitive and case-sensitive if the database is case-sensitive. The default value is *DBA*. ☞ See "User ID connection parameter" [*UltraLite Database User's Guide*, page 59].
VFSOnPalm As Boolean (read-write)	Indicates whether the Palm database is on a virtual file system (true) or on the Palm store (false). ☞ See "VFS On Palm parameter" [*UltraLite Database User's Guide*, page 56].

ULDatabaseManager class

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

Parms or no parms?
Two types of methods exist for creating, opening and dropping connections to your database: Methods WithParms and methods that do not use the ULConnectionParms object. Methods WithParms allow you to use a ULConnectionParms object to manipulate connection parameters with ease and accuracy. Methods that do not use the ULConnectionParms object require that you can successfully create a connections string and use that connection string in a CreateDatabase, OpenConnection or DropDatabase method.

Properties

The following are properties of ULDatabaseManager:

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

CreateDatabase method

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

Prototype

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

Description

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

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

☞ For more information about ApplyFile, see "ULDatabaseSchema class" on page 106 and "ApplyFile method" on page 107.

Parameters

parms A semicolon-separated list of database creation parameters.

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

☞ For information about connection parameters, see "Connection Parameters" [*UltraLite Database User's Guide*, page 49].

☞ For more information about the Palm_fs parameter, see "palm_fs parameter" [*UltraLite Database User's Guide*, page 56] .

Returns

Returns a connection to a newly created UltraLite database.

Examples

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

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

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

On Error Resume Next

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

☞ For information about connection parameters, see "OpenConnection method" on page 103.

CreateDatabaseWithParms method

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

Prototype

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

Description

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

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

Parameters

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

> **Note for VFS card for Palm users**
> You specify VFSOnPalm in the ULConnectionParms interface.

☞ For more information about the Palm_fs parameter, see “palm_fs parameter” [*UltraLite Database User’s Guide*, page 56] .

Returns

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

Examples

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

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

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

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

DropDatabase method

The DropDatabase method deletes a database file.

Prototype **DropDatabase(** *parms* As String **)**
Member of

Description Deletes the database file. All information in the database file is lost. Fails if the specified database does not exist, or if there exist open connections at the time of DropDatabase is executed.

Parameters **parms** The filename for the database.

Example The following example drops a database:

```
Dim parms As String
parms = "PALM_DB=Syb1;NT_FILE=c:\temp\ul_CustDB.udb"
DropDatabase(parms)
```

DropDatabaseWithParms method

The DropDatabaseWithParms method deletes a database file.

Prototype **DropDatabaseWithParms(** *parms* As ULConnectionParms **)**
Member of

Description Deletes the database file. All information in the database file is lost.

Parameters **parms** The ULConnectionParms object containing vital connection parameters .

Example The following example assumes you have declared and instantiated a ULConnectionParms object named **LoginParms** and used it to specify the database location.

```
DatabaseMgr.DropDatabaseWithParms LoginParms
```

OpenConnection method

Prototype **OpenConnection(** *connparms* As string **)** As ULConnection
Member of **UltraLiteAFLib.ULDatabaseManager**

Description If a database exists, use this method to connect to the database. If a database does not exist, or the connection parameters are invalid, the call will fail. Use the error object to determine why the call failed.

The function returns a ULConnection object which provides an open connection to a specified UltraLite database. The database filename is specified using the connparms string. Parameters are specified using a sequence of **name**=*value* pairs. If no user ID or password is given, the default is used.

It should contain a value of the form

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

Parameters

connparms The parameter used to establish a connection to a database. Parameters are specified as a semicolon separated list of **keyword=***value* pairs. If no user ID or password is given, the default is used.

> **Note for Palm users**
> The Palm_fs=vfs parameter needs to be specified both for CreateDatabase and OpenConnection methods when using a database on the Palm virtual file system.

Returns

☞ For more information about the Palm_fs parameter, see “palm_fs parameter” [*UltraLite Database User’s Guide*, page 56] .

The ULConnection object is returned if the connection was successful.

Example

The following example creates a new database connection from the CustDB sample application:

```
Set Connection = DatabaseMgr.OpenConnection(
"file_name=d:\Dbfile.udb;palm_db=Syb3;CE_file=\myapp\MyDB.udb")
```

OpenConnectionWithParms method

Prototype

OpenConnectionWithParms(*connparms* As ULConnection-Parms**)** As ULConnection
Member of **UltraLiteAFLib.ULDatabaseManager**

Description

If a database exists, use this method to receive a connection. If a database does not exist, or the connection parameters are invalid, the call will fail. Use the error object to determine why the call failed.

The function returns a ULConnection object which provides an open connection to a specified UltraLite database. The database filename is specified using the connparms object. Parameters are specified using a sequence of **name**=*value* pairs. If no user ID or password is given, the default is used.

Parameters

connparms The parameters defining this connection.

Returns

The ULConnection object is returned if the connection was successful.

Example

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

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

ULDatabaseSchema class

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

Properties

The following are properties of ULDatabaseSchema:

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

ApplyFile method

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

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

> **Caution**
> ApplyFile is very safe in the hands of an informed programmer. Do not delete columns unthinkingly unless you are willing to accept data loss, as data loss can occur under a number of circumstances including (1) if columns are deleted, or (2) if the data type for a column is changed to an incompatible type or (3) if you upgrade an 8.0.2 database using ApplyFile in UltraLite 9.0.

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

Example

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

ApplyFileWithParms method

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

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

> **Caution**
> ApplyFileWithParms is very safe in the hands of an informed programmer. Do not delete columns unthinkingly unless you are willing to accept data loss, as data loss can occur under a number of circumstances including (1) if columns are deleted, or (2) if the data type for a column is changed to an incompatible type or (3) if you upgrade an 8.0.2 database using ApplyFile in UltraLite 9.0.

Parameters
parms The object identifying the schema file to apply.

GetPublicationName method

Prototype	**GetPublicationName(** *id* As Integer **)** As String Member of **UltraLiteAFLib.ULDatabaseSchema**
Description	Returns the name of the specified publication. The publication *ID* can range from 1 to PublicationCount.
Parameters	**id** The *id* is the identifier of the publication whose name will be returned.
Returns	Returns the name of a publication in the connected database. For information about the ULPublicationSchema object, see "ULPublicationSchema" on page 116. For more information, see ULDatabaseSchema "Properties" on page 106

GetPublicationSchema method

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

GetTableName method

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

ULIndexSchema class

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

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

Properties

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

GetColumnName method

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

IsColumnDescending method

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

ULPreparedStatement class

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

Properties

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

AppendByteChunkParameter method

Prototype

AppendByteChunkParameter (
param_id As Integer**,**
data As Long**,**
data_len As Long**)**
Member of **UltraLiteAFLib.ULPreparedStatement**

Description

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

Parameters

parameter_id The 1-based parameter number to set.

data The array of bytes to be appended.

data_len The number of bytes from the array to append.

Errors set

Error	Description
ulSQLE_INVALID_PARAMETER	The error occurs if the data length is less than 0.
ulSQLE_CONVERSION_ERROR	The error occurs if the column data type is not LONG BINARY

AppendStringChunkParameter method

Prototype

AppendStringChunkParameter(
param_id As Integer **,**
chunk As String **)**
Member of **UltraLiteAFLib.ULPreparedStatement**

Description

Appends the string to the column if the type is ulTypeLongString.

Parameters

parameter_id The 1-based parameter number to set.

chunk A string to append to the existing string in a table.

Errors set

Error	Description
ulSQLE_CONVERSION_ERROR	The error occurs if the column data type is not LONG VARCHAR .

Close method

Prototype

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

Description

Frees resources associated with the ULPreparedStatement.

ExecuteQuery method

Prototype

ExecuteQuery( ) As ULResultSet
Member of **UltraLiteAFLib.ULPreparedStatement**

Description

Executes the query and returns a result set.

Returns

A ULResultSet object. The ULResultSet is the data you requested in your SELECT statement. To describe the product of your query, see "ULResultSetSchema." on page 123

ExecuteStatement method

Prototype
ExecuteStatement( ) As Long
Member of **UltraLiteAFLib.ULPreparedStatement**

Description
Executes the statement.

Returns
The number of rows updated.

SetBooleanParameter method

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

Description
Set the parameter to the Boolean value passed in.

Parameters
param_number The 1-based parameter number to set.

param_value The value the parameter should receive.

SetByteChunkParameter method

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

Description
Sets data in a binary or long binary column.

Parameters
param_number The 1-based parameter number to set.

data An array of bytes.

data_len The number of bytes from the array to append.

SetByteParameter method

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

Description
Set the parameter to the Byte value passed in.

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

param_value The value the parameter should receive.

SetDatetimeParameter method

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

Description Set the parameter to the Datetime value passed in.

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

param_value The value the parameter should receive.

SetDoubleParameter method

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

Description Set the parameter to the Double value passed in.

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

param_value The value the parameter should receive.

SetIntegerParameter method

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

Description Set the parameter to the Integer value passed in.

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

param_value The value the parameter should receive.

SetLongParameter method

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

Description
Set the parameter to the Long value passed in.

Parameters
param_number The 1-based parameter number to set.

param_value The value the parameter should receive.

SetNullParameter method

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

Description
Set the parameter to NL.

Parameters
parameter_id The 1-based parameter number to set.

SetRealParameter method

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

Description
Set the parameter to the Long value passed in.

Parameters
param_number The 1-based parameter number to set.

param_value The value the parameter should receive.

SetStringParameter method

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

Description
Set the parameter to the string passed in.

Parameters
param_number The 1-based parameter number to set.

param_value The value the parameter should receive.

ULPublicationSchema class

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

Properties

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

ContainsTable method

Prototype
ContainsTable(*name* As String **)** As Boolean
Member of **UltraLiteAFLib.ULPublicationSchema**

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

Parameters
name The target table name.

Returns
True if the table is in the publication.

False if the table is not in the publication.

ULResultSet class

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

Properties

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

Close method

Prototype

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

Description

Frees all resources associated with this object.

GetByteChunk method

Prototype

GetByteChunk (_
index As Integer**,** _
src_offset As Long**,** _
data As Long**,** _
data_len As Long**,** _
filled_len As Long _
) As Boolean
Member of **UltraLiteAFLib.ULResultSet**

Description

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

Parameters

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

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

data A pointer to an array of bytes. To get the pointer to the array of bytes, use the Visual Basic VarPtr() function.

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

filled_len The number of bytes fetched. Because you don't know how big the BLOB data is in advance, you generally fetch it using a fixed-length chunk, one chunk at a time. The last chunk may be smaller than your chunk size. filled_len reports how many bytes were actually fetched.

Returns

The number of bytes read.

Errors set

Error	Description
ulSQLE_-CONVERSION_-ERROR	The error occurs if the column data type is not BINARY or LONG BINARY
ulSQLE_INVALID_-PARAMETER	The error occurs if the column data type is BINARY and the offset is not 0 or 1, or, the data length is less than 0. The error also occurs if the column data type is LONG BINARY and the offset is less than 1.

Example

In the following example, edata is a column name. If the *data_len* parameter passed in is not sufficiently long, the entire application will terminate.

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

GetStringChunk method

Prototype

GetStringChunk(_
index As Integer**,** _
offset As Long**,** _
data As String**,** _
string_len As Long**,** _
filled_len As Long _
) As Boolean
Member of **UltraLiteAFLib.ULResultSet**

Description — Fills the string passed in with the binary data in the column. Suitable for Long Varchars.

Parameters

index The 1-based column ID of the target column.

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

data The data string.

string_len The length of the string you want returned.

filled_len The length of the string filled.

Returns — Gets BLOB data from a binary or long binary column.

Errors set

Error	Description
ulSQLE_-CONVERSION_-ERROR	The error occurs if the column data type is not CHAR or LONG VARCHAR
ulSQLE_INVALID_-PARAMETER	The error occurs if the column data type is CHAR and the src_offset is greater than 64K
ulSQLE_INVALID_-PARAMETER	The error occurs if offset is less than 0 or string length is less than 0

MoveAfterLast method

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

Description — Moves to a position after the last row of the ULResultSet.

MoveBeforeFirst method

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

Description — Moves to a position before the first row.

MoveFirst method

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

Description — Moves to the first row.

Returns — **True** if successful.

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

MoveLast method

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

MoveNext method

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

MovePrevious method

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

MoveRelative method

Prototype	**MoveRelative(** *index* As Long **)** As Boolean Member of **UltraLiteAFLib.ULResultSet**
Description	Moves a certain number of rows relative to the current row. Relative to the current position of the cursor in the resultset, positive index values move forward in the resultset, negative index values move backward in the resultset and zero does not move the cursor.
Parameters	**index** The number of rows to move. The value can be positive, negative, or zero.
Returns	**True** if successful. **False** if unsuccessful. The method fails, for example, if there are no rows.

IsNull method

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

GetDatetime method

Prototype	**GetDatetime(** *index* As Integer **)** As Date Member of **UltraLiteAFLib.ULResultSet**
Description	Gets the column value as an Date.
Parameters	**index** The 1-based ordinal in the result set to get.
Returns	The value as a Date.

GetDouble method

Prototype	**GetDouble(** *index* As Integer **)** As Double Member of **UltraLiteAFLib.ULResultSet**
Description	Gets the column value as a Double.
Parameters	**index** The 1-based ordinal in the result set to get.
Returns	The value as a Double.

GetInteger method

Prototype	**GetInteger(** *index* As Integer **)** As Integer Member of **UltraLiteAFLib.ULResultSet**
Description	Gets the column value as an Integer.
Parameters	**index** The 1-based ordinal in the result set to get.
Returns	The value as an Integer.

GetLong method

Prototype	**GetLong(** *index* As Integer **)** As Long Member of **UltraLiteAFLib.ULResultSet**
Description	Gets the column value as a Long.
Parameters	**index** The 1-based ordinal in the result set to get.

Returns	The value as a Long.

GetReal method

Prototype	**GetReal(** *index* As Integer **)** As Single Member of **UltraLiteAFLib.ULResultSet**
Description	Gets the column value as a Single.
Parameters	**index** The 1-based ordinal in the result set to get.
Returns	The value as a Real.

GetString method

Prototype	**GetString(** *index* As Integer **)** As String Member of **UltraLiteAFLib.ULResultSet**
Description	Gets the column value as a String.
Parameters	**index** The 1-based ordinal in the result set to get.
Returns	The value as a String.

ULResultSetSchema class

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

Properties

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

ULSQLCode enumeration

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

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

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

Constant	Value
ulSQLE_DATABASE_UPGRADE_FAILED	-672
ulSQLE_DATABASE_UPGRADE_NOT_-POSSIBLE	-673
ulSQLE_DATATYPE_NOT_ALLOWED	-624
ulSQLE_DBSPACE_FL	-604
ulSQLE_DIV_ZERO_ERROR	-628
ulSQLE_DOWNLOAD_CONFLICT	-839
ulSQLE_DROP_DATABASE_FAILED	-651
ulSQLE_DYNAMIC_MEMORY_EXHAUSTED	-78
ulSQLE_ENGINE_ALREADY_RUNNING	-96
ulSQLE_ENGINE_NOT_MTIUSER	-89
ulSQLE_ERROR	-300
ulSQLE_ERROR_CALLING_FUNCTION	-622
ulSQLE_EXPRESSION_ERROR	-156
ulSQLE_IDENTIFIER_TOO_LONG	-250
ulSQLE_INDEX_NOT_FOUND	-183
ulSQLE_INDEX_NOT_UNIQUE	-196
ulSQLE_INTERRUPTED	-299
ulSQLE_INVALID_AGGREGATE_-PLACEMENT	-862
ulSQLE_INVALID_FOREIGN_KEY	-194
ulSQLE_INVALID_FOREIGN_KEY_DEF	-113
ulSQLE_INVALID_GROUP_SELECT	-149
ulSQLE_INVALID_LOGON	-103
ulSQLE_INVALID_OPTION_SETTING	-201
ulSQLE_INVALID_ORDER	-152
ulSQLE_INVALID_ORDERBY_COLUMN	-854
ulSQLE_INVALID_PARAMETER	-735

Constant	Value
ulSQLE_INVALID_SQL_IDENTIFIER	-760
ulSQLE_INVALID_STATEMENT	-130
ulSQLE_LOCKED	-210,
ulSQLE_MEMORY_ERROR	-309
ulSQLE_METHOD_CANNOT_BE_CALLED	-669
ulSQLE_NAME_NOT_UNIQUE	-110
ulSQLE_NOERR	0
ulSQLE_NOTFOUND	100
ulSQLE_NOT_IMPLEMENTED	-134
ulSQLE_NO_CURRENT_ROW	-197
ulSQLE_NO_INDICATOR	-181
ulSQLE_OVERFLOW_ERROR	-158
ulSQLE_PERMISSION_DENIED	-121
ulSQLE_PRIMARY_KEY_NOT_UNIQUE	-193
ulSQLE_PRIMARY_KEY_VALUE_REF	-198
ulSQLE_PUBLICATION_NOT_FOUND	-280
ulSQLE_RESOURCE_GOVERNOR_-EXCEEDED	-685
ulSQLE_ROW_DROPPED_DURING_-SCHEMA_UPGRADE	130
ulSQLE_SERVER_SYNCHRONIZATION_-ERROR	-857
ulSQLE_START_STOP_DATABASE_DENIED	-75
ulSQLE_STATEMENT_ERROR	-132
ulSQLE_SYNTAX_ERROR	-131
ulSQLE_STRING_RIGHT_TRUNCATION	-638
ulSQLE_TABLE_HAS_PUBLICATIONS	-281
ulSQLE_TABLE_IN_USE	-214
ulSQLE_TABLE_NOT_FOUND	-141

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

ULSQLType enumeration

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

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

ULStreamErrorCode enumeration

The ULStreamErrorCode constants identify constants you can use to specify the ULStreamErrorCode.

Constant	Value
ulStreamErrorCodeNone	0
ulStreamErrorCodeParameter	1
ulStreamErrorCodeParameterNotUint32	2
ulStreamErrorCodeParameterNotUint32Range	3
ulStreamErrorCodeParameterNotBoolean	4
ulStreamErrorCodeParameterNotHex	5
ulStreamErrorCodeMemoryAllocation	6
ulStreamErrorCodeParse	7
ulStreamErrorCodeRead	8
ulStreamErrorCodeWrite	9
ulStreamErrorCodeEndWrite	10
ulStreamErrorCodeEndRead	11
ulStreamErrorCodeNotImplemented	12
ulStreamErrorCodeWouldBlock	13
ulStreamErrorCodeGenerateRandom	14
ulStreamErrorCodeInitRandom	15
ulStreamErrorCodeSeedRandom	16
ulStreamErrorCodeCreateRandomObject	17
ulStreamErrorCodeShuttingDown	18
ulStreamErrorCodeDequeuingConnection	19
ulStreamErrorCodeSecureCertificateRoot	20
ulStreamErrorCodeSecureCertificateCompanyName	21
ulStreamErrorCodeSecureCertificateChainLength	22
ulStreamErrorCodeSecureCertificateRef	23
ulStreamErrorCodeSecureCertificateNotTrusted	24

Constant	Value
ulStreamErrorCodeSecureDuplicateContext	25
ulStreamErrorCodeSecureSetIo	26
ulStreamErrorCodeSecureSetIoSemantics	27
ulStreamErrorCodeSecureCertificateChainFunc	28
ulStreamErrorCodeSecureCertificateChainRef	29
ulStreamErrorCodeSecureEnableNonBlocking	30
ulStreamErrorCodeSecureSetCipherSuites	31
ulStreamErrorCodeSecureSetChainNumber	32
ulStreamErrorCodeSecureCertificateFileNotFound	33
ulStreamErrorCodeSecureReadCertificate	34
ulStreamErrorCodeSecureReadPrivateKey	35
ulStreamErrorCodeSecureSetPrivateKey	36
ulStreamErrorCodeSecureCertificateExpiryDate	37
ulStreamErrorCodeSecureExportCertificate	38
ulStreamErrorCodeSecureAddCertificate	39
ulStreamErrorCodeSecureTrustedCertificateFileNotFound	40
ulStreamErrorCodeSecureTrustedCertificateRead	41
ulStreamErrorCodeSecureCertificateCount	42
ulStreamErrorCodeSecureCreateCertificate	43
ulStreamErrorCodeSecureImportCertificate	44
ulStreamErrorCodeSecureSetRandomRef	45
ulStreamErrorCodeSecureSetRandomFunc	46
ulStreamErrorCodeSecureSetProtocolSide	47
ulStreamErrorCodeSecureAddTrustedCertificate	48
ulStreamErrorCodeSecureCreatePrivateKeyObject	49
ulStreamErrorCodeSecureCertificateExpired	50
ulStreamErrorCodeSecureCertificateCompanyUnit	51
ulStreamErrorCodeSecureCertificateCommonName	52

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

ULStreamErrorContext enumeration

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

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

ULStreamErrorID enumeration

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

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

ULStreamType enumeration

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

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

ULSyncParms class

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

Properties

The following are properties of ULSyncParms:

Prototype	Description
CheckpointStore As Boolean	If true, adds checkpoints of the database during synchronization to limit database growth during the synchronization process. This is most useful for large downloads with many updates. See "Checkpoint Store synchronization parameter" [*UltraLite Database User's Guide*, page 164].
DownloadOnly As Boolean	Indicates if a synchronization only downloads data. See
NewPassword As String	Change a user password to this new password string on the next synchronization.
Password As String	The password corresponding to a given user name.
PingOnly As Boolean	If true, check the server for liveness, but do not synchronize data.
PublicationMask As Long	Specify the publications to synchronize. The default is to synchronize alll data.
SendColumnNames As Boolean	If SendColumnNames is true, column names are sent to the MobiLink synchronization server. Column names must be sent to the MobiLink synchronization server for automatic script generation.
SendDownloadAck As Boolean	If SendDownloadAck is true, a download acknowledgement is sent during synchronization.

Prototype	Description
Stream As ULStreamType constants	Set the type of stream to use during synchronization.
StreamParms As String	Set extra parameters for the given stream type.
UploadOnly As Boolean	Indicates whether a synchronization only uploads data.
UserName As String	The MobiLink user name for synchronization.
Version As String	The synchronization script version to run.

Examples

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

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

AddAuthenticationParm method

Prototype

AddAuthenticationParm(BSTR parm **)**
Member of **UltraLiteAFLib.ULSyncParms**

Description

Adds a parameter to be passed to the authenticate_parms MobiLink synchronization script.

Parameters

parm The parameter being added.

Returns

No return value.

See also

"Authentication Parameters synchronization parameter" [*UltraLite Database User's Guide*, page 162]

"authenticate_parameters connection event" [*MobiLink Synchronization Reference*, page 98]

ClearAuthenticationParms method

Prototype	**ClearAuthenticationParms( )** Member of **UltraLiteAFLib.ULSyncParms**
Description	Clears all parameters that were to be passed to the authenticate_parms MobiLink synchronization script.
Returns	No return value.
See also	"Authentication Parameters synchronization parameter" [*UltraLite Database User's Guide*, page 162] "authenticate_parameters connection event" [*MobiLink Synchronization Reference*, page 98]

ULSyncResult class

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

Properties

The following are properties of ULSyncResult:

Prototype	Description
AuthStatus As AuthStatusCode (read-only)	Gets the authorization status code for the last synchronization.
IgnoredRows As Boolean (read-only)	Indicates whether rows were ignored during the last synchronization.
StreamErrorCode As ULStream-ErrorCode (read-only)	Gets the error code reported by the synchronization stream.
StreamErrorContext As UL-StreamErrorContext (read-only)	Gets the basic network operation performed.
StreamErrorID As ULStreamErrorID (read-only)	Gets the network layer reporting the error.
StreamErrorSystem As Long (read-only)	Gets the stream error system-specific code.
UploadOK As Boolean (read-only)	Indicates whether data was uploaded successfully in the last synchronization.

ULSyncState enumeration

Constant	Value
ulSyncStateStarting	0
ulSyncStateConnecting	1
ulSyncStateSendingHeader	2
ulSyncStateSendingTable	3
ulSyncStateSendingData	4
ulSyncStateFinishingUpload	5
ulSyncStateReceivingUploadAck	6
ulSyncStateReceivingTable	7
ulSyncStateReceivingData	8
ulSyncStateCommittingDownload	9
ulSyncStateSendingDownloadAck	10
ulSyncStateDisconnecting	11
ulSyncStateDone	12
ulSyncStateError	13
ulSyncStateCancelled	99

ULTable class

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

Before you can work with table data, you must call the Open method. ULTable uses table modes for table operations:

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

Properties

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

Close method

Prototype

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

Description

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

all processing involving the table is complete.

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

Column method

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

Description: Returns the object for the specified column name.

For information about the **ULColumn** object, see "Column" on page 82

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

Returns: Returns a Columns object.

Delete method

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

Description: Deletes the current row from the table.

DeleteAllRows method

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

Description: Deletes all rows in the table.

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

FindBegin method

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

Description: Prepares a table for a find.

FindFirst method

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

Description: 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.

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

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

Note : Requires that FindBegin be called prior to using this method.

Parameters

num_columns An optional parameter referring to the number of columns to be used in the FindFirst. For example, if 2 is passed, the first two columns are used for the FindFirst. If num_columns exceeds the number of columns indexed, all columns are used in FindFirst.

Returns

True if successful.

False if unsuccessful.

FindLast method

Prototype

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

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.

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

☞ For more information, see "Open method" on page 147.

To specify the value to search for, set the column value for each column in the index for which you want to find the value. The cursor is left on the last row found that exactly matches the index value. On failure the cursor position is before the first row (**BOF)**.

> **Note**
> Requires that FindBegin be called prior to using this method.

Parameters

num_columns An optional parameter referring to the number of columns to be used in the FindLast. For example, if 2 is passed, the first two columns are used for the FindLast. If num_columns exceeds the number of columns indexed, all columns are used in FindLast.

Returns

True if successful.

False if unsuccessful.

FindNext method

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

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

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

☞ For more information, see "Open method" on page 147.

The cursor is left on the first row found that exactly matches the index value. On failure, the cursor position is after the last row (**EOF)**.

Note : Must be preceded by FindFirst or FindLast.

Parameters
num_columns An optional parameter referring to the number of columns to be used in the FindNext. For example, if 2 is passed, the first two columns are used for the FindNext. If num_columns exceeds the number of columns indexed, all columns are used in FindNext.

Returns
True if successful.

False if unsuccessful (EOF).

FindPrevious method

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

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

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

☞ For more information, see "Open method" on page 147.

On failure it is positioned before the first row (**BOF)**.

Parameters
num_columns An optional parameter referring to the number of columns to be used in the FindPrevious. For example, if 2 is passed, the first two columns are used for the FindPrevious. If num_columns exceeds the number of columns indexed, all columns are used in FindPrevious.

Returns

True if successful.

False if unsuccessful (BOF).

Insert method

Prototype

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

Description

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

Returns

True if successful.

False if unsuccessful (BOF).

InsertBegin method

Prototype

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

Description

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

Examples

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

```
On Error GoTo InsertError
CustomerTable.InsertBegin
CustomerTable.Column("Fname").StringValue = fname
CustomerTable.Column("Lname").StringValue = lname
CustomerTable.Insert
```

See also

"UpdateBegin method" on page 148

LookupBackward method

Prototype

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

Description

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

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

☞ For more information, see "Open method" on page 147.

To specify the value to search for, set the column value for each column in the index. The cursor is left on the last row that matches or is less than the

index value. On failure (that is, if no row is less than the value being looked for), the cursor position is before the first row (**BOF)**.

Parameters **num_columns** An optional parameter referring to the number of columns.

Returns **True** if successful.

False if unsuccessful.

LookupBegin method

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

Description Prepares a table for a lookup.

LookupForward method

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

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

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

☞ For more information, see "Open method" on page 147.

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure (that is, if no rows are greater than the value being looked for), the cursor position is after the last row (**EOF**).

Parameters **num_columns** An optional parameter referring to the number of columns.

Returns **True** if successful.

False if unsuccessful.

MoveAfterLast method

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

Description Moves to a position after the last row.

Returns **True** if successful.

False if the operation fails.

MoveBeforeFirst method

Prototype	**MoveBeforeFirst( )** As Boolean Member of **UltraLiteAFLib.ULTable**
Description	Moves to a position before the first row.
Returns	**True** if successful. **False** if the operation fails.

MoveFirst method

Prototype	**MoveFirst( )** As Boolean Member of **UltraLiteAFLib.ULTable**
Description	Moves to the first row.
Returns	**True** if successful. **False** if there is no data in the table.

MoveLast method

Prototype	**MoveLast( )** As Boolean Member of **UltraLiteAFLib.ULTable**
Description	Moves to the last row.
Returns	**True** if successful. **False** if there is no data in the table.

MoveNext method

Prototype	**MoveNext( )** As Boolean Member of **UltraLiteAFLib.ULTable**
Description	Moves to the next row.
Returns	**True** if successful. **False** if there is no more data in the table. For example, MoveNext fails if there are no more rows.

MovePrevious method

Prototype	**MovePrevious( )** As Boolean Member of **UltraLiteAFLib.ULTable**
Description	Moves to the previous row.
Returns	**True** if successful.

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

MoveRelative method

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

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

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

Returns
True if successful.

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

Open method

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

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

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

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

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

Truncate method

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

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

☞ For more information, see "StopSynchronizationDelete method" on page 95.

Update method

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

Description
Updates a row in the table with values specified in **ULColumn** methods.

Note : Must be preceded by a call to UpdateBegin.

UpdateBegin method

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

Description
Prepares a table for modifying the contents of the current row.

Example

```
On Error GoTo UpdateError
    CustomerTable.UpdateBegin
    CustomerTable.Column("Fname").StringValue = fname
...
    CustomerTable.Update
```

ULTableSchema class

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

Properties

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

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

GetColumnName method

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

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

Parameters
id The id of the column.

Returns
The name of a column.

GetIndex method

Prototype
GetIndex(*name* As String **)** As ULIndexSchema
Member of **UltraLiteAFLib.ULTableSchema**

Description
Returns the ULIndexSchema object for the specified index.

☞ For information about the ULIndexSchema object, see "ULIndexSchema" on page 109.

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

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

GetIndexName method

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

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

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

Returns Returns the name of the index.

InPublication method

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

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

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

Returns **True** if the table is part of the publication.

False if the table is not part of the publication.

Index

Symbols

?
  using 58

A

AddAuthenticationParm method (ULSyncParms class)
  UltraLite for MobileVB 136
AppendByteChunk method (ULColumn class)
  UltraLite for MobileVB 83
AppendByteChunkParameter method (ULPreparedStatement class)
  UltraLite for MobileVB 111
AppendStringChunk method (ULColumn class)
  UltraLite for MobileVB 84
AppendStringChunkParameter method (ULColumn class)
  UltraLite for MobileVBAPI 112
AppForge Booster
  MobileVB 2, 8
ApplyFile method (ULDatabaseSchema class)
  UltraLite for MobileVB 107
ApplyFileWithParms method (ULDatabaseSchema class)
  UltraLite for MobileVB 107
architecture
  UltraLite for MobileVB 4
AuthStatus property (ULSyncResult class)
  UltraLite for MobileVB 138
AutoCommit mode
  about 69
AutoCommit property (ULConnection class)
  UltraLite for MobileVB 89
AutoIncrement property (ULColumnSchema class)
  UltraLite for MobileVB 88
AutoIncrement property (ULConnectionParms class)
  UltraLite for MobileVB 97

B

BLOB data
  fetching 65
BOF property (ULTable class)
  UltraLite for MobileVB 140
BooleanValue property (ULColumn class)
  UltraLite for MobileVB 82
ByteValue property (ULColumn class)
  UltraLite for MobileVB 82

C

CancelSynchronize method (ULConnection class)
  UltraLite for MobileVB 90
casting
  data types 65
ChangeEncryptionKey method (ULConnection class)
  UltraLite for MobileVB 90
CheckpointStore property (ULSyncParms class)
  UltraLite for MobileVB 135
ClearAuthenticationParms method (ULSyncParms class)
  UltraLite for MobileVB 137
Close method (ULConnection class)
  UltraLite for MobileVB 90
Close method (ULPreparedStatement class)
  UltraLite for MobileVB 112
Close method (ULResultSet class)
  UltraLite for MobileVB 117
Close method (ULTable class)
  UltraLite for MobileVB 140
closing:
  connections 77
  tables 77

Column method (ULTable class)
UltraLite for MobileVB 141
ColumnCount property (ULIndexSchema class)
UltraLite for MobileVB 109
ColumnCount property (ULTableSchema class)
UltraLite for MobileVB 149
columns
accessing schema information 70
Commit method
about 69
Commit method (ULConnection class)
UltraLite for MobileVB 91
commits
about 69
connecting
UltraLite databases 54
connection parameters
databases 54
ContainsTable method (ULPublicationSchema class)
UltraLite for MobileVB 116
conventions
documentation viii
CountUploadRows method (ULConnection class)
UltraLite for MobileVB 91
CreateDatabase method (ULDatabaseManager class)
UltraLite for MobileVB 100
CreateDatabaseWithParms method (ULDatabaseManager class)
UltraLite for MobileVB 102
CustDB sample
UltraLite 51
UltraLite for MobileVB 74

D

data manipulation
about 58, 63
Dynamic SQL 58
Table API 63
Data Manipulation Language
about 58
data types
accessing 64
casting 65
database schema
accessing 70
database state:
maintaining on Palm OS 75
DatabaseID property (ULConnection class)
UltraLite for MobileVB 89
databases
accessing schema information 70
connecting to 54
Palm database 75
DateFormat property (ULDatabaseSchema class)
UltraLite for MobileVB 106
DateOrder property (ULDatabaseSchema class)
UltraLite for MobileVB 106
DatetimeValue property (ULColumn class)
UltraLite for MobileVB 82
DefaultValue property (ULColumnSchema class)
UltraLite for MobileVB 88
Delete method (ULTable class)
UltraLite for MobileVB 141
DeleteAllRows method (ULTable class)
UltraLite for MobileVB 141
deleting rows
about 66
development platforms
supported 2
UltraLite for MobileVB 2
DML operations
about 58
documentation
conventions viii
SQL Anywhere Studio vi
DoubleValue property (ULColumn class)
UltraLite for MobileVB 82
DownloadOnly property (ULSyncParms class)
UltraLite for MobileVB 135
DropDatabase method (ULDatabaseManager class) UltraLite for MobileVB 102
DropDatabaseWithParms method

(ULDatabaseManager class) UltraLite for MobileVB 103

E

EOF property (ULTable class)
UltraLite for MobileVB 140
error handling
about 71
errors
handling 71
ExecuteQuery method (ULPreparedStatement class)
UltraLite for MobileVB 112
ExecuteStatement method (ULPreparedStatement class)
UltraLite for MobileVB 113

F

feedback
documentation xii
providing xii
Find methods
about 66
find mode
about 63
FindBegin method (ULTable class)
UltraLite for MobileVB 141
FindFirst method (ULTable class)
UltraLite for MobileVB 141
FindLast method (ULTable class)
UltraLite for MobileVB 142
FindNext method (ULTable class)
UltraLite for MobileVB 143
FindPrevious method (ULTable class)
UltraLite for MobileVB 143
ForeignKey property (ULIndexSchema class)
UltraLite for MobileVB 109

G

GetByteChunk method (ULColumn class)
UltraLite for MobileVB 84, 117
GetColumnName method (ULIndexSchema class)
UltraLite for MobileVB 110
GetColumnName method (ULTableSchema class)
UltraLite for MobileVB 149
GetDatetime method (ULResultSet class)
UltraLite for MobileVB 121
GetDouble method (ULResultSet class)
UltraLite for MobileVB 121
GetIndex method (ULTableSchema class)
UltraLite for MobileVB 149
GetIndexName method (ULTableSchema class)
UltraLite for MobileVB 150
GetInteger method (ULResultSet class)
UltraLite for MobileVB 121
GetLong method (ULResultSet class)
UltraLite for MobileVB 121
GetNewUUID method (ULConnection class)
UltraLite for MobileVB 91
GetPublicationName method (ULDatabaseSchema class)
UltraLite for MobileVB 108
GetPublicationSchema method (ULDatabaseSchema class)
UltraLite for MobileVB 108
GetReal method (ULResultSet class)
UltraLite for MobileVB 122
GetString method (ULResultSet class)
UltraLite for MobileVB 122
GetStringChunk method (ULColumn class)
UltraLite 118
UltraLite for MobileVB 85
GetTable function (ULConnection class)
UltraLite for MobileVB 92
GetTableName method (ULDatabaseSchema class)
UltraLite for MobileVB 108
GlobalAutoIncrement property (ULColumnSchema class)
UltraLite for MobileVB 88
GlobalAutoIncrementUsage property (ULConnection class)
UltraLite for MobileVB 89
GrantConnectTo method (ULConnection class)
UltraLite for MobileVB 92

I

icons
used in manuals x
ID property (ULColumnSchema class)
UltraLite for MobileVB 88
idnexes
accessing schema information 70
IgnoredRows property (ULSyncResult class)
UltraLite for MobileVB 138
IndexCount property (ULTableSchema class)
UltraLite for MobileVB 149
InPublication method (ULTableSchema class)
UltraLite for MobileVB 150
Insert method (ULTable class)
UltraLite for MobileVB 144
insert mode
about 63
InsertBegin method (ULTable class)
UltraLite for MobileVB 144
inserting rows
about 66
IntegerValue property (ULColumn class)
UltraLite for MobileVB 82
internals
data manipulation 58, 63
IsColumnDescending method (ULIndexSchema class)
UltraLite for MobileVB 110
IsNull method (ULResultSet class)
UltraLite for MobileVB 121
IsNull property (ULColumn class)
UltraLite for MobileVB 82
IsOpen property (ULTable class)
UltraLite for MobileVB 140

L

LastDownloadTime method (ULConnection class)
UltraLite for MobileVB 92
LastIdentity property (ULConnection class)
UltraLite for MobileVB 89
LongValue property (ULColumn class)
UltraLite for MobileVB 82
Lookup methods
about 66
lookup mode
about 63
LookupBackward method (ULTable class)
UltraLite for MobileVB 144
LookupBegin method (ULTable class)
UltraLite for MobileVB 145
LookupForward method (ULTable class)
UltraLite for MobileVB 145

M

Mask property (ULPublicationSchema class)
UltraLite for MobileVB 116
Mask property (ULResultSet class)
UltraLite for MobileVB 117
Mask property (ULResultSetSchema class)
UltraLite for MobileVB 123
Microsoft Visual Basic
supported versions 2
MobileVB
AppForge Booster 2, 8
Development platforms 2
supported versions 2
modes
about 63
MoveAfterLast method (ULResultSet class)
UltraLite for MobileVB 119
MoveAfterLast method (ULTable class)
UltraLite for MobileVB 145
MoveBeforeFirst method (ULResultSet class)
UltraLite for MobileVB 119
MoveBeforeFirst method (ULTable class)
UltraLite for MobileVB 146
MoveFirst method
introduction 59, 64
MoveFirst method (ULResultSet class)
UltraLite for MobileVB 119
MoveFirst method (ULTable class)
UltraLite for MobileVB 146
MoveLast method (ULResultSet class)

UltraLite for MobileVB 120
MoveLast method (ULTable class)
UltraLite for MobileVB 146
MoveNext method
introduction 59, 64
MoveNext method (ULResultSet class)
UltraLite for MobileVB 120
MoveNext method (ULTable class)
UltraLite for MobileVB 146
MovePrevious method (ULResultSet class)
UltraLite for MobileVB 120
MovePrevious method (ULTable class)
UltraLite for MobileVB 146
MoveRelative method (ULResultSet class)
UltraLite for MobileVB 120
MoveRelative method (ULTable class)
UltraLite for MobileVB 147

N

Name property (ULColumnSchema class)
UltraLite for MobileVB 88
Name property (ULIndexSchema class)
UltraLite for MobileVB 109
Name property (ULPublicationSchema class)
UltraLite for MobileVB 116
Name property (ULResultSet class)
UltraLite for MobileVB 117
Name property (ULResultSetSchema class)
UltraLite for MobileVB 123
Name property (ULTableSchema class)
UltraLite for MobileVB 149
NearestCentury property (ULDatabaseSchema class)
UltraLite for MobileVB 106
NeverSynchronized property (ULTableSchema class)
UltraLite for MobileVB 149
NewPassword property (ULSyncParms class)
UltraLite for MobileVB 135
newsgroups
technical support xii
Nullable property (ULColumnSchema class)
UltraLite for MobileVB 88

O

OnReceive event (ULConnection class)
UltraLite for MobileVB 92
OnSend event(ULConnection class)
UltraLite for MobileVB 93
OnStateChange event(ULConnection class)
UltraLite for MobileVB 93
OnTableChange event (ULConnection class)
UltraLite for MobileVB 94
Open method
ULTable object 59, 64
Open method (ULTable class)
UltraLite for MobileVB 147
OpenByIndex method
ULTable object 59, 64
OpenConnection method (ULDatabaseManager class)
UltraLite for MobileVB 103
OpenConnectionWithparms method (ULDatabaseManager class)
UltraLite for MobileVB 104
OpenParms property (ULConnection class)
UltraLite for MobileVB 89
OptimalIndex property (ULColumnSchema class)
UltraLite for MobileVB 88

P

Palm Computing Platform
supported versions 2
Palm computing platform
described 75
Palm databases
PDB 75
Palm OS
example 77
unsupported versions 2
Password property (ULSyncParms class)
UltraLite for MobileVB 135
persistent name

example 77
persistent name:
maintaining 75
using 75
PingOnly property (ULSyncParms class)
UltraLite for MobileVB 135
platforms
supported 2
Precision property (ULColumnSchema class)
UltraLite for MobileVB 88
Precision property (ULDatabaseSchema class)
UltraLite for MobileVB 106
prepared statements
about 58
PrepareStatement method (ULConnection class)
UltraLite for MobileVB 94
PrimaryKey property (ULIndexSchema class)
UltraLite for MobileVB 109
PrimaryKey property (ULTableSchema class)
UltraLite for MobileVB 149
projects
creating UltraLite for MobileVB projects 9, 25, 41
PublicationCount property (ULDatabaseSchema class)
UltraLite for MobileVB 106
PublicationMask property (ULSyncParms class)
UltraLite for MobileVB 135
publications
accessing schema information 70

R

RealValue property (ULColumn class)
UltraLite for MobileVB 82
ReferencedIndexName property (ULIndexSchema class)
UltraLite for MobileVB 109
ReferencedTableName property (ULIndexSchema class)
UltraLite for MobileVB 109
ResetLastDownloadTime method (ULConnection class)
UltraLite for MobileVB 94
RevokeConnectFrom method (ULConnection class)
UltraLite for MobileVB 95
Rollback method
about 69
Rollback method (ULConnection class)
UltraLite for MobileVB 95
rollbacks
about 69
RowCount property (ULTable class)
UltraLite for MobileVB 140
rows
accessing current row 64

S

samples
UltraLite 51
UltraLite for MobileVB 74
Scale property (ULColumnSchema class)
UltraLite for MobileVB 88
schema
accessing 70
Schema property (ULColumn class)
UltraLite for MobileVB 82
Schema property (ULConnection class)
UltraLite for MobileVB 89
Schema property (ULTable class)
UltraLite for MobileVB 140
scrolling
through rows 64
searching
rows 66
SELECT
about 59
SendColumnNames property (ULSyncParms class)
UltraLite for MobileVB 135
SendDownloadAck property (ULSyncParms class)
UltraLite for MobileVB 135
SetBooleanParameter method (ULPreparedStatement class)
UltraLite for MobileVB 113
SetByteChunk method (ULColumn class)

UltraLite for MobileVB 86
SetByteChunkParameter method (ULPreparedStatement class)
UltraLite for MobileVB 113
SetByteParameter method (ULPreparedStatement class)
UltraLite for MobileVB 113
SetDatetimeParameter method (ULPreparedStatement class)
UltraLite for MobileVB 114
SetDoubleParameter method (ULPreparedStatement class)
UltraLite for MobileVB 114
SetIntegerParameter method (ULPreparedStatement class)
UltraLite for MobileVB 114
SetLongParameter method (ULPreparedStatement class)
UltraLite for MobileVB 115
SetNull method (ULColumn class)
UltraLite for MobileVB 87
SetNullParameter method (ULPreparedStatement class)
UltraLite for MobileVB 115
SetRealParameter method (ULPreparedStatement class)
UltraLite for MobileVB 115
SetStringParameter method (ULPreparedStatement class)
UltraLite for MobileVB 115
SetToDefault method (ULColumn class)
UltraLite for MobileVB 87
Signature property (ULDatabaseSchema class)
UltraLite for MobileVB 106
Size property (ULColumnSchema class)
UltraLite for MobileVB 88
SQL Anywhere Studio
documentation vi
SQLType property (ULColumnSchema class)
UltraLite for MobileVB 88
SQL Anywhere Studio
additional features 2
StartSynchronizationDelete method (ULConnection class)
UltraLite for MobileVB 95
StopSynchronizationDelete method (ULConnection class)
UltraLite for MobileVB 95
Stream property (ULSyncParms class)
UltraLite for MobileVB 135
StreamErrorContext property (ULSyncResult class)
UltraLite for MobileVB 138
StreamErrorID property (ULSyncResult class)
UltraLite for MobileVB 138
StreamErrorSystem property (ULSyncResult class)
UltraLite for MobileVB 138
StreamParms property (ULSyncParms class)
UltraLite for MobileVB 135
StringToUUID method (ULConnection class)
UltraLite for MobileVB 95
StringValue method
introduction 64
StringValue property (ULColumn class)
UltraLite for MobileVB 82
support
newsgroups xii
supported platforms 2
synchronization
adding the synchronization template 72
monitoring status 72
UltraLite for MobileVB 72
writing code 72
Synchronize method (ULConnection class)
UltraLite for MobileVB 96
system requirements
UltraLite for MobileVB 8

T

TableCount property (ULDatabaseSchema class)
UltraLite for MobileVB 106
tables
accessing schema information 70
target platforms
supported 2
UltraLite for MobileVB 2

technical support
newsgroups xii
TimeFormat property (ULDatabaseSchema class)
UltraLite for MobileVB 106
transaction processing
about 69
transactions
about 69
Truncate method (ULTable class)
UltraLite for MobileVB 147
tutorial for CE
UltraLite Component Suite 39
tutorials
UltraLite for MobileVB (Palm OS) 7
UltraLite for MobileVB (Windows CE) 23

U

ULAuthStatusCode constants
about 81
UltraLite for MobileVB 81
ULColumn class
about 82
properties 82
UltraLite for MobileVB 82
ULColumn object
introduction 64
ULColumnSchema class
about 88
properties 88
UltraLite for MobileVB 88
ULColumnSchema object
introduction 70
ULConnection class
about 89
properties 89
UltraLite for MobileVB 89
ULConnection object
introduction 54
ULConnectionParms class
about 97
properties 97
UltraLite for MobileVB 97
ULDatabaseManager class
about 100
properties 100
UltraLite for MobileVB 100
ULDatabaseManager object
introduction 54
ULDatabaseSchema class
about 106
properties 106
UltraLite for MobileVB 106
ULDatabaseSchema object
introduction 70
ULIndexSchema class
about 109
properties 109
UltraLite for MobileVB 109
ULIndexSchema object
introduction 70
ULPreparedStatement
about 58
ULPreparedStatement class
about 111
properties 111
UltraLite for MobileVB 111
ULPublicationSchema class
about 116
properties 116
UltraLite for MobileVB 116
ULPublicationSchema object
introduction 70
ULResultSet class
about 117
properties 117
UltraLite for MobileVB 117
ULResultSetSchema class
about 123
properties 123
UltraLite for MobileVB 123
ULSQLCode constants
about 124
UltraLite for MobileVB 124
ULSQLType constants
about 128
UltraLite for MobileVB 128
ULStreamErrorCode constants
about 129
UltraLite for MobileVB 129
ULStreamErrorCode property (ULSyncResult class)
UltraLite for MobileVB 138

ULStreamErrorContext constants
about 132
UltraLite for MobileVB 132
ULStreamErrorID constants
about 133
UltraLite for MobileVB 133
ULStreamType
about 134
UltraLite for MobileVB 134
ULSyncParms class
about 135
properties 135
UltraLite for MobileVB 135
ULSyncResult class
about 138
properties 138
UltraLite for MobileVB 138
ULSyncState enum
about 139
UltraLite for MobileVB 139
ULTable class
about 140
properties 140
UltraLite for MobileVB 140
ULTable object
introduction 59, 64
ULTableSchema class
about 149
properties 149
UltraLite for MobileVB 149
ULTableSchema object
introduction 70
UltraLite
about 1
UltraLite for MobileVB
architecture 4
ULColumnSchema class 88
ULConnection class 89
ULConnectionParms class 97
ULDatabaseManager class 100
ULDatabaseSchema class 106
ULIndexSchema class 109
ULPreparedStatement class 111
ULPublicationSchema class 116
ULResultSet class 117
ULResultSetSchema class 123
ULSQLCode constants 124
ULSQLType constants 128
ULStreamErrorCode constants 129
ULStreamErrorContext constants 132
ULStreamErrorID constants 133
ULStreamType 134
ULSyncParms class 135
ULSyncResult class 138
ULSyncState enum 139
ULTable class 140
ULTableSchema class 149
UltraLite for MobileVB API
ULAuthStatusCode constant 81
ULColumn class 82
ULConnection class 89
UltraLite for MobileVB projects
creating 9, 25, 41
UniqueIndex property (ULIndexSchema class)
UltraLite for MobileVB 109
UniqueKey property (ULIndexSchema class)
UltraLite for MobileVB 109
Update method (ULTable class)
UltraLite for MobileVB 148
update mode
about 63
UpdateBegin method (ULTable class)
UltraLite for MobileVB 148
updating rows
about 66
UploadOK property (ULSyncResult class)
UltraLite for MobileVB 138
UploadOnly property (ULSyncParms class)
UltraLite for MobileVB 135
UserName property (ULSyncParms class)
UltraLite for MobileVB 135
UUIDs
getting as string 91
StringToUUID method 95
UUIDToString method 96
UUIDToString method (ULConnection class)
UltraLite for MobileVB 96
UUIDValue property (ULColumn class)

UltraLite for MobileVB 82

V

values
accessing 64
Version property (ULDatabaseManager class)
UltraLite for MobileVB 100
Version property (ULSyncParms class)
UltraLite for MobileVB 135
Visual Basic
supported versions 2

W

Windows CE
supported versions 2