UltraLite™ for MobileVB User's Guide

Last modified: October 2002
Part Number: 36292-01-0802-01

Last modified October 2002. Part number 36292-01-0802-01.

Contents

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.

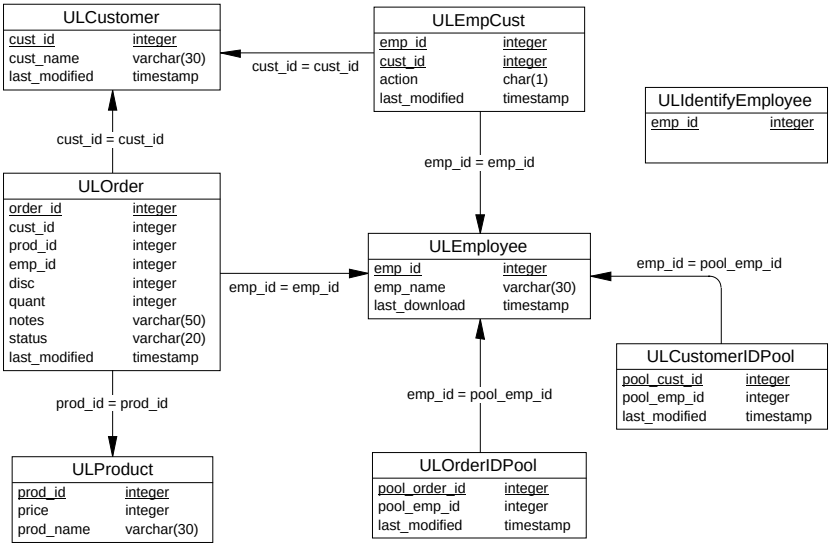
The UltraLite sample database

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

The UltraLite sample database schema is held in a file named *custdb.xml*, and is located in the *Samples\UltraLiteForMobileVB\CustDB* subdirectory of your SQL Anywhere directory. A complete application built on this database is also supplied as *Samples\UltraLiteFor MobileVB\CustDB\custdb.vbg*.

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 a newsgroup. The newsgroup can be found on the *forums.sybase.com* news server as news://forums.sybase.com/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 to UltraLite for MobileVB

About this chapter

This chapter introduces you to UltraLite for MobileVB features, platforms, architecture, and functionality. It assumes that you are familiar with the UltraLite Component Suite, as described in "Introduction to the UltraLite Component Suite" on page 1 of the book *UltraLite Foundations*.

Contents

UltraLite for MobileVB features

UltraLite for MobileVB is a member of the UltraLite Component Suite. It provides the following benefits for developers targeting small devices:

- a robust relational database store
- synchronization
- application development using the AppForge MobileVB development tools
- deployment on the Palm OS and Windows CE platforms.

For more information on the features and benefits of the UltraLite Component Suite, see "Introduction to the UltraLite Component Suite" on page 2 of the book *UltraLite Foundations*.

System requirements and supported platforms

Platform support for UltraLite is of the following kinds:

- **Target platforms** The **target platform** is the device and operating system on which you deploy your finished UltraLite application.
- **Development platforms** For each target platform, you develop your applications using a particular development tool and operating system. The tool and operating system comprise the **development platform**.

> **AppForge Booster**
> 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.

Development platforms

To develop applications using UltraLite for MobileVB, you require

- Microsoft Windows NT/2000/XP
- Microsoft Visual Basic 6
- AppForge 2.11 or AppForge MobileVB Version 3.0.

> **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\ulmvb8.dll* under your SQL Anywhere directory to the *\Program Files\AppForge* directory on your device.

Target platforms

UltraLite for MobileVB supports the following target platforms:

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

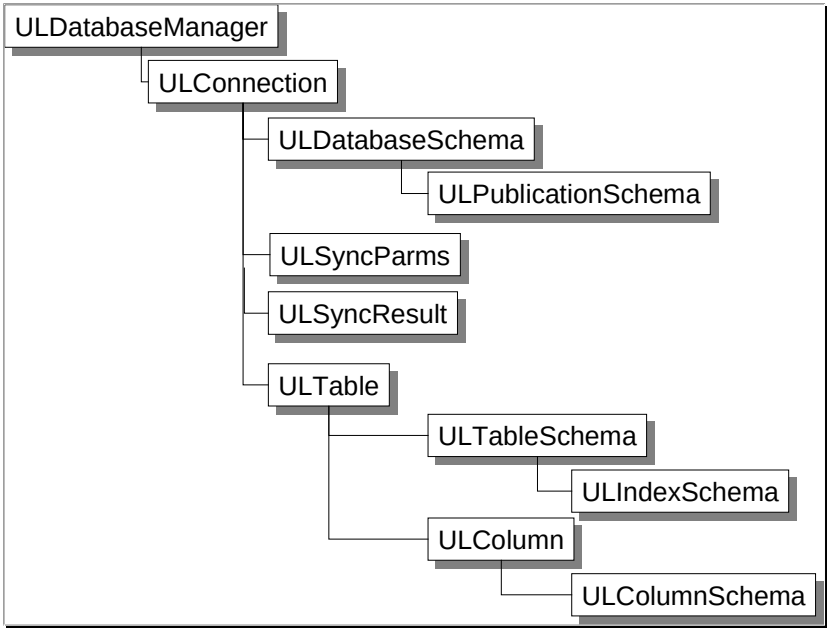
SQL Anywhere Studio

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 who wish to model an UltraLite database after an Adaptive Server Anywhere database, can use the *ulinit* command-line tool to generate an UltraLite schema file from 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 ingot that exposes a set of objects for data manipulation using the 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 on the ULDatabaseManager class, see "ULDatabaseManager class" on page 74.

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

  For more information on ULConnection, see "ULConnection class" on page 67.

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

For more information on the ULTable, ULColumn, and ULIndexSchema objects, see "ULTable class" on page 97, "ULColumn" on page 61, and "ULIndexSchema" on page 80.

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

  For more information on synchronization with MobiLink, see the MobiLink Synchronization Users Guide in the SQL Anywhere Studio.

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

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 50 minutes.

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.

To complete the synchronization section of the tutorial requires that:

- you can create an ODBC data source
- you can use command line options and parameters
- you have knowledge of both the Sybase Central Adaptive Server Anywhere plug-in and the MobiLink synchronization plug-in

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 database schema

A **schema** is a database definition without the data. You create an UltraLite schema file as a necessary first step to making an UltraLite database.

For more information on UltraLite schemas, see "UltraLite for MobileVB architecture" on page 5.

When creating UltraLite schemas for a Palm device, the following information is necessary:

- A way to identify the database so an application can connect to it. This is done with the Palm creator ID.
- A way to identify the schema on the development machine so it can be copied to the device.
- A way to identify the schema on the device.

Create your schema file using the UltraLite Schema Painter

To complete this tutorial you need a directory to hold the files you create. This directory is assumed to be *C:\tutorial\mvb*. If you create your tutorial directory elsewhere, supply the path to your location instead of *c:\tutorial\mvb* throughout.

❖ **To create the schema file using the UltraLite Schema Painter:**

1. Start the UltraLite Schema Painter:

   Click Start➤Programs➤Sybase SQL Anywhere 8➤UltraLite ➤UltraLite Schema Painter.

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

   - Open the Tools folder and double-click Create UltraLite schema.
   - In the file dialog box, type **c:\tutorial\mvb\tutCustomer.usm** or Browse to the folder and enter **tutCustomer**.
   - Click OK to create the schema.

3. Create a table called *customer*.

   - Expand the *tutCustomer* item in the left pane of the UltraLite Schema Painter and select the *Tables* folder.
   - Open the Tables folder and double-click Add Table. The New Table dialog appears.

- Enter the name *customer.*
- In the New Table dialog, add columns with the following properties.

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

- Set *Id* as the primary key: Click Primary Key and add *Id* to the index, marking it as ascending.
- Check your work and click OK to complete the table definition and dismiss the New Table dialog.

4 Click File ➤Save to save the *tutcustomer.usm* file.

5 Export a Palm schema file.

- Right click on the database icon and select Export Schema for Palm from the popup menu.
- Enter a Palm Creator ID of **Syb3**.

> **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.

- Leave the filename at its default setting to save the PDB file in your tutorial directory. Click OK.
- Exit the UltraLite Schema Painter

You have now defined the schema of your UltraLite database. Although this database contains only a single table, you can use many tables in UltraLite databases.

Lesson 2: Create a project architecture

The tutorial assumes the folder *c:\tutorial\mvb*, the same one holding your schema file, is where you will store your application files.

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 for MobileVB reference:**

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
   - Check the box beside the item iAnywhere Solutions, UltraLite for MobileVB 8.0.
     - If the item does not appear, click Browse and go to the *win32* subdirectory of your SQL Anywhere installation: *UltraLite\UltraLiteforMobileVB\win32*
     - Select the file *ulmvb8.dll* and click Open.
   - Click OK
5. Give the Project a name.
   - Click Project➤MobileVBProject1 Properties
   - Under Project Name, type **UltraLiteTutorial** typed all as 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\MainForm.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 3: Create a form interface

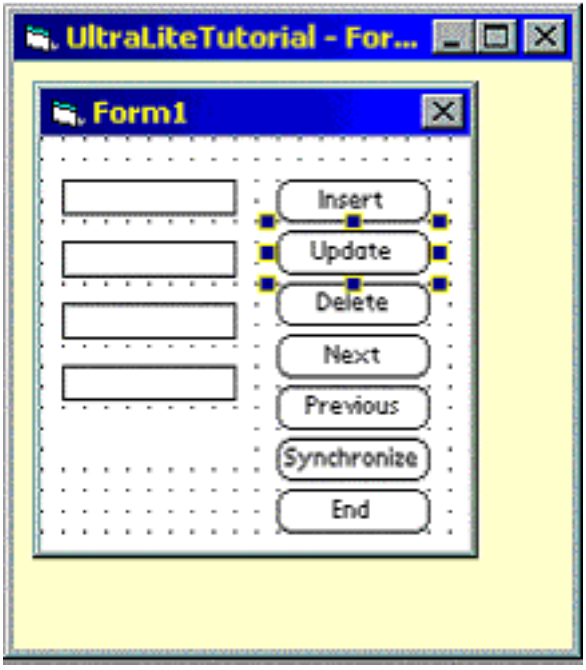
You are now ready to design your application form. The project should have a single form named Form1 displayed.

❖ **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 4: 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 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 UltraLite objects you will need in this sample:

```
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. This is the only object that can be instantiated.

2 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()
    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\mvb\tutCustomer.udb"
    schema_parms = open_parms & ";" & _
       "PALM_SCHEMA=tutCustomer;" & _
       "SCHEMA_FILE=c:\tutorial\mvb\tutCustomer.usm"

    On Error Resume Next

    Set Connection = _
        DatabaseMgr.OpenConnection(open_parms)
    If Err.Number = ULSQLCode.ulSQLE_NOERROR Then
        MsgBox "Connected to an existing database"
    ElseIf Err.Number = _
       ULSQLCode.ulSQLE_DATABASE_NOT_FOUND Then
        Err.Clear
        Set Connection = _
        DatabaseMgr.CreateDatabase(schema_parms)
        If Err.Number = ULSQLCode.ulSQLE_NOERROR _
        Then
            MsgBox "Connected to a new database"
        Else
            MsgBox Err.Description
        End If
    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.

3 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
```

4 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.

For more information, see "ULConnection class" on page 67.

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
        txtPhone.Text = _
        CustomerTable.Column("Phone").StringValue
    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:

```
dbmlsrv8 -c "dsn=ASA 8.0 Sample" -v+ -zu+ -za
```

The ASA 8.0 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 8 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 8.0 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 5: 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

Samples

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

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

CHAPTER 3

Tutorial: An UltraLite for MobileVB Application for PocketPC

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

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 your CE device that synchronizes with a larger database running on your desktop machine.

Timing

The tutorial takes about 50 minutes.

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.

To complete the synchronization section of the tutorial requires that:

- you can create an ODBC data source
- you can use command line options and parameters
- you have knowledge of both the Sybase Central Adaptive Server Anywhere plug-in and the MobiLink synchronization plug-in

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 database schema

A **schema** is a database definition without the data. You create an UltraLite schema file as a necessary first step to making an UltraLite database.

For more information on UltraLite schemas, see "UltraLite for MobileVB architecture" on page 5.

When creating UltraLite schemas, the following information is necessary:

- A way to identify the schema on the development machine so it can be copied to the device.
- A way to identify the schema on the device.

Create your schema file using the UltraLite Schema Painter

To complete this tutorial you need a directory to hold the files you create. This directory is assumed to be *C:\tutorial\mvb*. If you create your tutorial directory elsewhere, supply the path to your location instead of *c:\tutorial\mvb* throughout.

❖ **To create the schema file using the UltraLite Schema Painter:**

1. Start the UltraLite Schema Painter:

   Click Start➤Programs➤Sybase SQL Anywhere 8➤UltraLite ➤UltraLite Schema Painter.

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

   - Open the Tools folder and double-click Create UltraLite schema.
   - In the file dialog box, type **c:\tutorial\mvb\tutCustomer.usm** or Browse to the folder and enter **tutCustomer**.
   - Click OK to create the schema.

3. Create a table called *customer*.

   - Expand the *tutCustomer* item in the left pane of the UltraLite Schema Painter and select the *Tables* folder.
   - Open the Tables folder and double-click Add Table. The New Table dialog appears.
   - Enter the name *customer*.
   - In the New Table dialog, add columns with the following properties.

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

- Set *Id* as the primary key: Click Primary Key and add *Id* to the index, marking it as ascending.
- Check your work and click OK to complete the table definition and dismiss the New Table dialog.

4 Click File ➤Save to save the *tutcustomer.usm* file.

- Exit the UltraLite Schema Painter

You have now defined the schema of your UltraLite database. Although this database contains only a single table, you can use many tables in UltraLite databases.

Lesson 2: Create a project architecture

The tutorial assumes the folder *c:\tutorial\mvb*, the same one holding your schema file, is where you will store your application files.

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 for MobileVB reference:**

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 PocketPC.
4. Create a Visual Basic reference to the UltraLite for MobileVB component:
   - Click Project ➤ References
   - Check the box beside the item iAnywhere Solutions, UltraLite for MobileVB 8.0.
     - If the item does not appear, click Browse and go to the *win32* subdirectory of your SQL Anywhere installation: *UltraLite\UltraLiteforMobileVB\win32*
     - Select the file *ulmvb8.dll* and click Open.
   - Click OK
5. Give the Project a name.
   - Click Project➤MobileVBProject1 Properties
   - Under Project Name, type **UltraLiteTutorial** typed all as 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\MainForm.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 3: 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 a 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

2 Compile and validate the application.

- Choose MobileVB ➤Compile and Validate.

Lesson 4: 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. 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 UltraLite objects you will need in this sample:

     ```
     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. This is the only object that can be instantiated.
2. 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()
    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
       "FILE_NAME=c:\tutorial\mvb\tutCustomer.udb"
    schema_parms = open_parms & ";" & _
       "SCHEMA_FILE=c:\tutorial\mvb\tutCustomer.usm"

    On Error Resume Next

    Set Connection = _
        DatabaseMgr.OpenConnection(open_parms)
    If Err.Number = ULSQLCode.ulSQLE_NOERROR Then
        MsgBox "Connected to an existing database"
    ElseIf Err.Number = _
       ULSQLCode.ulSQLE_DATABASE_NOT_FOUND Then
        Err.Clear
        Set Connection = _
        DatabaseMgr.CreateDatabase(schema_parms)
        If Err.Number = ULSQLCode.ulSQLE_NOERROR _
        Then
            MsgBox "Connected to a new database"
        Else
            MsgBox Err.Description
        End If
    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.

3 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
```

4 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.

For more information, see "ULConnection class" on page 67.

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
        txtPhone.Text = _
        CustomerTable.Column("Phone").StringValue
    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.

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

```
dbmlsrv8 -c "dsn=ASA 8.0 Sample" -v+ -zu+ -za
```

The ASA 8.0 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 8 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 8.0 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 5: Deploying 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 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 on 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 from your desktop, the file *SQL Anywhere\Ultralite\UltraLite\UltraLiteForMobileVB\ce\arm\ulmvb8.dll* to your device *\Program Files\AppForge*. This step only needs to be performed once per device.
4. On your device, go to your Programs.
5. Choose *UltraLiteTutorialCE*. 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

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 4

Understanding UltraLite for MobileVB Development

About this chapter

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

Contents

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.

❖ **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. 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*.

For more information on connection parameters, see "Connection Parameters" on page 25 of the book *UltraLite Foundations*.

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 is committed to the database immediately. You can also set ULConnection.AutoCommit to False to build transactions into your application.

For more information, see "Transaction processing in UltraLite" on page 54.

- **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.

Accessing and manipulating data

UltraLite applications 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 makes a copy of the row in a buffer. 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.
- **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
TCustomer.MoveBeforeFirst
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.

```
' 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 "ULColumn" on page 61.

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 sort order 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 sort order 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. Setting these values affects the buffer holding the current row only, not the database. For example:

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

Only values in the columns of the index are relevant to the search.

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 "ULTable class" on page 97.

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 105

By default, UltraLite operates in AutoCommit mode, so that the Update is immediately applied to the row in permanent storage. If you have disabled AutoCommit mode, the Update is not applied until you execute a Commit operation. For more information, see "Transaction processing in UltraLite" on page 54.

> ***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" ).SringValue = "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, 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.

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

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 "ULSQLCode enum" on page 83.

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
```

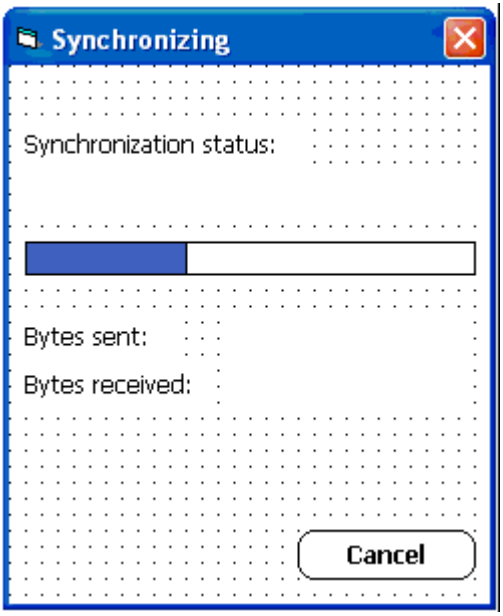
You can then access UltraLite errors using the GetError function.

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 (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 sample.

CHAPTER 5

API Reference

About this chapter

This chapter describes the UltraLite for MobileVB API.

Contents

ULAuthStatusCode constants

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 ULColumn object represents a particular value in a table; the row is determined by the ULTable object.

For information about the ULTable object, see "ULTable class" on page 97.

Properties

Prototype	Description
BooleanValue as Boolean	Returns the current value.
ByteValue as Byte	Returns the current value
DatetimeValue as Date	Returns the current value
DoubleValue as Double	Returns the current value
IntegerValue as Integer	Returns the current value
IsNull as Boolean (read only)	Indicates whether the column value is NULL
LongValue as Long	Returns the current value
RealValue as Single	Returns the current value
Schema as ULColumnSchema (read only)	Returns the object representing the schema of the column
StringValue as String	Returns the current value
UUIDValue As String	The column value as a UUID. For reading, gets the column value as a UUID. If the value is not a valid UUID, a SQLE_CONVERSION_ERROR is raised. For writing, stores the value as a UUID in the database. UUID is a BINARY16 data type.

AppendByteChunk method

Prototype **AppendByteChunk(** *data* As long, *data_len* As long **)**
Member of **UltraLiteAFLib.ULColumn**

Description Appends the buffer of bytes to the column if the type is `ulTypeLongBinary`.

Parameters

data An array of bytes.

data_len The number of bytes from the array to append.

Errors set

Error	Description
ulSQLE_INVALID_PARAMETER	If data length is less than 0
ulSQLE_CONVERSION_ERROR	If the column data type is not LONG BINARY.

Example

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

In the example code, *edata* is a column name and 512 bytes of data are appended to the column.

AppendStringChunk method

Prototype **AppendStringChunk(** *data* As String **)**
Member of **UltraLiteAFLib.ULColumn**

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

Parameters

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

Errors set

Error	Description
ulSQLE_CONVERSION_ERROR	If the column data type is not LONG VARCHAR.

GetByteChunk method

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

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

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 length of the buffer, or array.

offset The offset into the underlying array of bytes.

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	If the column data type isn't BINARY or LONG BINARY
ulSQLE_INVALID_PARAMETER	If the column data type is BINARY and any of the following is true: ♦ offset is not 0 or 1 ♦ data length is greater than 64K ♦ data length is less than 0
ulSQLE_INVALID_PARAMETER	If the column data type is LONG BINARY and any of the following is true: ♦ offset is less than 1 ♦ data length is less than 0

Example

```
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)
```

In the example code, *edata* is a column name. Note, if the *data_len* parameter passed in is not long enough, the entire application will terminate.

GetStringChunk method

Prototype

GetStringChunk (*src_offset* As Long,
data as string, *string_len* As Long,
filled_len as long **)** As Boolean
Member of **UltraLiteAFLib.ULColumn**

Description

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

Parameters

string_length The length of the string you want returned.

src_offset This is the character offset into the underlying data from which we start getting the string.

Returns

True if there is more data for this value.

False if there is no more data for this value.

Errors set

Error	Description
ulSQLE_CONVERSION_ERROR	If the column data type isn't CHAR or LONG VARCHAR
ulSQLE_INVALID_PARAMETER	If the column data type is CHAR and the src_offset is greater than 64K
ulSQLE_INVALID_PARAMETER	If src_offset is less than 0 or string length is less than 0.

SetByteChunk method

Prototype

SetByteChunk(*data* As long, *data_len* as long **)**
Member of **UltraLiteAFLib.ULColumn**

Description

Sets the value of the column in the database to the array of bytes in the data field. Suitable for binary or long binary columns.

Parameters

data An array of bytes.

data_len The length of the array.

Errors set

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

Example

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

In the example code, *edata* is a column name and 232 bytes of data in the array contain values to be set in the database.

SetNull method

Prototype **SetNull()**
Member of **UltraLiteAFLib.ULColumn**

Description Sets this column's 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.

ULColumnSchema class

The ULColumnSchema object allows you to obtain 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)	Determines whether this column defaults to an autoincrement value
DefaultValue as String (read only)	Indicates the value that is used if one was not provided when a row was inserted.
GlobalAutoIncrement as Boolean (read only)	Determines whether this column defaults to a global autoincrement value
ID as integer (read only)	The ID of the column
Name as String (read only)	The column name
Nullable as Boolean (read-only)	True if the column allows NULLs
OptimalIndex as ULIndexSchema (read only)	The index with this column as its first column.
Precision as Integer (read only)	The precision value for the column if it is of type ulTypeNumeric
Scale as Integer (read only)	The scale value for the column if it is of type ulTypeNumeric
Size as Long (read only)	The size the column was created with if its type takes a size
SQLType as ULSQLType (read only)	The SQL type assigned to the column when it was created

ULConnection class

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

Properties

The following are properties of ULConnection:

Prototype	Description
AutoCommit as Boolean	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	Sets the database ID value to be used for global autoincrement columns
GlobalAutoIncrementUsage as Integer (read only)	Returns the percentage of available global autoincrement values that have been used
LastIdentity as Long (read only)	Returns the most recent value inserted into a column with a default of autoincrement or global autoincrement.
OpenParms as String (read only)	The string used to open the connection to the database.
Schema as ULDatabaseSchema (read only)	Returns the ULDatabaseSchema object

CancelSynchronize method

Prototype

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

Description

When called during synchronization, the method cancels the synchronization. To allow this the ULConnection object must be declared **WithEvents**. The user can only call this method during one of the synchronization events.

Close method

Prototype

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

Description

Closes the connection to the database. No methods on the ULConnection object 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.

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

threshold The maximum number of rows to count. Use -1 to indicate no maximum. If not specified, this value is –1.

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`.

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.

Examples In this example, the Customer table is accessed.

```
Set CustomerTable = Connection.GetTable("Customer")
```

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 for the current user.

password The password for this user ID.

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

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.

Parameters

new_state The state that the synchronization process is about to enter.

old_state 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 DatabaseSchema.GetTableName method.

numTables The number of tables eligible to be synchronized.

Example See the custdb application for an example of this method.

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 this function is called, all subsequent delete operations are uploaded at the next synchronization.

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 a string in the form `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx` to an array of 16 bytes. The pointer to the buffer must be declared as 16 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.

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

Expects a VarPtr to a buffer of 16 bytes. Converts this buffer to a string in the form xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx. The buffer must be declared (1 to 16) As Byte (that is, an array of 16 bytes). Visual Basic is unable to check the bounds for this buffer so if it is not big enough, the application could overwrite memory.

ULDatabaseManager class

The ULDatabaseManager class is used to manage connections and databases. Your application should only have one instance of this object.

Properties

The following is a property of DatabaseManager:

Prototype	Description
Version as String	Identifies the version of the UltraLite for MobileVB 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. To alter the schema of an existing database, use the ULDatabaseSchema ApplyFile method.

For more information on ApplyFile, see "ULDatabaseSchema class" on page 77 and "ApplyFile method" on page 78.

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 on connection parameters, see "Connection Parameters" on page 25 of the book *UltraLite Foundations.*

For more information on the Palm_fs parameter, see "palm_fs parameter " on page 31 of the book *UltraLite Foundations*.

Examples

The following code creates a DatabaseManager 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
```

DropDatabase method

Deletes a database file.

Prototype

DropDatabase(*parms* As String)
Member of **UltraLiteAFLib.ULDatabaseManager**

Description

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

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)
```

OpenConnection method

Prototype

OpenConnection(*connparms* As string **)** 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* string. It should contain a value of the form

```
file_name=UDBFILE

DBF=UDBFILE

palm_db=CreatorID.
```

Parameters

connparms The parameters that determine the target database. Parameters are specified using a sequence of `"name=value"` pairs. If no user ID or password is given, the default is used.

> **Note for VFS card for Palm users**
> The Palm_fs=vfs parameter needs to be specified both for CreateDatabase and OpenConnection methods.

For more information on the Palm_fs parameter, see "palm_fs parameter " on page 31 of the book *UltraLite Foundations*.

Returns

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")
```

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)	Controls the interpretation of date formats; valid values are 'MDY', 'YMD', or 'DMY'.
NearestCentury as String (read only)	Controls 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)	Specifies the maximum number of digits in the result of any decimal arithmetic.
PublicationCount as Integer (read only)	The number of publications in the connected database.
Signature as String (read only)	An internal identifier representing the database schema.
TableCount as Integer (read only)	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)	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.

In most circumstances there is no data loss, but data loss can occur if columns are deleted, for example, or if the data type for a column is changed to an incompatible type.

Parameters

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

Example

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

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 82.

For more information, see ULDatabaseSchema "Properties" on page 77

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)	Returns the number of columns in the index
ForeignKey as Boolean (read only)	Returns whether this is a foreign key.
Name as String (read only)	Returns the name of the index
PrimaryKey as Boolean (read only)	Returns whether this is the primary key for this table.
ReferencedIndexName as String (read only)	The name of the index referenced by this index if it is a foreign key
ReferencedTableName as String (read only)	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 NULL 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.

ULPublicationSchema class

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

Properties

Prototype	Description
Mask as Long (read only)	Returns the mask (a unique identifier) for the publication
Name as String (read only)	Returns 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.

ULSQLCode enum

The ULSQLCode constants identify SQL codes.

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

Constant	Value
ulSQLE_BAD_ENCRYPTION_KEY	-840
ulSQLE_CANNOT_ACCESS_FILE	-602
ulSQLE_CANNOT_CHANGE_USER_NAME	-867
ulSQLE_COLUMN_CANNOT_BE_NULL	-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
ulSQLE_DATABASE_UPGRADE_FAILED	-672
ulSQLE_DATABASE_UPGRADE_NOT_POSSIBLE	-673
ulSQLE_DATATYPE_NOT_ALLOWED	-624
ulSQLE_DBSPACE_FULL	-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_MULTIUSER	-89

Constant	Value
ulSQLE_ERROR	-300
ulSQLE_IDENTIFIER_TOO_LONG	-250
ulSQLE_INDEX_NOT_FOUND	-183
ulSQLE_INDEX_NOT_UNIQUE	-196
ulSQLE_INTERRUPTED	-299
ulSQLE_INVALID_FOREIGN_KEY	-194
ulSQLE_INVALID_FOREIGN_KEY_DEF	-113
ulSQLE_INVALID_LOGON	-103
ulSQLE_INVALID_OPTION_SETTING	-201
ulSQLE_INVALID_PARAMETER	-735
ulSQLE_INVALID_SQL_IDENTIFIER	-760
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_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_STRING_RIGHT_TRUNCATION	-638
ulSQLE_TABLE_HAS_PUBLICATIONS	-281
ulSQLE_TABLE_IN_USE	-214

Constant	Value
ulSQLE_TABLE_NOT_FOUND	-141
ulSQLE_TOO_MANY_CONNECTIONS	-102
ulSQLE_UNABLE_TO_START_DATABASE	-82
ulSQLE_UNCOMMITTED_TRANSACTIONS	-755
ulSQLE_UNKNOWN_USERID	-140
ulSQLE_UNSUPPORTED_CHARACTER_SET_ERROR	-869
ulSQLE_UPLOAD_FAILED_AT_SERVER	-794

ULSQLType enum

The ULSQLType constants identify valid database column types.

Constant	Value
ULTypeLong	0
ULTypeShort	1
ULTypeUnsignedLong	2
ULTypeUnsignedShort	3
ULTypeBig	4
ULTypeUnsignedBig	5
ULTypeByte	6
ULTypeBit	7
ULTypeDateTime	8
ULTypeDate	9
ULTypeTime	10
ULTypeDouble	11
ULTypeReal	12
ULTypeBinary	13
ULTypeLongBinary	14
ULTypeString	15
ULTypeLongString	16
ULTypeNumeric	17

ULStreamErrorCode enum

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
ulStreamErrorCodeSecureDuplicateContext	25
ulStreamErrorCodeSecureSetIo	26

Constant	Value
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
ulStreamErrorCodeSecureHandshake	53
ulStreamErrorCodeHttpVersion	54
ulStreamErrorCodeSecureSetReadFunc	55
ulStreamErrorCodeSecureSetWriteFunc	56
ulStreamErrorCodeSocketHostNameNotFound	57

Constant	Value
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

ULStreamErrorContext enum

The ULStreamErrorContext constants identify constants you can use to specify ULStreamErrorContext.

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 enum

The ULStreamErrorID constants identify constants you can use to specify ULStreamErrorContext.

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

ULStreamType enum

The ULStreamType constants identify constants you can use to specify stream type.

Constant	Value	Description
ulUnknown	0	No stream type has been set. You must set a stream type before synchronization.
ulTCPIP	1	TCPIP 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	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
DownloadOnly as Boolean	If true, synchronization only downloads data
NewPassword as String	The user's password will be changed to this string on the next synchronization, if set
Password as String	Password corresponding to the given user name
PingOnly as Boolean	Only check the server for liveness. Do not synchronize data.
PublicationMask as Long	The publications to synchronize - the default is all
SendColumnNames as Boolean	If true, column names are sent to the MobiLink synchronization server
SendDownloadAck as Boolean	If true, a download acknowledgement is sent during synchronization
Stream as ULStreamType constants	The type of stream to use during synchronization
StreamParms as String	Extra parameters for the given stream type
UploadOnly as Boolean	If true, synchronization only uploads data
UserName as String	User name to connect 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
```

ULSyncResult class

The attributes of the ULSyncResult object indicate how the last synchronization went. Note that data for this object is not saved if the application is terminated.

Properties

The following are properties of ULSyncResult:

Prototype	Description
AuthStatus as ULAuthStatusCode (read only)	The authorization status code for the last synchronization.
IgnoredRows as Boolean (read only)	If true, rows were ignored during the last synchronization
StreamErrorCode as ULStreamErrorCode (read only)	The error code reported by the stream itself
StreamErrorContext as ULStreamErrorContext (read only)	The basic network operation being performed
StreamErrorID as ULStreamErrorID (read only)	The network layer reporting the error
StreamErrorSystem as Long (read only)	The stream error system-specific code
UploadOK as Boolean (read only)	If true, data was uploaded successfully in the last synchronization

ULSyncState

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.

Properties

Prototype	Description
BOF as Boolean (read only)	Returns whether you are currently positioned before the first row
EOF as Boolean (read only)	Returns whether you are currently positioned after the last row
IsOpen as Boolean (read only)	Returns whether or not this table is currently open
RowCount as Long (read only)	Returns the number of rows in this table
Schema as ULTableSchema (read only)	Returns information about the schema of this table.

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 Palm, 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 **ULColumn** object for the specified column name.

For information about the **ULColumn** object, see "ULColumn" on page 61.

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

Returns Returns the ULColumn 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.StartSynchronizationDelete** 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 104.

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 104.

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 104.

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.

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 105

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 104.

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 104.

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.

Example In this example, MoveFirst takes the cursor to the first record.

```
CustomerTable.MoveFirst
```

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.

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.

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.

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** The name of the index.

persistent_name For Palm Computing Platform applications, 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 72.

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.

Examples

```
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 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)	True if the table is always excluded from synchronization. Otherwise, false.
PrimaryKey as ULIndexSchema (read only)	The primary key for this table.
UploadUnchangedRows	True if all of this tables rows are sent to the consolidated database during synchronization even if they haven't changed.

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 80.

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 database. 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.

InPublication method

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

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

Returns **True** if the table is part of the publication.

False if the table is not part of the publication.

Index

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W