



# UltraLite<sup>™</sup> for eMbedded Visual Basic User's Guide

Last modified: October 2002  
Part Number: 36293-01-0802-01

Copyright © 1989–2002 Sybase, Inc. Portions copyright © 2001–2002 iAnywhere Solutions, Inc. All rights reserved.

No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of iAnywhere Solutions, Inc. iAnywhere Solutions, Inc. is a subsidiary of Sybase, Inc.

Sybase, SYBASE (logo), AccelaTrade, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, AnswerBase, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-FORMS, APT-Library, APT-Translator, ASEP, Backup Server, BayCam, Bit-Wise, BizTracker, Certified PowerBuilder Developer, Certified SYBASE Professional, Certified SYBASE Professional (logo), ClearConnect, Client Services, Client-Library, CodeBank, Column Design, ComponentPack, Connection Manager, Convoy/DM, Copernicus, CSP, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, Dynamo, e-ADK, E-Anywhere, e-Biz Integrator, E-Whatever, EC-GATEWAY, ECMAP, ECRT, eFulfillment Accelerator, Electronic Case Management, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, eProcurement Accelerator, eremote, Everything Works Better When Everything Works Together, EWA, Financial Fusion, Financial Fusion Server, First Impression, Formula One, Gateway Manager, GeoPoint, iAnywhere, iAnywhere Solutions, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InstaHelp, Intellidex, InternetBuilder, iremote, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, Logical Memory Manager, MainframeConnect, Maintenance Express, MAP, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, MethodSet, ML Query, MobiCATS, MySupport, Net-Gateway, Net-Library, New Era of Networks, Next Generation Learning, Next Generation Learning Studio, O DEVICE, OASiS, OASiS (logo), ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Biz, Open Business Interchange, Open Client, Open Client/Server, Open Client/Server Interfaces, Open ClientConnect, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, Partnerships that Work, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, PhysicalArchitect, Pocket PowerBuilder, PocketBuilder, Power Through Knowledge, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, Powering the New Economy, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, Powersoft Portfolio, Powersoft Professional, PowerStage, PowerStudio, PowerTips, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, Rapport, Relational Beans, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Report Workbench, Report-Execute, Resource Manager, RW-DisplayLib, RW-Library, S Designer, S-Designer, S.W.I.F.T. Message Format Libraries, SAFE, SAFE/PRO, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL Server SNMP SubAgent, SQL Server/CFT, SQL Server/DBM, SQL SMART, SQL Station, SQL Toolset, SQLJ, Stage III Engineering, Startup.Com, STEP, SupportNow, Sybase Central, Sybase Client/Server Interfaces, Sybase Development Framework, Sybase Financial Server, Sybase Gateways, Sybase Learning Connection, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase Synergy Program, Sybase User Workbench, Sybase Virtual Server Architecture, SybaseWare, Syber Financial, SyberAssist, SybMD, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, The Enterprise Client/Server Company, The Extensible Software Platform, The Future Is Wide Open, The Learning Connection, The Model For Client/Server Solutions, The Online Information Center, The Power of One, TradeForce, Transact-SQL, Translation Toolkit, Turning Imagination Into Reality, UltraLite, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Viewer, Visual Components, VisualSpeller, VisualWriter, VQL, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, WarehouseArchitect, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server, and XP Server are trademarks of Sybase, Inc. or its subsidiaries.

Certicom, MobileTrust, and SSL Plus are trademarks and Security Builder is a registered trademark of Certicom Corp. Copyright © 1997–2000 Certicom Corp. Portions are Copyright © 1997–1998, Consensus Development Corporation, a wholly owned subsidiary of Certicom Corp. All rights reserved. Contains an implementation of NR signatures, licensed under U.S. patent 5,600,725. Protected by U.S. patents 5,787,028; 4,745,568; 5,761,305. Patents pending.

All other trademarks are property of their respective owners.

Last modified October 2002. Part number 36293-01-0802-01.

# Contents

	<b>About This Manual.....</b>	<b>v</b>
	The UltraLite sample database .....	vi
	Finding out more and providing feedback.....	vii
<b>1</b>	<b>Introduction to UltraLite for eMbedded Visual Basic .....</b>	<b>1</b>
	UltraLite for eMbedded Visual Basic features.....	2
	System requirements and supported platforms .....	3
	UltraLite for eMbedded Visual Basic architecture.....	4
<b>2</b>	<b>Tutorial: An UltraLite for eMbedded Visual Basic</b>	
	<b>Application .....</b>	<b>7</b>
	Introduction .....	8
	Lesson 1: Create a database schema .....	9
	Lesson 2: Create a project architecture.....	11
	Lesson 3: Design the application form.....	13
	Lesson 4: Configure the emulator to support	
	UltraLite applications.....	14
	Lesson 5: Write the Visual Basic sample code.....	16
	Lesson 6: Deploy to a device.....	25
	Summary.....	26
<b>3</b>	<b>Understanding UltraLite for eMbedded Visual Basic</b>	
	<b>Development .....</b>	<b>27</b>
	Preparing to work with eMbedded Visual Basic.....	28
	Working with UltraLite databases .....	30
	Connecting to the UltraLite database.....	32
	Accessing and manipulating data .....	35
	Accessing schema information .....	41
	Error handling.....	42
	User authentication .....	43
	Synchronizing UltraLite applications .....	44

---

<b>API Reference .....</b>	<b>47</b>
IULColumns collection .....	49
IULIndexSchemas collection.....	50
IULPublicationSchemas collection.....	51
ULAuthStatusCode constants.....	52
ULColumn class.....	53
ULColumnSchema class.....	58
ULConnection class .....	59
ULDatabaseManager class.....	64
ULDatabaseSchema class.....	69
ULIndexSchema class .....	71
ULPublicationSchema class .....	72
ULSQLCode constants .....	73
ULSQLType constants.....	76
ULStreamErrorCode constants.....	77
ULStreamErrorContext constants.....	80
ULStreamErrorID constants.....	81
ULStreamType .....	82
ULSyncMasks Type .....	83
ULSyncParms class.....	84
ULSyncResult class .....	85
ULSyncState constants.....	86
ULTable class .....	87
ULTableSchema class .....	96
 <b>Index.....</b>	 <b>97</b>

# About This Manual

Subject	This manual describes UltraLite for eMbedded Visual Basic, which is part of the UltraLite Component Suite. With UltraLite for eMbedded Visual Basic you can develop and deploy database applications to handheld, mobile, or embedded devices running Windows CE.
Audience	This manual is intended for eMbedded Visual Basic 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. Familiarity with eMbedded Visual Basic is assumed.

---

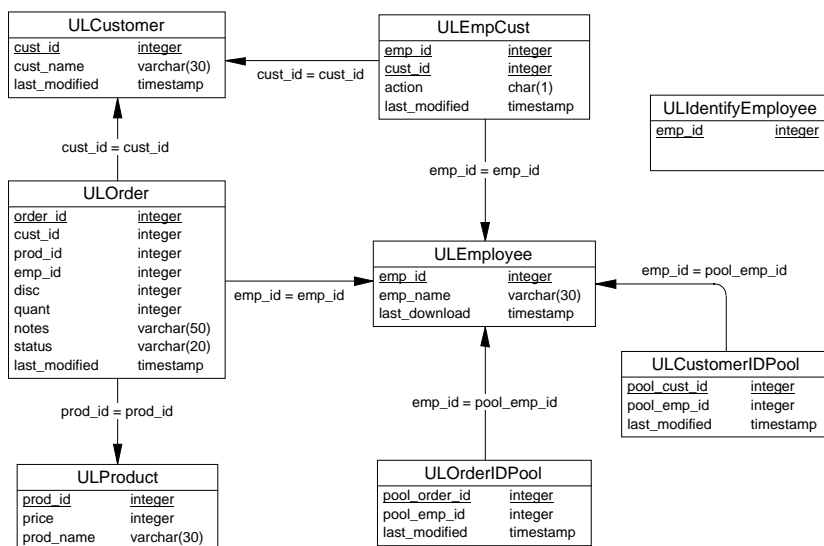
# The UltraLite sample database

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

The UltraLite sample database is held in a file named *custdb.db*, and is located in the *Samples\UltraLite\CustDB* subdirectory of your SQL Anywhere directory. A complete application built on this database is also supplied as *Samples\UltraLiteActiveX\CustDB\evb2002.ebp* and *evbPocketPC.ebp*.

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 and via e-mail. The newsgroup can be found on the *forums.sybase.com* news server as `news://forums.sybase.com/ianywhere.private.ultralitetools.beta`. The e-mail address is `ulbeta@ianywhere.com`.

### **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 eMbedded Visual Basic

About this chapter      This chapter introduces you to UltraLite for eMbedded Visual Basic features, supported platforms, architecture, and functionality.

### Contents

<b>Topic</b>	<b>Page</b>
UltraLite for eMbedded Visual Basic features	2
System requirements and supported platforms	3
UltraLite for eMbedded Visual Basic architecture	4

## UltraLite for eMbedded Visual Basic features

UltraLite for eMbedded Visual Basic 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 Microsoft eMbedded Visual Basic development tool
- ◆ deployment on 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**.

### Supported platforms

Development platforms	To develop applications using UltraLite, you require Visual Basic 6 or eMbedded Visual Basic Version 3.0.
Target platforms	UltraLite for eMbedded Visual Basic targets Windows CE 3.0 and higher, PocketPC emulator, MIPS and ARM devices and PocketPC 2002 on ARM.

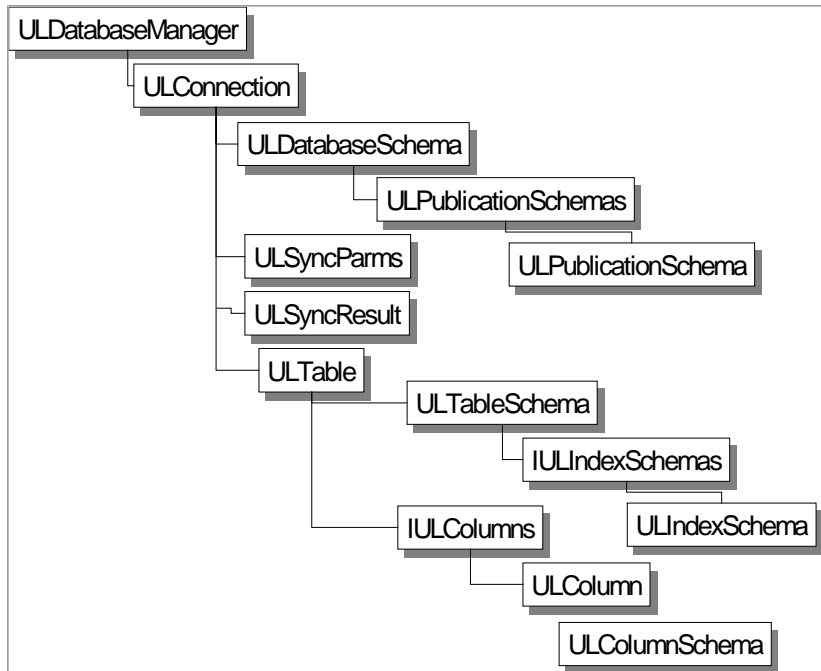
### SQL Anywhere Studio

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 eMbedded Visual Basic architecture

UltraLite for eMbedded Visual Basic provides a database engine for Windows CE. It provides a eMbedded Visual Basic ActiveX that exposes a set of objects for data manipulation using the UltraLite database.



Some of the more commonly-used high level objects are:

- ◆ **ULDatabaseManager** allows you to open connections and set an active listener. The ULDatabaseManager is the starting point for your eMbedded Visual Basic 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 64.

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

🔗 For more information on ULConnection, see "ULConnection class" on page 59.

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

☞ For more information on the ULTable, ULColumn, and ULIndexSchema objects, see "ULTable class" on page 87 and "ULColumn class" on page 53.

- ◆ **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 User's Guide* in the SQL Anywhere Studio.



## CHAPTER 2

# Tutorial: An UltraLite for eMbedded Visual Basic Application

### About this chapter

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

### Contents

Topic	Page
Introduction	8
Lesson 1: Create a database schema	9
Lesson 2: Create a project architecture	11
Lesson 3: Design the application form	13
Lesson 4: Configure the emulator to support UltraLite applications	14
Lesson 5: Write the Visual Basic sample code	16
Lesson 6: Deploy to a device	25
Summary	26

# Introduction

This tutorial walks you through building an UltraLite for eMbedded Visual Basic application. At the end of the tutorial you will have an application and small database on 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 numerous competencies:

- ◆ you can program Microsoft eMbedded Visual Basic 3
  - ◆ you can test and troubleshoot a eMbedded Visual Basic 3 application
  - ◆ you can add references and components as needed
  - ◆ you can register a control on Windows CE using Control Manager
  - ◆ you can use the Visual Basic Object Browser and navigate the eMbedded Visual Basic 3 environment
- ◆ you can use command line options and parameters

## Goals

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

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

- ◆ A way to identify the schema on the development machine so it can be copied to 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\evb*. If you create your tutorial directory elsewhere, supply the path to your location instead of *c:\tutorial\evb* 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 Schema Painter.

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

- ◆ Open the Tools folder and double-click Create UltraLite schema file.
- ◆ In the file dialog box, type **c:\tutorial\evb\tutcustomer.usm** or Browse to the folder and enter **tutcustomer**.
- ◆ Click Open 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 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\evb*, the same one holding your schema file, is where you will store your application files.

The first step is to create an eMbedded Visual Basic project for your application.

The UltraLite component for eMbedded Visual Basic development is named **iAnywhere Solutions ActiveX for UltraLite**. eMbedded Visual Basic used a desktop version of the UltraLite component in order to use UltraLite objects in your code.

### ❖ To create an UltraLite component reference:

- 1 Start eMbedded Visual Basic.
  - ◆ Click Start ► Programs ► Microsoft eMbedded Visual Tools ► eMbedded Visual Basic 3.0. The New Project window appears.
  - ◆ Choose a target of your choice and click OK.
- 2 Create a reference to the UltraLite component for eMbedded Visual Basic:
  - ◆ Click Project ► References.
  - ◆ If this is the first time you have run eMbedded Visual Basic with UltraLite, add the control to the list of available references:
    - ◆ Click Browse.
    - ◆ Browse to the *UltraLite\UltraLiteActiveX\win32\* directory.
    - ◆ Select *uldo8.dll* and click OK.
    - ◆ iAnywhere Solutions ActiveX for UltraLite is added to the list of references.
  - Your eMbedded Visual Basic environment is now capable of supporting UltraLite.
  - ◆ Select iAnywhere Solutions ActiveX for UltraLite and click OK to add the control to your project.
- 3 Save the Project:
  - ◆ Choose File ► Save Project.
  - ◆ Save the form as **c:\tutorial\evb\Form1.frm**.
  - ◆ Save the project as **c:\tutorial\evb\Form1.frm**.

You are now ready to design your application.

## Lesson 3: Design the application form

You are now ready to design your application form.

❖ **To design the form:**

- 1 Add the controls and the properties given in the table below to Form1:

Type	Name	Caption
TextBox	txtfname	
TextBox	txtlname	
TextBox	txtcity	
TextBox	txtphone	
Label	lblID	
Button	btnInsert	Insert
Button	btnUpdate	Update
Button	btnDelete	Delete
Button	btnNext	Next
Button	btnPrevious	Previous
Button	btnSync	Synchronize

- 2 Run your application to confirm that your setup is configured correctly.

- ◆ Click Run ►Execute. The application appears in the Windows CE emulator.

At this stage there is no UltraLite dependence in your application. If you have problems at this stage, check your Windows CE embedded tools setup.

- ◆ Click the OK button at the top right of the form to end the application.

You are now ready to add code to the application.

## Lesson 4: Configure the emulator to support UltraLite applications

Once you add UltraLite code to your application, you must add the UltraLite control to the emulator in order to debug and test your application. This lesson describes how to add the UltraLite control to the emulator.

### ❖ To configure the emulator for UltraLite applications:

- 1 Start the Control Manager.
  - ◆ Select Tools ► Remote Tools ► Control Manager.
- 2 Select the target device:
  - ◆ In the left pane, open Pocket PC and double-click Pocket PC Emulation.
- 3 Add the UltraLite control
  - ◆ Click Control ► Add New Control.
  - ◆ Browse to the device-specific version of the UltraLite control. Controls are stored in the subdirectories for each CE device processor. The control for the emulator is the file *uldo8.dll* held in the *ultralite\UltraLiteActiveX\ce\emulator30* subdirectory of your SQL Anywhere installation.
  - ◆ Click OK

In addition to the UltraLite control, you must deploy the database schema. In the next lesson you will write code so that when your application first connects to a database, it uses the schema to create the database file.

### ❖ To deploy the database schema file to the emulator:

- 1 Start the Windows CE File Viewer:
  - ◆ From eMbedded Visual Basic, click Tools ► Remote Tools ► File Viewer. The File viewer starts.
  - ◆ Ensure that you are in the root directory of the emulator by double clicking Pocket PC Emulation in the left pane.
- 2 Create a folder to hold your application:
  - ◆ In File Viewer, click File ► New Folder.
  - ◆ Create a folder named *tutorial*. This folder is used to hold your application files.

- ◆ Double click *tutorial* to navigate to that folder.
  - 3 Deploy the schema file to the emulator:
    - ◆ Click File ► Export File.
    - ◆ Navigate to your *c:\tutorial\levb* directory and double-click *tutcustomer.usm*. The schema file is deployed to the emulator.
- You are now ready to write UltraLite code and test it in the emulator.

## Lesson 5: Write the Visual Basic sample code

A connection to an UltraLite database requires a *ULDatabaseManager* object. You must create this object, and then use it to open an UltraLite database.

### Write code for connection to your database

In this application, you connect to the database in the Form Load event, but you can also use a general module.

#### ❖ 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 in the General area of your form. This code declares *ULDatabaseManager*, a connection, a table and three database columns:

```
Dim DatabaseMgr As ULDatabaseManager
Dim Connection As ULConnection
Dim CustomerTable As ULTable
Dim colID, colFirstName, colLastName As ULColumn
```

- 2 Add 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  
    On Error Resume Next  
    conn_parms = "uid=DBA;pwd=SQL"  
    open_parms = conn_parms & _  
        ";ce_file=\tutorial\tutCustomer.udb"  
    schema_parms = open_parms & _  
        ";ce_schema=\tutorial\tutCustomer.usm"  
  
    Set DatabaseMgr = _  
        CreateObject("UltraLite.ULDatabaseManager")  
    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 <> 0 Then  
            MsgBox Err.Description  
            Err.Clear  
        End If  
        MsgBox "Connected to a new database"  
    End If  
End Sub
```

- ◆ CreateObject is used to create the initial database manager object. Note that *lblStatus* shows you the status of your connection.
  - ◆ The error handling code checks if the connection is established. If a database file exists, the initial connection attempt fails.
- 3 Run the application in the development environment.
- ◆ Choose Run ►Execute.
  - ◆ 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 the OK button in the top right corner to terminate the application.
  - ◆ If you wish, you can use File Viewer to check that a database file (*tutcustomer.udb*) has been created on the emulator.

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

## Write code for data manipulation

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

### ❖ To open the table:

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

Add the following code to the Form\_Load routine, just before the End Sub instruction:

```
Set CustomerTable = Connection.GetTable("customer")
CustomerTable.Open
CustomerTable.MoveBeforeFirst
```

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.Columns("ID").Value
        txtFname.Text = _
        CustomerTable.Columns("Fname").Value
        txtLname.Text = _
        CustomerTable.Columns("Lname").Value
        txtCity.Text = _
        CustomerTable.Columns("City").SValue
        txtPhone.Text = _
        CustomerTable.Columns("Phone").Value
    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.

#### ❖ 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  
  
    CustomerTable.InsertBegin  
    CustomerTable.Columns("Fname").Value = _  
        fname  
    CustomerTable.Columns("Lname").Value = _  
        lname  
    If Len(city) > 0 Then  
        CustomerTable.Columns("City").Value = _  
            city  
    End If  
    If Len(phone) > 0 Then  
        CustomerTable.Columns("Phone").Value = _  
            phone  
    End If  
    CustomerTable.Insert  
    CustomerTable.MoveLast  
    DisplayCurrentRow  
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 OK 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  
    CustomerTable.UpdateBegin  
    CustomerTable.Columns("Fname").Value = _  
        fname  
    CustomerTable.Columns("Lname").Value = _  
        lname  
    If Len(city) > 0 Then  
        CustomerTable.Columns("City").Value = _  
            city  
    End If  
    If Len(phone) > 0 Then  
        CustomerTable.Columns("Phone").Value = _  
            phone  
    End If  
    CustomerTable.Update  
    DisplayCurrentRow  
    Exit Sub  
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 standalone 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.

You have now successfully coded your application.

## Write code to synchronize

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

Synchronization can be done using, for example, using a button control called *Synchronize*, which may have the following structure:

```
Call MyConnection.Synchronize
```

❖ **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()  
    Dim parms As ULSyncParms  
    Dim result As ULSyncResult  
    On Error Resume Next  
  
    Set parms = Connection.SyncParms  
    Set result = Connection.SyncResult  
    parms.UserName = "ULevbUser"  
    parms.Stream = ULStreamType.ulTCPIP  
    parms.Version = "ul_default"  
    parms.SendColumnNames = True  
    Connection.Synchronize (False)  
    If Err.Number <> _  
        ULSQLCode.ulSQLE_NOERROR Then  
        MsgBox result.StreamErrorCode  
    End If  
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.

## Synchronize your application

The next step is to synchronize the data in your database.

### ❖ To synchronize data:

- 1 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 8.0 Sample database.

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

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

Any rows in the table would be uploaded to the customer table in the ASA 8.0- Sample database.

- 4 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 6: Deploy to a device

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

### ❖ To deploy to a device:

- 1 Ensure that the iAnywhere Solutions, ActiveX for UltraLite is available on the target CE device.

- ◆ Select Tools►Remote Tools

- ◆ Select Control Manager.

This starts the Windows CE Control Manager.

- 2 Select your target device from the list presented and the hardware you are using. The right hand display shows the controls available on that device. The control is *ULDatabaseManager* class.

You can also copy the dll to \Windows directory on the device and register it with regsvrce, or you can use Tools►Remote Tools►Control Manager to select the target device. But know that if you use this method, you are using an ActiveX, not a control, thus you will not see the control in the list, even if it is present.

Be sure to use the device-specific version of the DLL. Controls are stored in subdirectories of your SQL Anywhere installation for each CE device processor, as follows:

- ◆ **ARM** *ultralite\UltraLiteActiveX\ce\arm\uldo8.dll*

- ◆ **Emulator** *ultralite\UltraLiteActiveX\ce\emulator30\uldo8.dll*

- ◆ **MIPS** *ultralite\UltraLiteActiveX\ce\mips\uldo8.dll*

## Summary

During this tutorial, you:

- ◆ Created a sample database using the UltraLite Schema Painter.
- ◆ Created an UltraLite for eMbedded Visual Basic application
- ◆ Synchronized a remote database with an Adaptive Server Anywhere consolidated database using UltraLite.
- ◆ Gained competence with the process of developing an UltraLite for eMbedded Visual Basic application

### Samples

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

- ◆ *Samples\UltraLiteActiveX\custdb\evbPocketPC.evb*
- ◆ *Samples\UltraLiteActiveX\custdb\evb2002.evb*
- ◆ *Samples\UltraLiteActiveX\dbview.evb\evb2002.evb*
- ◆ *Samples\UltraLiteActiveX\dbview.evb\pocketpc.evb*

C H A P T E R   3

Understanding UltraLite for eMbedded Visual Basic Development

About this chapter      This chapter describes how to develop applications with the UltraLite for eMbedded Visual Basic.

Contents	<table><tr><th>Topic</th><th>Page</th></tr><tr><td>Preparing to work with eMbedded Visual Basic</td><td>28</td></tr><tr><td>Working with UltraLite databases</td><td>30</td></tr><tr><td>Connecting to the UltraLite database</td><td>32</td></tr><tr><td>Accessing and manipulating data</td><td>35</td></tr><tr><td>Accessing schema information</td><td>41</td></tr><tr><td>Error handling</td><td>42</td></tr><tr><td>User authentication</td><td>43</td></tr><tr><td>Synchronizing UltraLite applications</td><td>44</td></tr></table>	Topic	Page	Preparing to work with eMbedded Visual Basic	28	Working with UltraLite databases	30	Connecting to the UltraLite database	32	Accessing and manipulating data	35	Accessing schema information	41	Error handling	42	User authentication	43	Synchronizing UltraLite applications	44
Topic	Page																		
Preparing to work with eMbedded Visual Basic	28																		
Working with UltraLite databases	30																		
Connecting to the UltraLite database	32																		
Accessing and manipulating data	35																		
Accessing schema information	41																		
Error handling	42																		
User authentication	43																		
Synchronizing UltraLite applications	44																		

## Preparing to work with eMbedded Visual Basic

There are several steps you must take before you can build UltraLite applications with eMbedded Visual Basic.

### Adding the UltraLite component to the design environment

To have access to UltraLite objects at development time, you must add the UltraLite component to the interface.

❖ **To add the UltraLite component to the eMbedded Visual Basic design environment:**

- 1 From the eMbedded Visual Basic menu, choose Project ►References.
- 2 If iAnywhere Solutions ActiveX for UltraLite 8.0 is not included in the list of references, click Browse.
- 3 Set the dialog to display All Files. Locate the file *uldo8.dll* in the *UltraLite\UltraLiteActiveX\win32* subdirectory of your SQL Anywhere directory. The iAnywhere Solutions ActiveX for UltraLite 8.0 control is added to the list of references.
- 4 Check iAnywhere Solutions ActiveX for UltraLite 8.0 and click OK to add the component to your project.

### Adding the UltraLite component to the device

To debug applications in the emulator, you must add the UltraLite component to the emulator. To deploy applications to your device, you must add the UltraLite component to the device. Both of these tasks can be carried out using the Windows CE Control Manager.

❖ **To add the UltraLite component to the device or emulator:**

- 1 From the eMbedded Visual Basic menu, choose Tools ►Remote Tools ►Control Manager.
- 2 In the left pane, open the device you are developing for, such as Pocket PC.
- 3 Open the device to which you are deploying, such as Pocket PC Emulation.
- 4 On the right pane, right click and choose Add New Control from the popup menu.

- 5 For the Emulator, navigate to the *UltraLite\UltraLiteActiveX\ce\emulator30* subdirectory of your SQL Anywhere directory. For a real device, navigate to the proper subdirectory of the *UltraLite\UltraLiteActive\ce* subdirectory of your SQL Anywhere directory for the chip used by your device.
- 6 Choose *uldo8.dll* from the directory. The `ULDatabaseManager` class is added to the device.

## Copying an UltraLite database to the device

In addition, you must add the UltraLite database file or schema file to the device.

### ❖ To add a schema file or database file to the device or emulator:

- 1 From the eMbedded Visual Basic menu, choose Tools ► Remote Tools ► File Viewer.
- 2 If your device is not shown in the left pane, connect to the device:
  - ◆ From the File Viewer menu, choose Connection ► Add Connection.
  - ◆ Select your device from the list and click OK to establish a connection.
- 3 Copy the schema file or database file to the device
  - ◆ Select a destination directory on the device.

It is often convenient to copy the file into the root directory of the device.
  - ◆ Choose File ► Export File.
  - ◆ Locate the schema file (.usm) or database file (.udb) on your desktop machine file system.
  - ◆ Click OK to export the file to the device.

## Working with UltraLite databases

UltraLite databases are files with a *.udb* extension. UltraLite databases are relational databases, and contain the following types of object:

- ◆ **Tables** A single UltraLite database can hold many tables. Relational database tables have a fixed number of columns, but can have any number of rows (up to a limit determined by the operating system on which you run). Each row has a single entry for each column. The special NULL entry is used when there is no value for the entry. When designing your database, each table should represent a separate type of item, such as Customers, Employees, and so on.
- ◆ **Indexes** The rows in a relational database table are not ordered. You can create indexes to access the rows in order. Indexes are commonly associated with a single column, but may also be associated with multiple columns.
- ◆ **Keys** Each table has a special index called the primary key. Entries in the primary key column or columns must be unique.

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

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

By enforcing referential integrity, UltraLite ensures that the data in your UltraLite database is correct, in the same manner that data elsewhere in the enterprise is correct.

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

Publications define sets of data to be synchronized. It is often desirable to synchronize all the data in an UltraLite database, but publications provide extra flexibility and control.

The schema and schema file

The database **schema** is the database without the data. It is the collection of tables, indexes, and so on within the database, and all the relationships between them.

You do not alter the schema of an UltraLite database directly. Instead, you create a schema file (which typically has the extension *.usm*) and upgrade the database schema from that file using a built-in UltraLite function in your application.

This process of creating a schema file and upgrading the database from the file applies both to the initial creation of the database and any subsequent schema changes.

## Creating UltraLite database schema files

You can create an UltraLite schema file in the following ways:

- ◆ **Schema painter** The UltraLite schema painter is a graphical utility for creating and editing UltraLite schema files.

To start the Schema painter, choose Start►Programs►Sybase SQL Anywhere 8►UltraLite Schema Painter, or double-click a schema file (with extension *usm*) in WIndows Explorer.

- ◆ **Generate the schema from an Adaptive Server Anywhere database** If you have the Adaptive Server Anywhere database management system, you can generate an UltraLite schema file using the *ulinit* command line utility.

You apply the schema file to the database from the UltraLite application. For more information, see "Connecting to the UltraLite database" on page 32.

## 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 DatabaseManager object.

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

The following code creates a DatabaseManager object named dbMgr, and makes a connection to a database.

```
Dim conn_parms As String
Dim open_parms As String
Dim schema_parms As String
On Error Resume Next
conn_parms = "uid=DBA;pwd=SQL"
open_parms = conn_parms & ";" & _
"ce_file=\tutCustomer.udb"
schema_parms = open_parms & ";" & _
"ce_schema=\tutCustomer.usm"

Set DatabaseMgr = _
CreateObject("UltraLite.ULDatabaseManager")
Set Connection = _
DatabaseMgr.openConnection(open_parms)
If Err.Number = _
ULSQLCodeConstants.ulSQLE_DATABASE_NOT_FOUND Then
Err.Clear
Set Connection = _
DatabaseMgr.CreateDatabase(schema_parms)
If Err.Number <> 0 Then
MsgBox Err.Description
Err.Clear
End If
End If
```

 For more details, see the code in *Samples\UltraLiteForEVB\dbview.evb\evb2002.evp* under your SQL Anywhere directory.

- 2 Open a connection to the database.



The `ULDatabaseManager.OpenConnection` method returns an open connection as a `Connection` object. This method takes a single string as its argument. The string is composed of a set of keyword=value pairs.

☞ 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 *tutcustomer.udb* in the root directory of the device.

```
conn_parms = "uid=DBA;pwd=SQL"
open_parms = conn_parms & ";" & _
"ce_file=\tutCustomer.udb"

Set Connection = _
DatabaseMgr.openConnection(open_parms)
```

### 3 Applying a new file to your schema

If you want to modify your existing database structure, you can do so with the `ApplyFile` method. In most cases there will be 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. The example below shows how you can use the `ApplyFile` method to send in new specifications for your database via a schema you design.

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

☞ For more information on `CreateDatabase`, `OpenConnection` and the `ApplyFile` method, see "CreateDatabase method" on page 64, "OpenConnection method" on page 68 and "ApplyFile method" on page 70.

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. Performance is better when AutoCommit is off and commits are performed directly.

☞ For more information, see "Transaction processing in UltraLite" on page 39.

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

🔗 For more information, see "User authentication" on page 43

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

🔗 For more information, see "Synchronizing UltraLite applications" on page 44.

- ◆ **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.

### 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 third column (which holds the last name of the customer) for each row.

```
Dim tCustomer as ULTable
Set tCustomer = conn.GetTable( "customer" )
tCustomer.Open
Set tCustomer = conn.GetTable("customer")
Set colLastName = tCustomer.Columns.(3)
tCustomer.MoveBeforeFirst
While tCustomer.MoveNext
    MsgBox colLastName.Value
Wend
```

The code above shows how the columns of the table are contained in a Columns collection. You can address columns by index number (the order in which they were created in the .usm file) or by name. To get the name of a column, you can use its Schema property:

```
colname = colLastName.Schema.name
```

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= Connection.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 ULColumn.Value property to get the value of that column for the current row. For example, the following code retrieves the value of three columns from the tcustomer ULTable object, and displays them in text boxes:

```
Dim colID, colFirstName, colLastName As ULColumn
Set colID = tCustomer.Columns.Item(1)
Set colFirstName = tCustomer.Columns.Item(2)
Set colLastName = tCustomer.Columns.Item(3)

txtID.Text = colID.Value
txtFirstName.Text = colFirstName.Value
txtLastName.Text = colLastName.Value
```

You can also use the Value property to set values. For example:

```
colLastName.Value = "Kaminski"
```

By assigning values to these properties you do not alter the value of the data in the database. You can assign values to the properties even if you are before the first row or after the last row of the table, but it is an error to try to access data when the current row is in one of these positions, for example:

```
' This code is incorrect
tCustomer.MoveBeforeFirst
id = colID.Value
```

### Casting values

As the Value method returns a variant, you can use it to access columns of any data type.

## Searching for rows with find and lookup

UltraLite has several modes of operation when working with data. Two of these modes are used for searching: the find and lookup modes. 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.

- ◆ **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:

```
ColLastName.Value = "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 use one of the other find or lookup methods to search using only a limited number of columns.

## 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 the table tCustomer:

```
tCustomer.UpdateBegin
```

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

```
ColFirstName.Value = "Elizabeth"
```

### 4 Execute the Update.

```
tCustomer.Update
```

After the update operation the current row is the row that was just updated. If you changed the value of a column in the index specified when the ULTable object was opened, there are some subtleties to the positioning.

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 39.

#### **Caution**

*Do not update the primary key of a row: delete the row and add a new row instead.*

## Inserting rows

The steps to insert a row are 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. Rows are automatically sorted in the index used to open the table.

For example, the following sequence of instructions inserts a new row:

```
CustomerTable.InsertBegin  
CustomerTable.Columns("Fname").Value = fname  
CustomerTable.Columns("Lname").Value = lname  
CustomerTable.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, NULL is used. If the column does not allow NULL, the following defaults are used:

- ◆ For numeric columns, zero.
- ◆ For character columns, an empty string.

To set a value to NULL explicitly, use the setNull method.

The Insert is permanently save to the database 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. Note that performance can be faster when AutoCommit is off.

## Data manipulation internals

UltraLite exposes the rows in a table to your application one at a time. The Table 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 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.

```
colID.Value = 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 `Value` property to set values in the buffer.

- 3 Carry out the operation.

Use a `ULTable` method such as `Insert`, `Update`, `Find`, or `Lookup` to carry out the operation, using the values in the buffer. 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.



## 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, access the ULConnection.Schema property.
- ◆ **ULTableSchema** The number and names of columns in the table, as well as the Indexes collections for this table.  
  
To obtain a ULTableSchema object, access the ULTable.Schema property.
- ◆ **ULColumnSchema** Information about an individual column.  
  
To obtain a ULColumnSchema object, access the ULColumn.Schema property.
- ◆ **ULIndexSchema** Information about the column in the index. 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.  
  
The ULIndexSchema objects are available as part of the ULTableSchema.Indexes collection.
- ◆ **ULPublicationSchema** Tables and columns contained in a publication. Publications are also comprised of schema only, and so there is a ULPublicationSchema object rather than a ULPublication object.  
  
The ULPublicationSchema objects are available as part of the ULDatabaseSchema.Publications collection.

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

## Error handling

You can use the standard eMbedded Visual Basic error-handling features to handle errors. The Err object holds the SQLCODE value for an error. SQLCODE values are negative numbers indicating the particular kind of error. You can also get the last error with Connection.LastErrorCode.

**Note**

For users of SQL Anywhere Studio, the Adaptive Server Anywhere Error Messages manual is part of the SQL Anywhere Studio online books. If you do not have SQL Anywhere Studio, you can open this book by double-clicking *dberen8.chm* in the *docs* subdirectory of your SQL Anywhere directory.

## User authentication

There is a common sequence of events to managing user IDs and passwords.

- 1 New users have to be added from an existing connection. As all UltraLite databases are created with a default user ID and password of DBA and SQL, respectively, you must first connect as this initial user and implement user management only upon successful connection.
- 2 You cannot change a user ID: you add a user and delete an existing user. A maximum of four user IDs are permitted for each UltraLite database.
- 3 To change the password for an existing user ID, use the `ULConnection.GrantConnectTo` method.

## Synchronizing UltraLite applications

Users of SQL Anywhere Studio 8.0.1 can synchronize UltraLite applications with a central database. This database may be a desktop database for personal applications, or a multi-user database for shared data, including enterprise data. Synchronization requires the MobiLink synchronization software included with SQL Anywhere Studio.

Synchronization details can be found in the *MobiLink User's Guide* and the *UltraLite User's Guide* included with SQL Anywhere Studio documentation. This section provides a brief introduction to synchronization and describes some features of particular interest to users of the UltraLite Component Suite.

You can also find a working example of synchronization in the CustDB sample application. For Native UltraLite for Java, the CustDB sample is in the *Samples\UltraLiteActiveX\CustDB* directory.

UltraLite for eMbedded Visual Basic supports TCP/IP, and HTTP synchronization. TCP/IP and HTTP synchronization are initiated by the UltraLite application. In all cases, you use methods and properties of the ULConnection object to control synchronization.

Controlling TCP/IP  
and HTTP  
synchronization

### ❖ To control synchronization over TCP/IP or HTTP, your application must carry out the following sequence of operations:

- 1 Prepare the synchronization information.

Assign values to properties of the ULConnection.SyncInfo object.

🔗 For information about the properties and the values that you should set, look up **synchronization parameters: about** in the SQL Anywhere Studio online books index.

- 2 Synchronize.

Call the ULConnection.Synchronize method.

## Monitoring Synchronization progress

You can monitor the progress of your synchronizations by simply writing code into your UltraLite for eMbedded Visual Basic project. The code sample below shows how you can set your databasemanager object to work with the Synchronization Progress Dialog:

```
Set DBMgr = _  
CreateObjectWithEvents( _  
"UltraLite.ULDatabaseManager", "UL_")
```

Now, write code into your form that captures event notifications by the Synchroniztion Progress Dialog. The first method shows users insert, update and delete data when data is sent to the consolidated database.

```
Private Sub UL_OnSend(ByVal nBytes As Long, ByVal _
nInserts As Long, ByVal _
    nUpdates As Long, ByVal nDeletes As Long)_
    prLine "OnSend " & nBytes & " bytes, " & nInserts & _
        " inserts, " & nUpdates & " updates, " &
nDeletes & " deletes"_
End Sub
```

The second method shows users insert, update and delete data when data is received at the consolidated database.

```
Private Sub UL_OnReceive(ByVal nBytes As Long, ByVal _
nInserts As Long,_
    ByVal nUpdates As Long, ByVal nDeletes As Long)_
    prLine "OnReceive " & nBytes & " bytes, " & _
nInserts & " inserts, " & nUpdates & _
    " updates, " & nDeletes & " deletes"
End Sub
```

The third method shows users insert, update and delete data when data states are changing.

```
Private Sub UL_OnStateChange(ByVal newState As Long, _
ByVal oldState As Long)
    prLine "OnStateChange new:" & newState & ", old: " _
    & oldState
End Sub
```

The fourth method shows users insert, update and delete data when table data are changing.

```
Private Sub UL_OnTableChange(ByVal newTableIndex As _
Long, ByVal numTables As Long)
    prLine "OnTableChange index:" & newTableIndex & ",
#tables=" & numTables
End Sub
```



## CHAPTER 4

# API Reference

About this chapter

This chapter describes the UltraLite for eMbedded Visual Basic API.

Contents

Topic	Page
IULColumns collection	49
IULIndexSchemas collection	50
IULPublicationSchemas collection	51
ULAuthStatusCode constants	52
ULColumn class	53
ULColumnSchema class	58
ULConnection class	59
ULDatabaseManager class	64
ULDatabaseSchema class	69
ULIndexSchema class	71
ULPublicationSchema class	72
ULSQLCode constants	73
ULSQLType constants	76
ULStreamErrorCode constants	77
ULStreamErrorContext constants	80
ULStreamErrorID constants	81
ULStreamType	82
ULSyncMasks Type	83
ULSyncParms class	84
ULSyncResult class	85
ULSyncState constants	86
ULTable class	87
ULTableSchema class	96





# IULColumns collection

A collection of ULColumn objects.

## Properties

Prototype	Description
Count as long (read-only)	Returns the number of columns in the collection.
Item (Index) as ULColumn (read-only)	Returns a value from the collection. Index can be a number from 1 to count, or the name of a column.

## Example

You can enumerate all columns in eMbedded Visual Basic using the For Each statement:

```
Dim col As ULColumn
For Each col In table.Columns
    If col.IsNull Then
        MsgBox col.Schema.Name
        & "is null"
    End If
Next
```

# IULIndexSchemas collection

A collection of IULIndexSchema objects.

## Properties

Prototype	Description
Count as long (read-only)	Returns the number of indexes in the collection.
Item (Index) as ULIIndexSchema (read-only)	Returns an index from the collection. Items are indexed using 1-origin indexing. Index can be a number from 1 to count.

## Example

You can enumerate all the indexes on a table using the For Each statement:

```
Dim ix As ULIIndexSchema
For Each ix In TableSchema.Indexes
    'use ix
Next
```

# IULPublicationSchemas collection

A collection of IULPublicationSchema objects.

## Properties

Prototype	Description
Count as long (read-only)	Returns the number of indexes in the collection.
Item (Index) as ULIndexSchema (read-only)	Returns an index from the collection. Index can be a number from 1 to count.

## Example

You can enumerate all the publications using the For Each statement:


```
Dim ps As IULPublicationSchema
For Each ps In connection.schema.publications
    'use ps
Next
```

# ULAuthStatusCode constants

Constant	Value
ulAuthUnknown	0
ulAuthValid	1000
ulAuthValidButExpiresSoon	2000
ulAuthExpired	3000
ulAuthInvalid	4000
ulAuthInUse	5000

# ULColumn class

ULColumn allows you to get values from a table in a database. Each ULColumn object represents a particular column in a table.

 For information about the ULTable object, see "ULTable class" on page 87.

## Properties

Prototype	Description
IsNull as Boolean (read-only)	Indicates whether the column value is NULL. True if the column is NULL.
Schema as ULColumnSchema (read-only)	Returns the object representing the schema of the column.
Value as Variant	The data value of this column in the current row.

## AppendByteChunk method

<b>Prototype</b>	<b>AppendByteChunk</b> ( <i>byteArray</i> , [ <i>chunkSize</i> ] ) as Boolean Member of <b>Ultralite.ULColumn</b>
<b>Description</b>	Appends the buffer of bytes to the column if the type is ulTypeLongBinary.
<b>Parameters</b>	<p><b>chunkSize</b> The size of the data chunk expressed as a Long.</p> <p><b>byteArray</b> A variant. The data length, or the number of bytes to copy. If not provided, uses the size of the array.</p>
<b>Returns</b>	<p><b>True</b> if successful.</p> <p><b>False</b> if unsuccessful.</p> <p>The errors returned appear below.</p>

Error	Description
ulSQLE_INVALID_PARAMETER	A parameter is invalid, for example, if the data length is less than 0.
ulSQLE_CONVERSION_ERROR	If the column data type is not LONG BINARY or BINARY.

Example

```
Dim data (512) as Byte
...
table.Columns("edata").AppendByteChunk(data)
```

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

AppendStringChunk method

**Prototype**                      **AppendStringChunk( chunk )**  
Member of **Ultralite.ULColumn**

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

**Parameters**                      **chunk**    A string.  
The errors returned appear below.

Error	Description
ulSQL_CONVERSION_ERROR	If the column data type is not LONG STRING or STRING.

GetByteChunk method

**Prototype**                      **GetByteChunk( offset As Long, pByteArray, [ chunkSize ] ) As Long**  
Member of **Ultralite.ULColumn**

**Description**                      Fills the array with the binary data in the column. Suitable for BLOBs.

**Parameters**                      **offset**    The offset into the underlying array of bytes.  
**chunkSize**    An array of bytes expressed as Long type.  
**pByteArray**    A variant. Optional. Array data is passed by reference as array.

**Returns**                              The number of bytes read.

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 are true: <ul style="list-style-type: none"> <li>◆ offset is not 0 or 1</li> <li>◆ data length is greater than 64Kb</li> <li>◆ data length is less than 0</li> </ul>
ulSQLE_INVALID_PARAMETER	If the column data type is LONG BINARY and any of the following is true: <ul style="list-style-type: none"> <li>◆ offset is less than 1</li> <li>◆ data length is less than 0</li> </ul>

**Example**

```
Dim data (512) as Byte
...
table.GetColumn("edata").GetByteChunk(0,data)
```

In this example, *edata* is a column name.

**GetStringChunk method**

<b>Prototype</b>	<b>GetStringChunk( offset As Long, pStringObj, [ chunkSize ] ) As Long</b> Member of <b>Ultralite.ULColumn</b>
<b>Description</b>	Fills the string passed in (which should have space preallocated) with the binary data in the column. Suitable for long strings.
<b>Parameters</b>	<p><b>offset</b> The character offset into the underlying data from which we start getting the string.</p> <p><b>pStringObj</b> The string you want returned. This variant is passed by reference.</p> <p><b>chunkSize</b> The number of characters to retrieve.</p>
<b>Returns</b>	The number of characters copied. Room is left for a null termination character and the length does not include that character.

Example

```
Dim cd as ULColumn
Dim S as Strong
Dim l, off as Long
S=String(512, vbNulChar)
Off=0
Do
    L=col.GetStringChunk(offset, S, 512)
    If l=0 then Exit Do
    'use string ins
Loop
```

The errors returned appear below.

Error	Description
ulSQLE_CONVERSION_ERROR	If the column data type isn't ULTypeString or ULTypeLongString.
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 1 or string length is less than 0

SetByteChunk method

**Prototype**                      **SetByteChunk( ByteArray, [ length ] )** As Boolean  
Member of **Ultralite.ULColumn**

**Description**                      Sets the value of the column in the database to the array of bytes in the data field.

**Parameters**                      **ByteArray**    An array of bytes of type variant.

**length**    The length of the array.

**Returns**                      **True** if successful.

**False** if unsuccessful.

The errors returned appear below.

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.GetColumn("edata").SetByteChunk(data,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.

## SetToDefault method

**Prototype**

**SetToDefault( )**  
Member of **UltraLite.IULColumn**

**Description**

Sets the column to its default value as defined in the database schema. For example, an autoincrement column will be assigned the next available value, and incremented.

## 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. True if the column is autoincrement.
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. True if the column is global autoincrement.
ID as long (read-only)	Indicates the index number of the column in the range 1 to ULTableSchema.ColumnCount.
Name as String (read-only)	Returns the column name.
Nullable as Boolean (read-only)	Returns true if the column allows NULLs.
OptimalIndex as ULIndexSchema (read-only)	The best index to search this column. Or throws an error if no such index exists.
Precision as Long (read-only)	Returns the precision value for the column.
Scale as Long (read-only)	Returns the scale value for the column.
Size as Long (read-only)	Returns the column size for binary, numeric, and char data types.
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, and to synchronize.

## Example

Declare a connection in an eMbedded Visual Basic form with:

```
Private 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 - write-only	Sets the database ID value to be used for global autoincrement columns.
DatabaseManager as ULDatabaseManager (read-only)	Returns the owning database manager object.
DatabaseNew as Boolean (read-only)	Returns true if there is no database schema loaded. In this case, your application must load a new schema.
ErrorResume as Boolean	Sets the error handling property.
GlobalAutoIncrementUsage as Long (read-only)	Returns the percentage of available global autoincrement values that have been used.
LastErrorCode as ULSQLCodeConstants (read-only)	Returns the last error number.
LastErrorDescription As String (read-only)	Returns the last error description.

Prototype	Description
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 database.
Schema as ULDatabaseSchema (read-only)	Returns the ULDatabaseSchema object.
SyncParms As ULSyncParms (read-only)	Returns the synchronization parameters object.
SyncResult as ULSyncResult (read-only)	Returns the results of the most recent synchronization.

## CancelSynchronize method

Prototype	<b>CancelSynchronize( )</b> Member of <b>UltraLite.ULConnection</b>
Description	When called during synchronization, the method cancels the synchronization. The user can only call this method during one of the synchronization events.

## Close method

Prototype	<b>Close( )</b> Member of <b>UltraLite.ULConnection</b>
Description	Closes the connection to the database. No methods on the ULConnection object should be called after this method is called.

## Commit method

Prototype	<b>Commit( )</b> Member of <b>UltraLite.ULConnection</b>
Description	Commits outstanding changes to the database. This is only useful if AutoCommit is false.

## CountUploadRows method

<b>Prototype</b>	<b>CountUploadRows</b> ( [ <i>mask</i> as Long = 0 ], [ <i>threshold</i> as Long = -1 ] ) As Long Member of <b>UltraLite.ULConnection</b>
<b>Description</b>	Returns the number of rows that need to be uploaded when synchronization takes place.
<b>Parameters</b>	<b>mask</b> A unique identifier that refers to the publications to check. Use 0 for all publications. The default is 0.  <b>threshold</b> The maximum number of rows to count. Use -1 to indicate the maximum. The default is -1.

## GetNewUUID method

<b>Prototype</b>	<b>GetNewUUID</b> ( ) As String Member of <b>UltraLite.ULConnection</b>
<b>Description</b>	Returns a new universally unique identifier in a string format. This string is of the form xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx. Each call returns a new UUID.

## GetTable method

<b>Prototype</b>	<b>GetTable</b> ( <i>name</i> As String ) As ULTable Member of <b>UltraLite.ULConnection</b>
<b>Description</b>	Returns the <b>ULTable</b> object for the specified table. The table must be opened before data can be read from it.
<b>Parameters</b>	<b>name</b> The name of the table sought.
<b>Returns</b>	Returns the ULTable object.

## GrantConnectTo method

<b>Prototype</b>	<b>GrantConnectTo</b> ( <i>userid</i> as String, <i>password</i> as String ) Member of <b>UltraLite.ULConnection</b>
<b>Description</b>	Grants the specified user permission to connect to the database with the given password.
<b>Parameters</b>	<b>userid</b> 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 **UltraLite.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.

## RevokeConnectFrom method

**Prototype** **RevokeConnectFrom( userid as String )**  
Member of **UltraLite.ULConnection**

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

**Parameters** **userid** The user ID to be made unable to connect.

## Rollback method

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

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

## StartSynchronizationDelete method

**Prototype** **StartSynchronizationDelete( )**  
Member of **UltraLite.ULConnection**

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

## StopSynchronizationDelete method

**Prototype** **StopSynchronizationDelete( )**  
Member of **UltraLite.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 the information on the consolidated database.

## StringToUUID method

**Prototype** **StringToUUID( val )** as Variant  
Member of **UltraLite.ULConnection**

**Description** Converts a string in the form xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx to a variant array of 16 bytes.

**Parameters** **val** A string type holding a representation of a UUID value. You can obtain a new string UUID using GetNewUUID.

**See also** "GetNewUUID method" on page 61

## Synchronize method

**Prototype** **Synchronize( [ show-progress as Boolean ] )**  
Member of **UltraLite.ULConnection**

**Description** Synchronizes a consolidated database. This function does not return until synchronization is complete.

**Parameters** **show-progress** True or false. Set this to true to show the progress of synchronization as it happens. Default is false.

## UUIDToString method

**Prototype** **UUIDToString( val )** As String  
Member of **UltraLite.ULConnection**

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

**Parameters** **val** An array of 16 bytes. A variant type.

**See also** "GetNewUUID method" on page 61

# 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 are properties of ULDatabaseManager:

Prototype	Description
ErrorResume as Boolean	Set error handling property. Default is false. If set to true, a VB error will not be raised when DatabaseManager methods fail.
Version as String (read-only)	Returns the version string, in the form

## CreateDatabase method

The CreateDatabase method creates a new database and returns a connection to access its tables.

Prototype	<b>CreateDatabase</b> ( <i>parms</i> As String ) As ULConnection Member of <b>UltraLite.ULDatabaseManager</b>
Description	<p>The CreateDatabase function 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.</p> <p> For more information about ApplyFile, see "ApplyFile method" on page 70.</p>
Parameters	<p><b>parms</b> A semicolon-separated list of database creation parameters.</p> <p> For information on connection parameters, see "Connection Parameters" on page 25 of the book <i>UltraLite Foundations</i>.</p>
Examples	<p>The example below uses CreateObject to create and open a new database.</p> <pre>conn_parms = "uid=DBA;pwd=SQL"</pre>



```

'UID and PWD are set as default
open_parms = conn_parms & ";" & _
"file_name=\tutCustomer.udb"
schema_parms = open_parms & ";" & _
"schema_name=\tutCustomer.usm"

Set DatabaseMgr = _
CreateObject("UltraLite.ULDatabaseManager")
Set Connection = _
DatabaseMgr.CreateDatabase(schema_parms)


```

The example below shows how you can create a DatabaseManager with events. This tactic is used for showing synchronization progress.

```

Set DBMgr = CreateObjectWithEvents_
("UltraLite.ULDatabaseManager", "UL_")

```

 For information on OpenConnection, see "OpenConnection method" on page 68.

## DropDatabase method

The DropDatabase method deletes a database file.

### Prototype

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

### Description

The DropDatabase method deletes the database file. All information in the database file is lost. The data is not recoverable using UltraLite Components.

### Parameters

**parms** The filename for the database and schema files.

### Example

Note in the example below that you must use the same connection parameters when you drop the database that you use when you created the database.

```

conn_parms = "uid=DBA;pwd=SQL"

open_parms = conn_parms & ";" & _
"ce_file=\tutCustomer.udb"
DropDatabase(open_parms)

```

## OnReceive event

### Prototype

**OnReceive** ( *nBytes* As Long, *nInserts* As Long, *nUpdates* as Long, *nDeletes* as Long )  
Member of **UltraLite.ULDatabaseManager**

### 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.

**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**

```
Private Sub UL_OnReceive(ByVal nBytes As Long, _
    ByVal nInserts As Long, ByVal nUpdates As Long, _
    ByVal nDeletes As Long)

    prLine "OnReceive " & nBytes & " bytes, " & _
        nInserts & _
        " inserts, " & nUpdates & " updates, " & _
        nDeletes & " deletes"

End Sub
```

## OnSend event

**Prototype**

**OnSend**( *nBytes* As Long, *nInserts* As Long, *nUpdates* as Long, *nDeletes* as Long )  
Member of **UltraLite.ULDatabaseManager**

**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**

```
Private Sub Connection_OnSend(ByVal nBytes As Long, _  
    ByVal nInserts As Long, _  
    ByVal nUpdates As Long, _  
    ByVal nDeletes As Long)  
    send_count = send_count + nBytes  
    DisplaySyncStatus  
End Sub
```

## OnStateChange event

**Prototype**

**OnStateChange**( *newState* As ULSyncState, *oldState* As ULSyncState )  
Member of **UltraLite.ULDatabaseManager**

**Description**

This event is called whenever the state of the synchronization changes.

**Parameters**

**newState** The state that the synchronization process is about to enter.

**oldState** The state that the synchronization process just completed.

**Example**

```
Private Sub UL_OnStateChange(ByVal newState As _  
    Long, ByVal oldState As Long)  
  
    prLine "OnStateChange new:" & newState & _  
    ", old: " & oldState  
  
End Sub
```

## OnTableChange event

**Prototype**

**OnTableChange**( *newTableIndex* As Long, *numTables* As Long )  
Member of **UltraLite.ULDatabaseManager**

**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, and so it cannot be used with the DatabaseSchema.GetTableName method.

**numTables** The number of tables eligible to be synchronized.

**Example**

```
Private Sub UL_OnTableChange(ByVal newTableIndex _  
    As Long, ByVal numTables As Long)  
  
    prLine "OnTableChange index:" & newTableIndex & ",  
    #tables=" & numTables  
  
End Sub
```

## OpenConnection method

<b>Prototype</b>	<b>OpenConnection( <i>parms</i> As string )</b> As ULConnection Member of <b>UltraLite.ULDatabaseManager</b>
<b>Description</b>	<p>If a database exists, use this method to receive a connection. If a database does not exist, the call will fail.</p> <p>The function returns a <i>ULConnection</i> object which provides an open connection to a specified UltraLite database.</p> <p>Parameters are specified using a sequence of "name=value" pairs. If no user ID or password is given, the default is used.</p>
<b>Parameters</b>	<p><b>parms</b> The parameters that determine which database to connect to. The database filename is specified using the <i>parms</i> string. It should contain a value of the form file_name=UDBFILE or DBF=UDBFILE. See the <i>UltraLite Component Suite Foundations</i> book for more information on connecting to UltraLite databases.</p> <p>🔗 For information on connection parameters, see "Connection Parameters" on page 25 of the book <i>UltraLite Foundations</i>.</p>
<b>Returns</b>	The ULConnection object is returned if the connection was successful.
<b>Example</b>	<p>The example below shows how to use connection parameters in the OpenConnection method.</p> <pre>conn_parms = "uid=DBA;pwd=SQL" _ open_parms = conn_parms &amp; ";" &amp; "ce_file _ =\tutCustomer.udb" Set DatabaseMgr = CreateObject_ ("UltraLite.ULDatabaseManager") Set Connection = DatabaseMgr. _ OpenConnection(open_parms)</pre>

## Shutdown method

<b>Prototype</b>	<b>Shutdown( )</b> Member of <b>UltraLite.ULDatabaseManager</b>
<b>Description</b>	Shut down all open connections.

# 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 USM file.
DateOrder as String (read-only)	Controls the interpretation of date formats; valid values are MDY, YMD, or DMY.
NearestCentury as String (read-only)	Gets the nearest century database option. 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.
Precision as String (read-only)	Gets the database precision. Specifies the maximum number of digits in the result of any decimal arithmetic.
Publications as IULPublicationSchemas(read-only)	A collection of publication schema objects.
Signature as Variant (read-only)	Gets the database signature, an internal identifier representing the database schema.
TableCount as Long (read-only)	Returns the number of tables in the 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 ) As Boolean  
Member of **UltraLite.ULDatabaseSchema**

### Description

Applies a database schema update. Changes the schema of this database. This method is only useful on those occasions where you want to modify your existing database structure using DDL with, in most circumstances, no data loss. 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 schema file(s) containing the changes you wish to make.

### Returns

**True** if successful.

**False** if unsuccessful.

## 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 Long (read-only)	Returns the number of columns in the index.
ColumnName(position As Long) As String (read-only)	Column name in position of index.
ForeignKey as Boolean	Returns whether this is a foreign key. True if it is a foreign key.
IsColumnDescending( position As Long) As Boolean (read-only)	True if column in position of the index is sorted descending. False if ascending.
Name as String (read-only)	Returns the name of the index.
PrimaryKey as Boolean	True if the index is a primary key.
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. True if the index is unique.
UniqueKey as Boolean (read-only)	Indicates whether the index is a unique constraint on a table. True if the index is a primary key or has a unique constraint.

## 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.



## ULSQLCode constants

The ULSQLCodeConstants identify SQL error codes.

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_MULTUSER	-89
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_NOERROR	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
ulSQLE_TABLE_NOT_FOUND	-141
ulSQLE_TOO_MANY_CONNECTIONS	-102

uISQLE_UNABLE_TO_START_DATABASE	-82
uISQLE_UNCOMMITTED_TRANSACTIONS	-755
uISQLE_UNKNOWN_USERID	-140
uISQLE_UNSUPPORTED_CHARACTER_SET_ERROR	-869
uISQLE_UPLOAD_FAILED_AT_SERVER	-794

# ULSQLType constants

The ULSQLTypeConstants identify valid database column types.

Constant	Value
ulTypeBig	4
ulTypeBinary	13
ulTypeBit	7
ulTypeByte	6
ulTypeDate	9
ulTypeDateTime	8
ulTypeDouble	11
ulTypeLong	0
ulTypeLongBinary	14
ulTypeLongString	16
ulTypeNumeric	17
ulTypeReal	12
ulTypeShort	1
ulTypeString	15
ulTypeTime	10
ulTypeUnsignedBig	5
ulTypeUnsignedLong	2
ulTypeUnsignedShort	3

## ULStreamErrorCode constants

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

Constant	Value
<code>ULStreamErrorCodeNone</code>	0
<code>ULStreamErrorCodeParameter</code>	1
<code>ULStreamErrorCodeParameterNotUInt32</code>	2
<code>ULStreamErrorCodeParameterNotUInt32Range</code>	3
<code>ULStreamErrorCodeParameterNotBoolean</code>	4
<code>ULStreamErrorCodeParameterNotHex</code>	5
<code>ULStreamErrorCodeMemoryAllocation</code>	6
<code>ULStreamErrorCodeParse</code>	7
<code>ULStreamErrorCodeRead</code>	8
<code>ULStreamErrorCodeWrite</code>	9
<code>ULStreamErrorCodeEndWrite</code>	10
<code>ULStreamErrorCodeEndRead</code>	11
<code>ULStreamErrorCodeNotImplemented</code>	12
<code>ULStreamErrorCodeWouldBlock</code>	13
<code>ULStreamErrorCodeGenerateRandom</code>	14
<code>ULStreamErrorCodeInitRandom</code>	15
<code>ULStreamErrorCodeSeedRandom</code>	16
<code>ULStreamErrorCodeCreateRandomObject</code>	17
<code>ULStreamErrorCodeShuttingDown</code>	18
<code>ULStreamErrorCodeDequeuingConnection</code>	19
<code>ULStreamErrorCodeSecureCertificateRoot</code>	20
<code>ULStreamErrorCodeSecureCertificateCompanyName</code>	21
<code>ULStreamErrorCodeSecureCertificateChainLength</code>	22
<code>ULStreamErrorCodeSecureCertificateRef</code>	23
<code>ULStreamErrorCodeSecureCertificateNotTrusted</code>	24
<code>ULStreamErrorCodeSecureDuplicateContext</code>	25
<code>ULStreamErrorCodeSecureSetIo</code>	26

<b>Constant</b>	<b>Value</b>
UStreamErrorCodeSecureSetIoSemantics	27
UStreamErrorCodeSecureCertificateChainFunc	28
UStreamErrorCodeSecureCertificateChainRef	29
UStreamErrorCodeSecureEnableNonBlocking	30
UStreamErrorCodeSecureSetCipherSuites	31
UStreamErrorCodeSecureSetChainNumber	32
UStreamErrorCodeSecureCertificateFileNotFound	33
UStreamErrorCodeSecureReadCertificate	34
UStreamErrorCodeSecureReadPrivateKey	35
UStreamErrorCodeSecureSetPrivateKey	36
UStreamErrorCodeSecureCertificateExpiryDate	37
UStreamErrorCodeSecureExportCertificate	38
UStreamErrorCodeSecureAddCertificate	39
UStreamErrorCodeSecureTrustedCertificateFileNotFound	40
UStreamErrorCodeSecureTrustedCertificateRead	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
<code>ulStreamErrorCodeSocketGetHostByAddr</code>	58
<code>ulStreamErrorCodeSocketLocalhostNameNotFound</code>	59
<code>ulStreamErrorCodeSocketCreateTcpip</code>	60
<code>ulStreamErrorCodeSocketCreateUdp</code>	61
<code>ulStreamErrorCodeSocketBind</code>	62
<code>ulStreamErrorCodeSocketCleanup</code>	63
<code>ulStreamErrorCodeSocketClose</code>	64
<code>ulStreamErrorCodeSocketConnect</code>	65
<code>ulStreamErrorCodeSocketGetName</code>	66
<code>ulStreamErrorCodeSocketGetOption</code>	67
<code>ulStreamErrorCodeSocketSetOption</code>	68
<code>ulStreamErrorCodeSocketListen</code>	69
<code>ulStreamErrorCodeSocketShutdown</code>	70
<code>ulStreamErrorCodeSocketSelect</code>	71
<code>ulStreamErrorCodeSocketStartup</code>	72
<code>ulStreamErrorCodeSocketPortOutOfRange</code>	73
<code>ulStreamErrorCodeLoadNetworkLibrary</code>	74
<code>ulStreamErrorCodeActsycnNoPort</code>	75

# ULStreamErrorContext constants

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 constants

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

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

## ULStreamType

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

Constant	Value	Description
ulHTTP	1	HTTP stream
ulHTTPS	3	TCPIP stream
ulTCPIP	2	TCP/IP stream

## ULSyncMasks Type

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

Constant	Value	Description
<code>ulSyncAllPublications</code>	1	A mask to include all publications
<code>ulSyncAllTables</code>	0	A mask to include all tables in the database

## 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.
PublicationMask as Long	The publications to synchronize - the default is all publications.
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 ULSyncTypeConstants	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 submit during synchronization.
Version as String	The synchronization script version to run.

## ULSyncResult class

The attributes set for the ULSyncResult 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 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 ULSyncStreamErrorCode (read-only)	The error code reported by the stream itself.
StreamErrorContext as ULSyncStreamErrorContext (read-only)	The basic network operation being performed.
StreamErrorID as ULSyncStreamErrorID (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 constants

Constant	Value
ulSyncStateStarting	0
ulSyncStateConnecting	1
ulSyncStateSendingHeader	2
ulSyncStateSendingTable	3
ulSyncStateSendingData	4
ulSyncStateFinishingUpload	5
ulSyncStateReceivingUploadAck	6
ulSyncStateReceivingTable	7
ulSyncStateReceivingData	8
ulSyncStateCommittingDownload	9
ulSyncStateSendingDownloadAck	10
ulSyncStateDisconnecting	11
ulSyncStateDone	12
ulSyncStateError	13
ulSyncStateCancelled	99

## ULTable class

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

Before you can work with table data, you must call the Open method. ULTable uses table modes for table operations:

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

## Properties

Prototype	Description
BOF as Boolean (read-only)	Returns whether you are currently positioned before the first row.
Columns as IULColumns (read-only)	Returns a collection of column objects
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 **UltraLite.ULTable**

### Description

Close the table cursor. Once closed, table data can no longer be read.

## Columns method

**Columns( *name* As String )** As ULColumn  
Member of **UltraLite.ULTable**

**Description** Returns the **IULColumns** object for the specified column.

🔗 For information about the **IULColumns** object, see "ULColumn class" on page 53.

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

## Delete method

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

**Description** Deletes the current row from the table.

## DeleteAllRows method

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

**Description** Prepares a table for a find.


## FindFirst method

**Prototype** **FindFirst( [ *num\_columns* As Long = 32767 ] )** As Boolean  
Member of **UltraLite.ULTable**



<b>Description</b>	<p>Move forwards through the table from the beginning, looking for a row that exactly matches a value or set of values in the current index.</p> <p>The current index is that used to specify the sort order of the table. It is specified when your application calls the <b>Open</b> method. The default index is the primary key.</p> <p>To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure the cursor position is <b>AfterLast()</b>.</p> <p><i>Note:</i> FindBegin must be called before using this method.</p>
<b>Parameters</b>	<p><b>num_columns</b> An optional parameter referring to the number of columns in the index that should be checked.</p>
<b>Returns</b>	<p><b>True</b> if successful.</p> <p><b>False</b> if unsuccessful.</p>

## FindLast method

<b>Prototype</b>	<b>FindLast</b> ( [ <i>num_columns</i> As Long = 32767 ] ) As Boolean Member of <b>UltraLite.ULTable</b>
<b>Description</b>	<p>Move backwards through the table from the end, looking for a row that matches a value or set of values in the current index.</p> <p>The current index is used to specify the sort order of the table. It is specified when your application calls the <b>Open</b> method. The default index is the primary key.</p> <p> For more information, see "Open method" on page 94.</p> <p>To specify the value to search for, set the column value for each column in the index for which you want to find the value. The cursor is left on the last row found that exactly matches the index value. On failure the cursor position is <b>BeforeFirst()</b>.</p> <p><i>Note:</i> FindBegin must be called before using this method.</p>
<b>Parameters</b>	<p><b>num_columns</b> An optional parameter referring to the number of columns in the index that should be checked.</p>
<b>Returns</b>	<p><b>True</b> if successful.</p> <p><b>False</b> if unsuccessful.</p>

## FindNext method

<b>Prototype</b>	<b>FindNext</b> ( [ <i>num_columns</i> As Long = 32767 ] ) As Boolean Member of <b>UltraLite.ULTable</b>
<b>Description</b>	<p>Move forwards through the table from the current position, looking for the next row that exactly matches a value or set of values in the current index.</p> <p>The current index is that used to specify the sort order of the table, It is specified when your application calls the <b>Open</b> method. The default index is the primary key.</p> <p>🔗 For more information, see "Open method" on page 94.</p> <p>To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure, the cursor position is <b>AfterLast()</b>.</p> <p><i>Note:</i> Must be preceded by FindFirst or FindLast.</p>
<b>Parameters</b>	<b>num_columns</b> An optional parameter referring to the number of columns to be used in the comparison.
<b>Returns</b>	<p><b>True</b> if successful.</p> <p><b>False</b> if unsuccessful and record is then EOF.</p>

## FindPrevious method

<b>Prototype</b>	<b>FindPrevious</b> ( [ <i>num_columns</i> As Long = 32767 ] ) As Boolean Member of <b>UltraLite.ULTable</b>
<b>Description</b>	<p>Move backwards through the table from the current position, looking for the previous row that exactly matches a value or set of values in the current index.</p> <p>The current index is that used to specify the sort order of the table. It is specified when your application calls the <b>OpenByIndex</b> method. The default index is the primary key.</p> <p>🔗 For more information, see "Open method" on page 94.</p> <p>On failure the cursor position is <b>BeforeFirst()</b>.</p>
<b>Parameters</b>	<b>num_columns</b> An optional parameter referring to the number of columns to be used in the comparison.
<b>Returns</b>	<p><b>False</b> if unsuccessful and record is then BOF.</p> <p><b>True</b> if successful.</p>


## Insert method

<b>Prototype</b>	<b>Insert( )</b> As Boolean Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Inserts a row in the table with values specified in previous <b>Set</b> methods. Must be preceded by <b>InsertBegin</b> .

## InsertBegin method

<b>Prototype</b>	<b>InsertBegin( )</b> Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Prepares a table for inserting a new row, setting column values to their defaults.
<b>Example</b>	<pre> CustomerTable.InsertBegin     CustomerTable.Columns("Fname").Value = fname     CustomerTable.Columns("Lname").Value = lname     If Len(city) &gt; 0 Then         CustomerTable.Columns("City").Value = city     End If      If Len(phone) &gt; 0 Then         CustomerTable.Columns("phone").Value = phone     End If CustomerTable.Insert </pre>

## LookupBackward method


<b>Prototype</b>	<b>LookupBackward( [ num_columns As Long = 32767 ] )</b> As Boolean Member of <b>UltraLite.ULTable</b>
<b>Description</b>	<p>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.</p> <p>The current index is that used to specify the sort order of the table, It is specified when your application calls the <b>OpenByIndex</b> method. The default index is the primary key.</p> <p> For more information, see "Open method" on page 94.</p> <p>To specify the value to search for, set the column value for each column in the index. The cursor is left on the 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 <b>BeforeFirst()</b>.</p>

<b>Parameters</b>	<b>num_columns</b> An optional parameter referring to the number of columns to be used in the comparison.
<b>Returns</b>	<b>True</b> if successful. <b>False</b> if unsuccessful.

## LookupBegin method

<b>Prototype</b>	<b>LookupBegin( )</b> Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Prepares a table for a lookup.

## LookupForward method

<b>Prototype</b>	<b>LookupForward( [ num_columns As Long = 32767 ] ) As Boolean</b> Member of <b>UltraLite.ULTable</b>
<b>Description</b>	<p>Move forward through the table starting from the beginning, looking for the first row that matches or is greater than a value or set of values in the current index.</p> <p>The current index is that used to specify the sort order of the table. It is specified when your application calls the <b>OpenByIndex</b> method. The default index is the primary key.</p> <p> For more information, see "Open method" on page 94.</p> <p>To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure (that is, if no rows are greater than the value being looked for), the cursor position is <b>AfterLast()</b>.</p>
<b>Parameters</b>	<b>num_columns</b> An optional parameter referring to the number of columns to be used in the comparison.
<b>Returns</b>	<b>True</b> if successful. <b>False</b> if unsuccessful.

## MoveAfterLast method

<b>Prototype</b>	<b>MoveAfterLast ( ) As Boolean</b> Member of <b>UltraLite.ULTable</b>
------------------	---

<b>Description</b>	Moves to a position after the last row.
<b>Returns</b>	<b>True</b> if successful. <b>False</b> if there is no data in the table.

## MoveBeforeFirst method

<b>Prototype</b>	<b>MoveBeforeFirst ( )</b> Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Moves to a position before the first row.

## MoveFirst method

<b>Prototype</b>	<b>MoveFirst ( )</b> As Boolean Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Moves to the first row.
<b>Returns</b>	<b>True</b> if successful. <b>False</b> if there is no data in the table.

## MoveLast method

<b>Prototype</b>	<b>MoveLast ( )</b> As Boolean Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Moves to the last row.
<b>Returns</b>	<b>True</b> if successful. <b>False</b> if there is no data in the table.

## MoveNext method

<b>Prototype</b>	<b>MoveNext ( )</b> As Boolean Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Moves to the next row.
<b>Returns</b>	<b>True</b> if successful. <b>False</b> if there is no more data in the table.

## MovePrevious method

<b>Prototype</b>	<b>MovePrevious ( )</b> As Boolean Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Moves to the previous row.
<b>Returns</b>	<b>True</b> if successful. <b>False</b> if there is no more data in the table.


## MoveRelative method

<b>Prototype</b>	<b>MoveRelative ( index As Long )</b> As Boolean Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Moves a certain number of rows relative to the current row.
<b>Parameters</b>	<b>index</b> The number of rows to move.
<b>Returns</b>	<b>True</b> if successful. <b>False</b> if the move failed.

## Open method

<b>Prototype</b>	<b>Open( [ index_name As String ] )</b> Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Opens the table so it can be read or manipulated. When you use the Open method, the rows are ordered by the named index. If no index is provided, rows are ordered by the primary key. The cursor is positioned before the first row in the table.
<b>Parameters</b>	<b>index_name</b> The name of the index.

## Truncate method

<b>Prototype</b>	<b>Truncate ( )</b> Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Removes all data from this table but does not affect the data in the consolidated database. Deletions are not sent up in the next synchronization, so truncate will not affect the data in the consolidated database.   For more information, see "StopSynchronizationDelete method" on page 62.

## Update method

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

## UpdateBegin method

<b>Prototype</b>	<b>UpdateBegin( )</b> Member of <b>UltraLite.ULTable</b>
<b>Description</b>	Prepares a table for modifying the contents of the current row.
<b>Example</b>	<pre>Table.UpdateBegin Table.Columns ( "ColName" ).Value="New Value" Table.Update</pre>

# 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.
Indexes As IULIndexSchemas	The collection of indexes on this table.
Name as String (read-only)	This table's name.
NeverSynchronized As Boolean (read-only)	True if this table is to be excluded from all synchronizations.
PrimaryKey as ULIndexSchema (read-only)	The primary key for this table.
UploadUnchangedRows as Boolean (read-only)	True if all rows in the table should be uploaded on synchronization, rather than just the rows changed since the last synchronization.

## InPublication method

Prototype	<b>InPublication( <i>pub_name</i> As String )</b> As Boolean Member of <b>UltraLite.ULTableSchema</b>
Description	Indicates whether this table is part of the specified publication.
Returns	<b>True</b> if the table is part of the publication. <b>False</b> if the table is not part of the publication.



# Index

## A

- Accessing and manipulating data
  - about, 35
  - UltraLite for eMbedded Visual Basic, 35
- Accessing schema information
  - about, 41
  - UltraLite for eMbedded Visual Basic, 41
- AppendByteChunk method (ULColumn class)
  - UltraLite for eMbedded Visual Basic API, 53
- AppendStringChunk method (ULColumn class)
  - UltraLite for eMbedded Visual Basic API, 54
- ApplyFile (ULDatabaseManager class)
  - UltraLite for eMbedded Visual Basic API, 70
- Architecture
  - UltraLite for eMbedded Visual Basic, 4
- AuthStatus property (ULSyncResult class)
  - UltraLite for eMbedded Visual Basic API, 85
- AutoCommit mode
  - about, 39
- AutoCommit property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 59
- AutoIncrement property (ULColumnSchema class)
  - UltraLite for eMbedded Visual Basic API, 58

## B

- BOF property (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 87

## C

- CancelSynchronize method (ULConnection class)
  - UltraLite for eMbedded Visual Basic API, 60
- casting
  - data types, 36
- CheckpointStore property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 84
- Close method (ULConnection class)
  - UltraLite for eMbedded Visual Basic API, 60
- Close method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 87
- ColumnCount property (ULIndexSchema class)
  - UltraLite for eMbedded Visual Basic API, 71
- ColumnCount property (ULTableSchema class)
  - UltraLite for eMbedded Visual Basic API, 96
- ColumnName property (ULIndexSchema class)
  - UltraLite for eMbedded Visual Basic API, 71
- columns
  - accessing schema information, 41
- Columns collection
  - introduction, 35
- Columns method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 88
- Commit method
  - about, 39
- Commit method (ULConnection class)
  - UltraLite for eMbedded Visual Basic API, 60
- commits
  - about, 39

- connecting
  - UltraLite databases, 32
- Connecting to the UltraLite database
  - about, 32
  - UltraLite for eMbedded Visual Basic, 32
- connection parameters
  - databases, 32
- Count property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 49
- Count property (IULIndexSchemas collection)
  - UltraLite for eMbedded Visual Basic API, 50
- Count property (IULPublicationSchemas collection)
  - UltraLite for eMbedded Visual Basic API, 51
- CountUploadRows method (ULConnection class)
  - UltraLite for eMbedded Visual Basic API, 61
- CreateDatabase method (ULDatabaseManager class)
  - UltraLite for eMbedded Visual Basic API, 64
- CustDB sample
  - UltraLite for eMbedded Visual Basic, 26

## D

- data manipulation
  - about, 35, 39
- data types
  - accessing, 36
  - casting, 36
- database schema
  - accessing, 41
- DatabaseID property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 59
- DatabaseManager property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 59
- DatabaseNew property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 59
- databases
  - accessing schema information, 41
  - connecting to, 32
  - schema, 30
  - working with, 30

- DateFormat property (ULDatabaseSchema class)
  - UltraLite for eMbedded Visual Basic API, 69
- DateOrder property (ULDatabaseSchema class)
  - UltraLite for eMbedded Visual Basic API, 69
- default values
  - setting, 57
- DefaultValue property (ULColumnSchema class)
  - UltraLite for eMbedded Visual Basic API, 58
- Delete method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 88
- DeleteAllRows method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 88
- deleting rows
  - about, 37
- Development platforms
  - supported, 3
  - UltraLite for eMbedded Visual Basic, 3
- DownloadOnly property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 84
- DropDatabase method (ULDatabaseManager class)
  - UltraLite for eMbedded Visual Basic API, 65

## E

- eMbedded Visual Basic
  - Development platforms, 3
  - supported versions, 3
- EOF property (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 87
- error handling
  - about, 42
- Error handling
  - about, 42
  - UltraLite for eMbedded Visual Basic, 42
- ErrorResume property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 59
- ErrorResume property (ULDatabaseManager class)
  - UltraLite for eMbedded Visual Basic API, 64
- errors
  - handling, 42

## F

- features
  - UltraLite for eMbedded Visual Basic, 2
- feedback
  - documentation, vii
  - providing, vii
- find methods
  - about, 36
- find mode
  - about, 39
- FindBegin method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 88
- FindFirst method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 88
- FindLast method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 89
- FindNext method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 90
- FindPrevious method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 90
- ForeignKey property (ULIndexSchema class)
  - UltraLite for eMbedded Visual Basic API, 71

## G

- GetByteChunk method (ULColumn class)
  - UltraLite for eMbedded Visual Basic API, 54
- GetStringChunk method (ULColumn class)
  - UltraLite for eMbedded Visual Basic API, 55
- GetTable function (ULConnection class)
  - UltraLite for eMbedded Visual Basic API, 61
- GlobalAutoIncrement property (ULColumnSchema class)
  - UltraLite for eMbedded Visual Basic API, 58
- GlobalAutoIncrementUsage property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 59
- grantConnectTo method
  - introduction, 43

- GrantConnectTo method (ULConnection class)
  - UltraLite for eMbedded Visual Basic API, 61

## I

- ID property (ULColumnSchema class)
  - UltraLite for eMbedded Visual Basic API, 58
- IgnoredRows property (ULSyncResult class)
  - UltraLite for eMbedded Visual Basic API, 85
- indexes
  - accessing schema information, 41
- Indexes property (ULTableSchema class)
  - UltraLite for eMbedded Visual Basic API, 96
- InPublication method (ULTableSchema class)
  - UltraLite for eMbedded Visual Basic API, 96
- Insert method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 91
- insert mode
  - about, 39
- InsertBegin method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 91
- inserting rows
  - about, 37
- internals
  - data manipulation, 39
- IsNull property (ULColumn class)
  - UltraLite for eMbedded Visual Basic API, 53
- IsOpen property (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 87
- Item property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 49
- Item property (IULIndexSchemas collection)
  - UltraLite for eMbedded Visual Basic API, 50
- Item property (IULPublicationSchemas collection)
  - UltraLite for eMbedded Visual Basic API, 51
- IULColumns collection
  - about, 49
  - properties, 49
  - UltraLite for eMbedded Visual Basic API, 49

IULIndexSchemas collection  
     about, 50  
     properties, 50  
     UltraLite for eMbedded Visual Basic API, 50

IULPublicationSchemas class  
     properties, 51

IULPublicationSchemas collection  
     about, 51  
     UltraLite for eMbedded Visual Basic PI, 51

**L**

LastDownloadTime method (ULConnection class)  
     UltraLite for eMbedded Visual Basic API, 62

LastErrorCode property (IULColumns collection)  
     UltraLite for eMbedded Visual Basic API, 59

LastErrorDescription property (IULColumns collection)  
     UltraLite for eMbedded Visual Basic API, 59

LastIdentity property (IULColumns collection)  
     UltraLite for eMbedded Visual Basic API, 59

lookup methods  
     about, 36

lookup mode  
     about, 39

LookupBackward method (ULTable class)  
     UltraLite for eMbedded Visual Basic API, 91

LookupBegin method (ULTable class)  
     UltraLite for eMbedded Visual Basic API, 92

LookupForward method (ULTable class)  
     UltraLite for eMbedded Visual Basic API, 92

## M

masks  
     publications, 83

Microsoft Visual Basic  
     supported versions, 3

modes  
     about, 39

MoveAfterLast method (ULTable class)  
     UltraLite for eMbedded Visual Basic API, 92

MoveBeforeFirst method (ULTable class)  
     UltraLite for eMbedded Visual Basic API, 93

MoveFirst method  
     introduction, 35

MoveFirst method (ULTable class)  
     UltraLite for eMbedded Visual Basic API, 93

MoveLast method (ULTable class)  
     UltraLite for eMbedded Visual Basic API, 93

MoveNext method  
     introduction, 35

MoveNext method (ULTable class)  
     UltraLite for eMbedded Visual Basic API, 93

MovePrevious method (ULTable class)  
     UltraLite for eMbedded Visual Basic API, 94

MoveRelative method (ULTable class)  
     UltraLite for eMbedded Visual Basic API, 94

## N

Name property (ULColumnSchema class)  
     UltraLite for eMbedded Visual Basic API, 58

Name property (ULIndexSchema class)  
     UltraLite for eMbedded Visual Basic API, 71

Name property (ULTableSchema class)  
     UltraLite for eMbedded Visual Basic API, 96

NearestCentury property (ULDatabaseSchema class)  
     UltraLite for eMbedded Visual Basic API, 69

NeverSynchronized property (ULTableSchema class)  
     UltraLite for eMbedded Visual Basic API, 96

NewPassword property (IULColumns collection)  
     UltraLite for eMbedded Visual Basic API, 84

newsgroups  
     technical support, vii

Nullable property (ULColumnSchema class)  
     UltraLite for eMbedded Visual Basic API, 58

## O

- object hierarchy
  - UltraLite for eMbedded Visual Basic, 4
- OnReceive event (ULDatabaseManager class)
  - UltraLite for eMbedded Visual Basic API, 65
- OnSend event (ULDatabaseManager class)
  - UltraLite for eMbedded Visual Basic API, 66
- OnStateChange event (ULDatabaseManager class)
  - UltraLite for eMbedded Visual Basic API, 67
- OnTableChange event (ULDatabaseManager class)
  - UltraLite for eMbedded Visual Basic API, 67
- Open method
  - ULTable object, 35
- Open method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 94
- OpenConnection method (ULDatabaseManager class)
  - UltraLite for eMbedded Visual Basic API, 68
- OpenParms property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 59
- OptimalIndex property (ULColumnSchema class)
  - UltraLite for eMbedded Visual Basic API, 58

## P

- Password property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 84
- passwords
  - authentication, 43
- platforms
  - supported, 3
- Precision property (ULColumnSchema class)
  - UltraLite for eMbedded Visual Basic API, 58
- Precision property (ULDatabaseSchema class)
  - UltraLite for eMbedded Visual Basic API, 69
- Preparing to work with eMbedded Visual Basic
  - about, 28
  - UltraLite for eMbedded Visual Basic, 28
- PrimaryKey property (ULIndexSchema class)
  - UltraLite for eMbedded Visual Basic API, 71

- PrimaryKey property (ULTableSchema class)
  - UltraLite for eMbedded Visual Basic API, 96
- projects
  - creating UltraLite for eMbedded Visual Basic projects, 11
- publication masks
  - about, 83
  - all, 83
- PublicationMask property (IULColumns collection)
  - UltraLite for eMbedded Visual Basic API, 84
- publications
  - accessing schema information, 41
- Publications property (ULDatabaseSchema class)
  - UltraLite for eMbedded Visual Basic API, 69

## R

- ReferencedIndexName property (ULIndexSchema class)
  - UltraLite for eMbedded Visual Basic API, 71
- ReferencedTableName property (ULIndexSchema class)
  - UltraLite for eMbedded Visual Basic API, 71
- RevokeConnectFrom method (ULConnection class)
  - UltraLite for eMbedded Visual Basic API, 62
- revokeConnectionFrom method
  - introduction, 43
- Rollback method
  - about, 39
- Rollback method (ULConnection class)
  - UltraLite for eMbedded Visual Basic API, 62
- rollbacks
  - about, 39
- RowCount property (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 87
- rows
  - accessing current row, 36

**S**

## samples

- UltraLite for eMbedded Visual Basic, 26

## Scale property (ULColumnSchema class)

- UltraLite for eMbedded Visual Basic API, 58

## schema

- accessing, 41

## schema files

- about, 30
- creating, 31

## Schema painter

- starting, 31

## Schema property (IULColumns collection)

- UltraLite for eMbedded Visual Basic API, 59

## Schema property (ULColumn class)

- UltraLite for eMbedded Visual Basic API, 53

## Schema property (ULTable class)

- UltraLite for eMbedded Visual Basic API, 87

## sColumnDescendinproperty (ULIndexSchema class)

- UltraLite for eMbedded Visual Basic API, 71

## scrolling

- through rows, 35

## searching

- rows, 36

## SendColumnNames property (IULColumns collection)

- UltraLite for eMbedded Visual Basic API, 84

## SendDownloadAck property (IULColumns collection)

- UltraLite for eMbedded Visual Basic API, 84

## SetByteChunk method (ULColumn class)

- UltraLite for eMbedded Visual Basic API, 56

## SetToDefault method (ULColumn class)

- UltraLite for eMbedded Visual Basic API, 57

## Shutdown method (ULDatabaseManager class)

- UltraLite for eMbedded Visual Basic API, 68

## Signature property (ULDatabaseSchema class)

- UltraLite for eMbedded Visual Basic API, 69

## Size property (ULColumnSchema class)

- UltraLite for eMbedded Visual Basic API, 58

## SQL Anywhere Studio

- additional features, 3

## SQLType property (ULColumnSchema class)

- UltraLite for eMbedded Visual Basic API, 58

## StartSynchronizationDelete method (ULConnection class)

- UltraLite for eMbedded Visual Basic API, 62

## StopSynchronizationDelete method (ULConnection class)

- UltraLite for eMbedded Visual Basic API, 62

## Stream property (IULColumns collection)

- UltraLite for eMbedded Visual Basic API, 84

## StreamErrorCode property (ULSyncResult class)

- UltraLite for eMbedded Visual Basic API, 85

## StreamErrorContext property (ULSyncResult class)

- UltraLite for eMbedded Visual Basic API, 85

## StreamErrorID property (ULSyncResult class)

- UltraLite for eMbedded Visual Basic API, 85

## StreamErrorSystem property (ULSyncResult class)

- UltraLite for eMbedded Visual Basic API, 85

## StreamParms property (IULColumns collection)

- UltraLite for eMbedded Visual Basic API, 84

## StringToUUID method (ULConnection class)

- UltraLite for eMbedded Visual Basic API, 63

## support

- newsgroups, vii

## supported platforms, 3

## synchronization

- HTTP, 44

- introduction, 44

- SQL Anywhere Studio required, 3

- TCP/IP, 44

## Synchronize method (ULConnection class)

- UltraLite for eMbedded Visual Basic API, 63

## Synchronizing UltraLite applications

- about, 44

- UltraLite for eMbedded Visual Basic, 44

## SyncParms property (IULColumns collection)

- UltraLite for eMbedded Visual Basic API, 59

## SyncResult property (IULColumns collection)

- UltraLite for eMbedded Visual Basic API, 59

## T

- TableCount property (ULDatabaseSchema class)
  - UltraLite for eMbedded Visual Basic API, 69
- tables
  - accessing schema information, 41
- target platforms
  - supported, 3
  - UltraLite for eMbedded Visual Basic, 3
- technical support
  - newsgroups, vii
- TimeFormat property (ULDatabaseSchema class)
  - UltraLite for eMbedded Visual Basic API, 69
- TimestampFormat property (ULDatabaseSchema class)
  - UltraLite for eMbedded Visual Basic API, 69
- transaction processing
  - about, 39
- transactions
  - about, 39
- Truncate method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 94
- tutorial
  - UltraLite for eMbedded Visual Basic, 7

## U

- udb files
  - UltraLite databases, 30
- ULAuthStatusCode constants
  - about, 52
  - UltraLite for eMbedded Visual Basic API, 52
- ULColumn class
  - about, 53
  - properties, 53
  - UltraLite for eMbedded Visual Basic API, 53
- ULColumnSchema class
  - about, 58
  - properties, 58
  - UltraLite for eMbedded Visual Basic API, 58
- ULColumnSchema object
  - introduction, 41
- ULConnection class
  - about, 59
  - properties, 59
  - UltraLite for eMbedded Visual Basic API, 59
- ULConnection object
  - introduction, 32
- ULDatabaseManager class
  - about, 64
  - UltraLite for eMbedded Visual Basic API, 64
- ULDatabaseManager object
  - introduction, 32
- ULDatabaseSchema class
  - about, 69
  - properties, 64, 69
  - UltraLite for eMbedded Visual Basic API, 69
- ULDatabaseSchema object
  - introduction, 41
- ULIndexSchema class
  - about, 71
  - properties, 71
  - UltraLite for eMbedded Visual Basic API, 71
- ULIndexSchema object
  - introduction, 41
- ULPublicationSchema class
  - about, 72
  - properties, 72
  - UltraLite for eMbedded Visual Basic API, 72
- ULPublicationSchema object
  - introduction, 41
- ULSQLCode constants
  - about, 73
  - UltraLite for eMbedded Visual Basic API, 73
- ULSQLType constants
  - about, 76
  - UltraLite for eMbedded Visual Basic API, 76
- ULStreamErrorCode constants
  - about, 77
  - UltraLite for eMbedded Visual Basic API, 77
- ULStreamErrorContext constants
  - about, 80
  - UltraLite for eMbedded Visual Basic PI, 80

- ULStreamErrorID constants
  - about, 81
  - UltraLite for eMbedded Visual Basic API, 81
- ULStreamType
  - about, 82
  - UltraLite for eMbedded Visual Basic API, 82
- ULSyncParms class
  - about, 84
  - properties, 84
  - UltraLite for eMbedded Visual Basic API, 84
- ULSyncResult class
  - about, 85
  - properties, 85
  - UltraLite for eMbedded Visual Basic API, 85
- ULSyncState constants
  - about, 86
  - UltraLite for eMbedded Visual Basic API, 86
- ULTable class
  - about, 87
  - properties, 87
  - UltraLite for eMbedded Visual Basic API, 87
- ULTable object
  - introduction, 35
- ULTableSchema class
  - about, 96
  - properties, 96
  - UltraLite for eMbedded Visual Basic API, 96
- ULTableSchema object
  - introduction, 41
- UltraLite
  - about, 1
- UltraLite databases
  - about, 30
  - features, 30
  - schema, 30
- UltraLite for eMbedded Visual Basic
  - Accessing and manipulating data, 35
  - Accessing schema information, 41
  - architecture, 4
  - Connecting to the UltraLite database, 32
  - Error handling, 42
  - features, 2
  - object hierarchy, 4
  - Preparing to work with eMbedded Visual Basic, 28
  - Synchronizing UltraLite applications, 44
  - UltraLite for eMbedded Visual Basic
    - architecture, 4
    - UltraLite for eMbedded Visual Basic features, 2
    - User authentication, 43
- UltraLite for eMbedded Visual Basic API
  - IULColumns class, 53
  - IULColumns collection, 49
  - IULIndexSchemas collection, 50
  - IULPublicationSchemas collection, 51
  - ULAuthStatusCode constants, 52
  - ULColumnSchema class, 58
  - ULConnection class, 59
  - ULDatabaseManager class, 64
  - ULDatabaseSchema class, 69
  - ULIndexSchema class, 71
  - ULPublicationSchema class, 72
  - ULSQLCode constants, 73
  - ULSQLType constants, 76
  - ULStreamErrorCodeConstants, 77
  - ULStreamErrorContext constants, 80
  - ULStreamErrorID constants, 81
  - ULStreamType, 82
  - ULSyncParms class, 84
  - ULSyncResult class, 85
  - ULSyncState constants, 86
  - ULTable class, 87
  - ULTableSchema class, 96
- UltraLite for eMbedded Visual Basic architecture
  - about, 4
  - UltraLite for eMbeddd Visual Basic, 4
- UltraLite for eMbedded Visual Basic features
  - about, 2
  - UltraLite for eMbedded Visual Basic, 2
- UltraLite for eMbedded Visual Basic projects
  - creating, 11
- UniqueIndex property (ULIndexSchema class)
  - UltraLite for eMbedded Visual Basic API, 71
- UniqueKey property (ULIndexSchema class)
  - UltraLite for eMbedded Visual Basic API, 71
- Update method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 95
- update mode
  - about, 39
- UpdateBegin method (ULTable class)
  - UltraLite for eMbedded Visual Basic API, 95



updating rows  
about, 37

UploadOK property (ULSyncResult class)  
UltraLite for eMbedded Visual Basic API, 85

UploadOnly property (IULColumns collection)  
UltraLite for eMbedded Visual Basic API, 84

UploadUnchangedRows property (ULTableSchema  
class)  
UltraLite for eMbedded Visual Basic API, 96

user authentication  
about, 43

User authentication  
about, 43  
UltraLite for eMbedded Visual Basic, 43

UserName property (IULColumns collection)  
UltraLite for eMbedded Visual Basic API, 84

users  
authentication, 43

usm files  
about, 30  
creating, 31

UUIDToString method (ULConnection class)  
UltraLite for eMbedded Visual Basic API, 63

## V

Value property (ULColumn class)  
UltraLite for eMbedded Visual Basic API, 53

values  
accessing, 36

Version property (IULColumns collection)  
UltraLite for eMbedded Visual Basic API, 84

Version property (ULDatabaseManager class)  
UltraLite for eMbedded Visual Basic API, 64

Visual Basic  
supported versions, 3

## W

Windows CE  
supported versions, 3

