iAnywhere®
A SYBASE COMPANY

# UltraLite™ ActiveX User's Guide

Part number: DC36293-01-0902-01
Last modified: October 2004

# Contents

# About This Manual

| | |
|---|---|
| Subject | This manual describes UltraLite ActiveX. With UltraLite ActiveX 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 and JScript application developers who want 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 or JScript is assumed. |

## SQL Anywhere Studio documentation

This book is part of the SQL Anywhere documentation set. This section describes the books in the documentation set and how you can use them.

**The SQL Anywhere Studio documentation**

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

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

- **SQL Anywhere Studio Security Guide** This book provides information about security features in Adaptive Server Anywhere databases. Adaptive Server Anywhere 7.0 was awarded a TCSEC (Trusted Computer System Evaluation Criteria) C2 security rating from the U.S. Government. This book may be of interest to those who wish to run the current version of Adaptive Server Anywhere in a manner equivalent to the C2-certified environment.
- **MobiLink Administration Guide** This book describes how to use the MobiLink data synchronization system for mobile computing, which enables sharing of data between a single Oracle, Sybase, Microsoft or IBM database and many Adaptive Server Anywhere or UltraLite databases.
- **MobiLink Clients** This book describes how to set up and synchronize Adaptive Server Anywhere and UltraLite remote databases.
- **MobiLink Server-Initiated Synchronization User's Guide** This book describes MobiLink server-initiated synchronization, a feature of MobiLink that allows you to initiate synchronization from the consolidated database.
- **MobiLink Tutorials** This book provides several tutorials that walk you through how to set up and run MobiLink applications.
- **QAnywhere User's Guide** This manual describes MobiLink QAnywhere, a messaging platform that enables the development and deployment of messaging applications for mobile and wireless clients, as well as traditional desktop and laptop clients.
- **iAnywhere Solutions ODBC Drivers** This book describes how to set up ODBC drivers to access consolidated databases other than Adaptive Server Anywhere from the MobiLink synchronization server and from Adaptive Server Anywhere remote data access.
- **SQL Remote User's Guide** This book describes all aspects of the SQL Remote data replication system for mobile computing, which enables sharing of data between a single Adaptive Server Anywhere or Adaptive Server Enterprise database and many Adaptive Server Anywhere databases using an indirect link such as e-mail or file transfer.
- **SQL Anywhere Studio Help** This book includes the context-sensitive help for Sybase Central, Interactive SQL, and other graphical tools. It is not included in the printed documentation set.
- **UltraLite Database User's Guide** This book is intended for all UltraLite developers. It introduces the UltraLite database system and provides information common to all UltraLite programming interfaces.

♦ **UltraLite Interface Guides** A separate book is provided for each UltraLite programming interface. Some of these interfaces are provided as UltraLite components for rapid application development, and others are provided as static interfaces for C, C++, and Java development.

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

Documentation formats

SQL Anywhere Studio provides documentation in the following formats:

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

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

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

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

  The PDF books are accessible from the online books, or from the Windows Start menu.

♦ **Printed books** The complete set of books is available from Sybase sales or from eShop, the Sybase online store, at *http://eshop.sybase.com/eshop/documentation*.

# Documentation conventions

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

## Syntax conventions

The following conventions are used in the SQL syntax descriptions:

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

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

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

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

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

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

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

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

  **RELEASE SAVEPOINT** [ *savepoint-name* ]

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

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

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

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

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

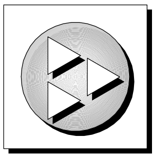
  [ **QUOTES** { **ON** | **OFF** } ]

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

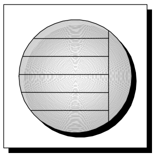
**Graphic icons**

The following icons are used in this documentation.

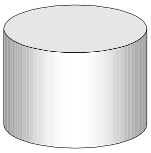
- A client application.



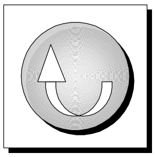
- A database server, such as Sybase Adaptive Server Anywhere.



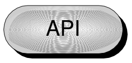
- A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



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



- A programming interface.



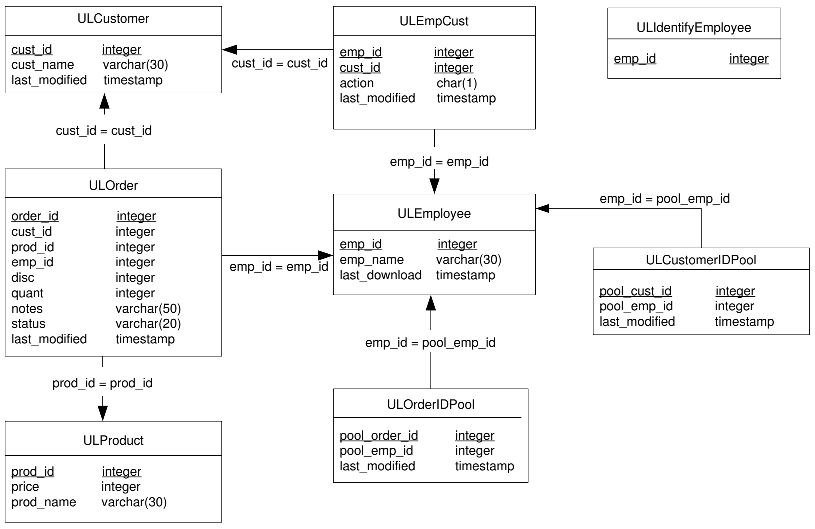
# The CustDB sample database

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

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

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

The following diagram shows the tables in the CustDB database and how they are related to each other.



## Finding out more and providing feedback

Finding out more

Additional information and resources, including a code exchange, are available at the iAnywhere Developer Network at *http://www.ianywhere.com/developer/*.

If you have questions or need help, you can post messages to the iAnywhere Solutions newsgroups listed below.

When you write to one of these newsgroups, always provide detailed information about your problem, including the build number of your version of SQL Anywhere Studio. You can find this information by typing **dbeng9 -v** at a command prompt.

The newsgroups are located on the *forums.sybase.com* news server. The newsgroups include the following:

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

**Newsgroup disclaimer**

iAnywhere Solutions has no obligation to provide solutions, information or ideas on its newsgroups, nor is iAnywhere Solutions obliged to provide anything other than a systems operator to monitor the service and ensure its operation and availability.

iAnywhere Solutions Technical Advisors as well as other staff assist on the newsgroup service when they have time available. They offer their help on a volunteer basis and may not be available on a regular basis to provide solutions and information. Their ability to help is based on their workload.

Feedback

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

You can e-mail comments and suggestions to the SQL Anywhere documentation team at iasdoc@ianywhere.com. Although we do not reply to e-mails sent to that address, we read all suggestions with interest.

In addition, you can provide feedback on the documentation and the software through the newsgroups listed above.

CHAPTER 1

# Introduction to UltraLite ActiveX

About this chapter

This chapter introduces UltraLite ActiveX. It assumes that you are familiar with the features of UltraLite, as described in “Welcome to UltraLite” [*UltraLite Database User’s Guide*, page 3].

Contents

# UltraLite ActiveX features

UltraLite ActiveX is a relational data management system for mobile devices. It has the performance, resource efficiency, robustness, and security required by business applications. UltraLite also provides synchronization with enterprise data stores.

## System requirements and supported platforms

Development platforms

To develop applications using UltraLite ActiveX, you require the following:

- ♦ One of the following:
  - eMbedded Visual Basic 3.0, for development using Visual Basic.
  - Pocket Internet Explorer, for development using JScript.

☞ For more information, see "UltraLite development platforms" [*Introducing SQL Anywhere Studio,* page 99].

Target platforms

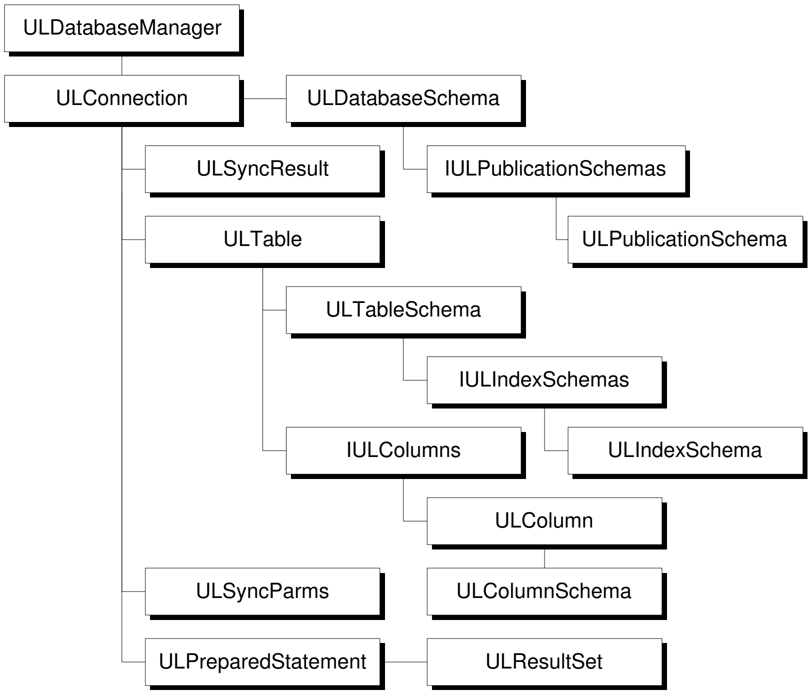
UltraLite ActiveX supports the following target platforms:

- ♦ Windows CE 3.0 and higher, with Pocket PC on the ARM and MIPS processors.

☞ For more information, see "UltraLite target platforms" [*Introducing SQL Anywhere Studio,* page 109].

## UltraLite ActiveX architecture

The UltraLite programming interface exposes a set of objects for data manipulation using an UltraLite database. The following figure describes the object hierarchy.



The following list describes some of the more commonly-used high level objects.

- **ULDatabaseManager** manages connections to UltraLite databases.

  ☞ For more information, see "ULDatabaseManager class" on page 101.

- **ULConnectionParms** holds a set of connection parameters.

  ☞ For more information, see "ULConnectionParms class" on page 98.

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

  ☞ For more information, see "ULConnection class" on page 91.

- **ULPreparedStatement, ULResultSet, and ULResultSetSchema** manage database requests and their results using SQL.

☞ For more information, see “ULPreparedStatement class” on page 113, “ULResultSet class” on page 117, and “ULResultSetSchema class” on page 122.

- **ULTable and ULColumn** manage data using a table-based API.

  ☞ For more information, see “ULTable class” on page 142 and “ULColumn class” on page 85.

- **ULSyncParms and ULSyncResult** manage synchronization through the MobiLink synchronization server.

  ☞ For more information about synchronization with MobiLink, see “UltraLite Clients” [*MobiLink Clients,* page 277].

CHAPTER 2

# Understanding UltraLite ActiveX Development

About this chapter

This chapter explains how to develop applications using UltraLite ActiveX.

☞ For a hands-on tutorial, see "Tutorial: A Sample UltraLite ActiveX Application" on page 43 or "Tutorial: An UltraLite ActiveX Application for Pocket IE" on page 63

Contents

# Preparing to use UltraLite ActiveX

The following procedures describe the steps you must take before you can build an application using UltraLite ActiveX.

## Adding UltraLite ActiveX to the eMbedded Visual Basic design environment

To access the UltraLite ActiveX control from your eMbedded Visual Basic project, you must add UltraLite ActiveX to the design environment.

The following procedure is only required for eMbedded Visual Basic applications.

❖ **To add a reference to UltraLite ActiveX**

1. From the eMbedded Visual Basic menu, choose Project ➤ References.
2. If iAnywhere Solutions, ActiveX for UltraLite 9.0 is included in the list of available references, select it and click OK.

   If iAnywhere Solutions, ActiveX for UltraLite 9.0 does not appear in the list of available references:

   ♦ Add the control to the list of available references.

     Browse to the *UltraLite\UltraLiteActiveX\win32* subdirectory of your SQL Anywhere 9.0 installation. Open *uldo9.dll.*

   ♦ Select iAnywhere Solutions, ActiveX for UltraLite 9.0 and click OK.

## Adding UltraLite ActiveX to a Windows CE device

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

The following procedure applies to both JScript and eMbedded Visual Basic development. If you do not have eMbedded Visual Basic installed on your computer, you can copy the appropriate DLL to the *\Windows* directory on your device and register it using *regsvrce.exe*. If you do not have *regsvrce.exe* on your device, you can copy it to your device from the Microsoft Windows CE SDK.

### ❖ To add the UltraLite ActiveX control to Windows CE

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. Choose Control ➤ Add New Control.
5. Browse to one of the following platform-specific files, located in subdirectories of your SQL Anywhere installation:
   - **ARM** *UltraLite\UltraLiteActiveX\ce\arm\uldo9.dll*
   - **MIPS** *UltraLite\UltraLiteActiveX\ce\mips\uldo9.dll*
   - **Emulator** *UltraLite\UltraLiteActiveX\ce\emulator30\uldo9.dll*
   - **Intel 386** *UltraLite\UltraLiteActiveX\ce\386\uldo9.dll*
6. Click Open.

## Deploying a schema file to a Windows CE device

A schema file is used in the initial creation of an UltraLite database to specify the structure of the database.

The following procedure deploys an UltraLite schema file to a Windows CE device.

☞ For information about creating an UltraLite schema file, see "Creating UltraLite database schema files" on page 9.

### ❖ To deploy a schema file to Windows CE

1. From the eMbedded Visual Basic menu, choose Tools ➤ Remote Tools ➤ File Viewer.
2. If your device does not appear in the left pane, connect to the device:
   - Choose Connection ➤ Add Connection.
   - Select your device from the list and click OK to establish a connection.
3. Copy the schema file to the device
   - Select a destination directory on the device.
   - Choose File ➤ Export File.
   - Locate the schema (*.usm*) file.

- Click OK to export the file to the device.

## Working with JScript

The following sections describe how to set up your HTML page in order to optimize the performance of UltraLite ActiveX.

### Cached pages

To prevent Pocket IE from using locally cached pages and to ensure all contact is dynamically recalculated, the page expiry should be set to 0. This will cause Pocket IE to always reload the page. Use the following as the first line of your HTML document.

```
<META HTTP-EQUIV="Expires" CONTENT="0">
```

### Script execution order

To force scripts to run before the page is loaded, for example, to get a database connection which results in dynamic HTML content, place those scripts in the HTML head:

```
<head> <script>... </script></head>
```

### Pocket Internet Explorer limitations

The JScript language supported by Pocket IE is incompatible with that of Internet Explorer 4 and up, in the following ways:

- Pocket IE JScript is case insensitive. Scripts written for Pocket IE may not work on IE unless the proper case is used.
- The BUTTON tag is not supported, but a button can be created using <INPUT TYPE="BUTTON">.
- HTML tag names are in the same namespace as JScript functions. Ensure that the name of an HTML element does not conflict with any function names in the current document.

  For example, pressing this button will cause an error:

  ```
  <SCRIPT>
  function b_Done() { ... }
  </SCRIPT>
  <INPUT NAME="b_Done" TYPE="BUTTON" VALUE="Done" onClick="b_
           Done()">
  ```

- Dynamic HTML is not supported by Pocket IE. To get dynamic content, you can emit HTML with **document.write**, or by setting DIV contents.

## Working with the database schema

The schema is the structure of a database. It is the collection of table definitions, index definitions, and publication definitions within the database, and all the relationships between them.

You create UltraLite databases by creating an UltraLite database schema file and apply that file to a database by calling a function in your application.

For information on creating UltraLite database schema files, see

### Creating UltraLite database schema files

You can create an UltraLite schema file using the UltraLite Schema Painter or the *ulinit* utility.

- **UltraLite 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 ➤ SQL Anywhere 9 ➤ UltraLite ➤ UltraLite Schema Painter, or double-click a schema (*.usm*) file in Windows Explorer.

  ☞ For more information about using the UltraLite Schema Painter, see "Lesson 1: Create an UltraLite database schema" [*UltraLite Database User's Guide,* page 130].

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

  ☞ For more information about using the *ulinit* utility, see "The ulinit utility" [*UltraLite Database User's Guide,* page 112].

### Changing the schema of a database

To change the schema of an existing database, create a schema file with the new schema and apply this schema to the existing database. In most cases there will be no data loss, but data loss can occur if columns are deleted or if the data type for a column is changed to an incompatible type.

☞ For more information about these methods, see "ApplyFile method" on page 110 and "ApplyFileWithParms method" on page 110.

☞ For information about preparing a new schema file for deployment, see "Upgrading UltraLite database schemas" [*UltraLite Database User's Guide,* page 54].

Example

The following code applies a new schema file.

```
’ eMbedded Visual Basic
Dim db As ULDatabaseManager
Dim conn As ULConnection
Set db = CreateObject("UltraLite.ULDatabaseManager")
Set conn = db.OpenConnection("dbf = \My Documents\mydb.udb")
conn.Schema.ApplyFile("\My Documents\myschema.usm")
```

```
// JScript
var db;
var conn;
db = new ActiveXObject("UltraLite.ULDatabaseManager");
conn = db.OpenConnection("dbf = \\My Documents\\mydb.udb");
conn.Schema.ApplyFile("\\My Documents\\myschema.usm");
```

## Connecting to an UltraLite database

UltraLite applications must connect to a database before carrying out operations on the data in it. This section describes how to connect to an UltraLite database.

**Using the ULConnection object**

The following properties of the ULConnection object govern global application behavior.

☞ For more information about the ULConnection object, see "ULConnection class" on page 91.

- **Commit behavior** By default, UltraLite applications are in AutoCommit mode. Each insert, update, or delete statement is committed to the database immediately. Set ULConnection.AutoCommit to false to build transactions into your application. Turning AutoCommit off and performing commits directly can improve the performance of your application.

  ☞ For more information, see "Commit method" on page 93.

- **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 "Authenticating users" on page 38.

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

  ☞ For more information, see "Synchronizing data" on page 39.

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

  ☞ For more information, see "GetTable method" on page 94.

**Connecting to a database**

You can connect to a database using either a ULConnectionParms object or a connection string. Methods that use a ULConnectionParms object allow you to manipulate connection parameters with ease and accuracy. Methods that use a connection string require that you successfully create a connections string.

The following procedure uses a ULConnectionParms object to connect to an UltraLite database.

☞ For more information about connecting to an UltraLite database using a ULConnectionParms object, see "CreateDatabaseWithParms method" on page 102 and "OpenConnectionWithParms method" on page 107.

### ❖ To connect to an UltraLite database using ULConnectionParms

1. Create a ULDatabaseManager 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 as global to the application or as a class-level variable.

   ```
   ’ eMbedded Visual Basic
   Dim DatabaseMgr As ULDatabaseManager
   Set DatabaseMgr =
            CreateObject("UltraLite.ULDatabaseManager")
   ```

   ```
   // JScript
   var DatabaseMgr;
   DatabaseMgr = new ActiveXObject(
            "UltraLite.ULDatabaseManager" );
   ```

2. Declare a ULConnection object.

   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 as global to the application.

   ```
   ’ eMbedded Visual Basic
   Dim Connection As ULConnection
   ```

   ```
   // JScript
   var Connection;
   ```

3. Create a ULConnectionParms object.

   ```
     ’ eMbedded Visual Basic
     Dim LoginParms As ULConnectionParms
     Set LoginParms = CreateObject("UltraLite.ULConnectionParms")
   ```

   ```
     // JScript
     var LoginParms;
     LoginParms = new
              ActiveXObject("UltraLite.ULConnectionParms");
   ```

4. Set the required properties of the ULConnectionParms object.

   For example, the following code specifies the location of the database on the Windows CE device as *\tutorial\tutorial.udb.*

   ```
     ’ eMbedded Visual Basic
     LoginParms.DatabaseOnCE = "\tutorial\tutorial.udb"
   ```

   ```
     // JScript
     LoginParms.DatabaseOnCE = "\\tutorial\\tutorial.udb";
   ```

   Using the following properties, you must specify a schema file for CreateDatabaseWithParms or a database file for

OpenConnectionWithParms. For information about additional properties, see "Properties" on page 98.

| Keyword | Description |
|---|---|
| DatabaseOnCE | The path and filename of the UltraLite database on Windows CE. |
| DatabaseOnDesktop | The path and filename of the UltraLite database on the desktop machine |
| SchemaOnCE | The path and filename of the UltraLite schema on Windows CE. |
| SchemaOnDesktop | The path and filename of the UltraLite schema on the desktop machine. |

5. Open a connection to the database.

   CreateDatabaseWithParms and OpenConnectionWithParms return an open connection as a ULConnection object. Each method takes a single ULConnectionParms object as its argument.

   The following code attempts to connect to an existing database. If the database does not exist, the OpenConnectionWithParms method returns an error. This causes CreateDatabaseWithParms to create a database using the specified schema file.

   In Crossfire, be sure to include the GetOcx method on the ULConnectionParms object.

```
' eMbedded Visual Basic
On Error Resume Next
Set Connection = DatabaseMgr.OpenConnectionWithParms(
         LoginParms )
if Err.Number <> ULSQLCode.ulSQLE_NOERROR Then
  Set Connection = DatabaseMgr.CreateDatabaseWithParms(
         LoginParms )
End If
```

```
// JScript
DatabaseMgr.ErrorResume = true;
Connection = DatabaseMgr.OpenConnectionWithParms( LoginParms
         );
if ( DatabaseMgr.LastErrorCode != ULSQLCode.ulSQLE_NOERROR )
         {
  Connection = DatabaseMgr.CreateDatabaseWithParms(
         LoginParms );
}
```

## Using frames to maintain application state (JScript)

In Pocket IE, UltraLite ActiveX is accessed using JScript embedded on an HTML page.

In the following code, a ULDatabaseManager object is created. When a page containing this script is loaded, a new ULDatabaseManager object is created. When the browser moves to another page, that object is discarded. If a database connection was obtained on one page, it is lost when that page is unloaded.

```
<SCRIPT LANGUAGE="JScript">
var DatabaseMgr;
DatabaseMgr = new ActiveXObject( "UltraLite.ULDatabaseManager"
         );
</SCRIPT>
```

Losing application state, including database connections, whenever a form changes, is slow and expensive. To avoid this, Pocket IE applications can use HTML frames to maintain application state. The database connection can be kept by the frameset document. Pocket IE requires at least two frames within a frameset, and draws a three-pixel border between the frames.

**Note**
Support for frames is not included in versions of Pocket IE prior to 1.1.

The following code defines two frames.

```
// JScript
<frameset rows="5%,*" BORDER=0>
<frame SRC="topline.htm" NAME="TopLine" MARGINWIDTH=0
         MARGINHEIGHT=0>
<frame SRC="connect.htm" NAME="Connect" MARGINWIDTH=0
         MARGINHEIGHT=0>
</frameset>
<SCRIPT LANGUAGE="JScript">
var Connection;      // UltraLite Connection
var DatabaseMgr;  // UltraLite Database Manager
</SCRIPT>
```

In the CustDB example, the minimal frame defined in *topline.htm* is a placeholder that displays the application name at the top of the screen. The rest of the screen is used to display the other pages of the application. The HTML and JScript in *connect.htm* provide the initial form for the application. Since all these forms will be swapped into the same frameset, they can access their parent frameset objects using **window.topwindow**.

The following code creates a new variable, conn, and assigns the connection created in the top frame to it. If you are in **top**, referencing the connection as **top.Connection** is optional.

```
// JScript
<SCRIPT>
function usingConnection() {
  if ( top.Connection == null ) {
    return;     // not yet connected..
  }
  var conn = top.Connection;
  ...
}
</SCRIPT>
```

To swap forms in and out of the frameset, a JScript function can use the **document.location.replace** method. The following code closes the current connection and returns to the connect form.

```
// JScript
<SCRIPT>
function exitApp() {
  if ( top.Connection != null ) {
    top.Connection.Close();
  }
  document.location.replace("connect.htm");
}
</SCRIPT>
<INPUT NAME="b_Done" TYPE="BUTTON" VALUE="Done"
         onClick="exitApp()">
```

## Encryption and obfuscation

You can encrypt or obfuscate your UltraLite database using UltraLite ActiveX.

Encryption

To create a database with encryption, set the ULConnectionParms.EncryptionKey property. When you call CreateDatabaseWithParms and pass in the ConnectionParms object, the database created and encrypted with the specified key.

☞ For more information about the EncryptionKey property, see "Encryption Key connection parameter " [*UltraLite Database User's Guide,* page 75] and "ChangeEncryptionKey method" on page 92.

Example

You can change the encryption key by specifying the new encryption key on the Connection object. In this example, "apricot" is the encryption key.

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

After the database is encrypted, connections to the database must specify the correct encryption key. Otherwise, the connection fails.

Obfuscation

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

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

Example

The following code obfuscates a new database.

```
’ eMbedded Visual Basic
open_parms = "ce_file=\tutorial.udb;ce_schema=\
         tutorial.usm;obfuscate=1"
Set Connection = DatabaseMgr.CreateDatabase( open_parms )
// JScript
open_parms = "ce_file=\\tutorial.udb;ce_schema=\\
         tutorial.usm;obfuscate=1";
Connection = DatabaseMgr.CreateDatabase( open_parms );
```

## Working with data using dynamic SQL

UltraLite applications can access table data using dynamic SQL or the Table API. This section describes data access using dynamic SQL.

☞ For information about the Table API, see "Working with data using the table API" on page 25.

This section explains how to perform the following tasks using dynamic SQL.

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

☞ This section does not describe the SQL language itself. For information about dynamic SQL features, see "Dynamic SQL" [*UltraLite Database User's Guide*, page 159].

☞ The sequence of operations required is similar for any SQL operation. For an overview, see "Using dynamic SQL" [*UltraLite Database User's Guide*, page 161].

### Data manipulation: INSERT, UPDATE and DELETE

With UltraLite, you can perform SQL Data Manipulation Language operations. These operations are performed using the ExecuteStatement method, a member of the ULPreparedStatement class.

☞ For more information the ULPreparedStatement class, see "ULPreparedStatement class" on page 113.

**Using parameters in your prepared statements**
Placeholders for parameters are supplied using the ? character. For any INSERT, UPDATE or DELETE, each ? is referenced according to its ordinal position in the prepared statement. For example, the first ? is referred to as 1, and the second as 2.

## ❖ To INSERT a row

1. Declare a ULPreparedStatement object.

   ```
   ' eMbedded Visual Basic
   Dim PrepStmt As ULPreparedStatement
   // JScript
   var PrepStmt;
   ```

2. Assign an INSERT statement to your prepared statement object. In the following code, TableName and ColumnName are the names of a table and column.

   ```
   ' eMbedded Visual Basic
   Set PrepStmt = Connection.PrepareStatement( _
       "INSERT into TableName(ColumnName) VALUES (?)")
   // JScript
   PrepStmt = Connection.PrepareStatement(
       "INSERT into TableName(ColumnName) VALUES (?)");
   ```

3. Assign parameter values for the statement.

   ```
   ' eMbedded Visual Basic
   Dim NewValue As String
   NewValue = "Bob"
   PrepStmt.SetStringParameter 1, NewValue
   // JScript
   var NewValue;
   NewValue = "Bob";
   PS.SetStringParameter(1, NewValue);
   ```

4. Execute the statement.

   ```
   ' eMbedded Visual Basic
   PrepStmt.ExecuteStatement
   // JScript
   PrepStmt.ExecuteStatement();
   ```

## ❖ To UPDATE a row

1. Declare a ULPreparedStatement object.

   ```
   ' eMbedded Visual Basic
   Dim PrepStmt As ULPreparedStatement
   // JScript
   var PrepStmt;
   ```

2. Assign an UPDATE statement to your prepared statement object. In the following code, TableName and ColumnName are the names of a table and column.

```
' eMbedded Visual Basic
Set PrepStmt = Connection.PrepareStatement( _
    "UPDATE TableName SET ColumnName = ? WHERE ID = ?")
```

```
// JScript
PrepStmt = Connection.PrepareStatement(
    "UPDATE TableName SET ColumnName = ? WHERE ID = ?");
```

3. Assign parameter values for the statement.

```
' eMbedded Visual Basic
Dim NewValue As String
NewValue = "Bob"
PrepStmt.SetParameter 1, NewValue
PrepStmt.SetParameter 2, "6"
```

```
// JScript
var NewValue;
NewValue = "Bob";
PrepStmt.SetParameter(1, NewValue);
PrepStmt.SetParameter(2, "6");
```

4. Execute the statement

```
' eMbedded Visual Basic
PrepStmt.ExecuteStatement
```

```
// JScript
PrepStmt.ExecuteStatement();
```

## ❖ To DELETE a row

1. Declare a ULPreparedStatement object.

```
' eMbedded Visual Basic
Dim PrepStmt As ULPreparedStatement
```

```
// JScript
var PrepStmt;
```

2. Assign a DELETE statement to your prepared statement object.

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

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

3. Assign parameter values for the statement.

```
' eMbedded Visual Basic
Dim IDValue As String
IDValue = "6"
PrepStmt.SetParameter 1, IDValue
```

```
// JScript
var IDValue;
IDValue = "6";
PrepStmt.SetParameter( 1, IDValue );
```

4. Execute the statement.

```
'eMbedded Visual Basic
PrepStmt.ExecuteStatement
```

```
// JScript
PrepStmt.ExecuteStatement();
```

## Data retrieval: SELECT

When you execute a SELECT statement, the ULPreparedStatement.ExecuteQuery method returns a ULResultSet object.

The ULResultSet class contains methods for navigating within a result set. The values are then accessed using the ULResultSet Value property.

☞ For more information about ULResultSet objects, see "ULResultSet class" on page 117.

Example

In the following code, the results of a SELECT query are accessed through a ULResultSet. When first assigned, the ULResultSet is positioned before the first row. The ULResultSet.MoveFirst method is then called to navigate to the first record in the result set.

☞ For more information about navigating a result set, see "Navigation with dynamic SQL" on page 22.

```
' eMbedded Visual Basic
Dim MyResultSet As ULResultSet
Dim PrepStmt As ULPreparedStatement
Set PrepStmt = Connection.PrepareStatement( _
    "SELECT ID, Name FROM customer")
Set MyResultSet = PrepStmt.ExecuteQuery
MyResultSet.MoveFirst
```

```
// JScript
var MyResultSet;
var PrepStmt;
PrepStmt = Connection.PrepareStatement( _
    "SELECT ID, Name FROM customer");
MyResultSet = PrepStmt.ExecuteQuery();
MyResultSet.MoveFirst();
```

Example

The following code demonstrates how to use the Value property to obtain the column values for the current row. It uses the Value property access both Integer and String values. UltraLite ActiveX uses a variant data type to achieve this flexibility.

The Value property uses the following syntax, where *Index* is the ordinal position of the column name in your SELECT statement.

```
MyResultSetName.Value(Index)
```

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

```
' eMbedded Visual Basic
If MyResultSet.RowCount = 0 Then
  lblID.Caption = ""
  txtName.Text = ""
Else
  lblID.Caption = MyResultSet.Value(1)
  txtName.Text = MyResultSet.Value(2)
  MyResultSet.MoveRelative(0)
End If
```

```
// JScript
If ( MyResultSet.RowCount == 0 ) {
  lblID.Caption = "";
  txtName.Text = "";
} Else {
  lblID.Caption = MyResultSet.Value(1);
  lblID.Text = MyResultSet.Value(2);
  MyResultSet.MoveRelative(0);
}
```

The following procedure uses a SELECT statement to retrieve information from the database. The results of the query are assigned to a ULResultSet object.

## ❖ To perform a SELECT statement

1. Declare a ULPreparedStatement object.

   ```
   ' eMbedded Visual Basic
   Dim PrepStmt As ULPreparedStatement
   ```

   ```
   // JScript
   var PrepStmt;
   ```

2. Assign a prepared statement to your ULPreparedStatement object. In the following code, TableName and ColumnName are the names of a table and column.

   ```
   ' eMbedded Visual Basic
   Set PrepStmt = Connection.PrepareStatement( _
       "SELECT ColumnName FROM TableName")
   ```

   ```
   // JScript
   PrepStmt = Connection.PrepareStatement(
       "SELECT ColumnName FROM TableName")
   ```

3. Execute the query.

   In the eMbedded Visual Basic code below, a listbox captures the result of the SELECT query.

```
  ’ eMbedded Visual Basic
  Dim MyResultSet As ULResultSet
  Set MyResultSet = PrepStmt.ExecuteQuery
  While MyResultSet.MoveNext
    listbox.AddItem y.Value(1)
  Wend
```

   In the JScript code below, an string captures the result of the SELECT query as an HTML table.

```
// JScript
var MyResultSet;
var resultTable;
MyResultSet = PrepStmt.ExecuteQuery();
var ncols = MyResultSet.Schema.ColumnCount;
var fld, line, nrows;
resultTable = "<html><table cellpadding=0
         cellspacing=0><tr>";
resultTable = resultTable + "<th>" + "ColumnName" + "</th>";
resultTable = resultTable + "</tr><tr>";
nrows = 0;
while ( MyResultSet.MoveNext() ) {
  line = "<tr>";
    nrows++;
  if ( MyResultSet.IsNull(1) ) {
        fld = "(null)";
      } else {
        fld = MyResultSet.Value(1);
      }
      line = line + "<td>" + fld + "</td>";
    resultTable = resultTable + line + "</tr>";
}
resultTable = resultTable + _
    "</table><B>#Rows=" + nrows + "</B></html>";
```

## Navigation with dynamic SQL

UltraLite ActiveX provides you with a number of methods to navigate a result set in order to perform a wide range of navigation tasks.

The following methods of the ULResultSet object allow you to navigate your result set:

- **MoveAfterLast** moves to a position after the last row.
- **MoveBeforeFirst** moves to a position before the first row.
- **MoveFirst** moves to the first row.

- **MoveLast** moves to the last row.
- **MoveNext** moves to the next row.
- **MovePrevious** moves to the previous row.
- **MoveRelative** moves a certain number of rows relative to the current row. Positive index values move forward in the result set, negative index values move backward in the result set, and zero does not move the cursor. Zero is useful if you want to repopulate a row buffer.

Example

The following code demonstrates how to use the MoveFirst method to navigate within a result set.

```
' eMbedded Visual Basic
Set PrepStmt = Connection.PrepareStatement( _
    "SELECT ID, Name FROM customer")
Set MyResultSet = PrepStmt.ExecuteQuery
MyResultSet.MoveFirst
```

```
// JScript
PrepStmt = Connection.PrepareStatement(
    "SELECT ID, Name FROM customer");
MyResultSet = PrepStmt.ExecuteQuery();
MyResultSet.MoveFirst();
```

The same technique is used for all of the Move methods.

☞ For more information about these navigational methods, see "ULResultSet class" on page 117.

## ULResultSet schema property

The ULResultSet.Schema property allows you to retrieve information about the columns in the query. The properties of this ULResultSetSchema object include ColumnName, ColumnCount, ColumnPrecision, ColumnScale, ColumnSize, and ColumnSQLType.

Example

The following example demonstrates how to use ULResultSet.Schema to capture schema information.

```
' eMbedded Visual Basic
Dim i As Integer
Dim MySchema as ULResultSetSchema
Set MySchema = MyResultSet.Schema
For i = 1 To MySchema.ColumnCount
  cn = MySchema.ColumnName(i)
  ct = MySchema.ColumnSQLType(i)
  MsgBox cn, ct
Next i
```

```
// JScript
var i;
var MySchema;
For ( i = 1; i <= MySchema.ColumnCount; i++) {
  cn = MySchema.ColumnName(i);
  ct = MySchema.ColumnSQLType(i);
  alert ( cn + " " + ct );
}
```

## Working with data using the table API

UltraLite applications can access table data using dynamic SQL or the Table API. This section describes data access using the Table API.

☞ For information about dynamic SQL, see "Working with data using dynamic SQL" on page 17.

This section explains how to perform the following tasks using the Table API.

- 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.

### Navigation with the Table API

UltraLite ActiveX provides you with a number of methods to navigate a table in order to perform a wide range of navigation tasks.

The following methods of the ULTable object allow you to navigate your result set:

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

Example

The following code opens the customer table and scrolls through its rows. It then displays a message box or alert with the last name of each customer.

```
' eMbedded Visual Basic
Dim tCustomer as ULTable
Set tCustomer = Connection.GetTable( "customer" )
tCustomer.Open
' the third column contains the last name of the customer
Set colLastName = tCustomer.Columns.Item(3)
tCustomer.MoveBeforeFirst
While tCustomer.MoveNext
    MsgBox colLastName.Value
Wend
```

```
// JScript
var tCustomer;
tCustomer = Connection.GetTable( "customer" );
tCustomer.Open();
// the third column contains the last name of the customer
colLastName = tCustomer.Columns.Item(3);
tCustomer.MoveBeforeFirst();
While (tCustomer.MoveNext()) {
  alert( colLastName.Value );
}
```

**The Columns collection**

The columns of a table are contained in a Columns collection. You can address columns by index number according to the order in which they were created in the schema file or by name.

☞ For more information, see "IULColumns collection" on page 81.

**Example**

The following code accesses the LastName column.

```
' eMbedded Visual Basic
Set colLastName = tCustomer.Columns.( LastName )
```

```
// JScript
colLastName = tCustomer.Columns.( LastName );
```

**Specifying an index**

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.

**Example**

The following code moves to the first row of the customer table as ordered by the ix_name index.

```
' eMbedded Visual Basic
Set tCustomer = Connection.GetTable("customer")
tCustomer.Open "ix_name"
tCustomer.MoveFirst
```

```
// JScript
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 places.

- 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.

Example

The following code retrieves the value of three columns from the tCustomer ULTable object, and displays them in text boxes.

```
' eMbedded Visual Basic
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
```

```
// JScript
var colID, colFirstName, colLastName;
colID = tCustomer.Columns.Item(1);
colFirstName = tCustomer.Columns.Item(2);
colLastName = tCustomer.Columns.Item(3);
txtID.Text = colID.Value;
txtFirstName.value = colFirstName.Value;
txtLastName.value = colLastName.Value;
```

You can also use the Value property to set values.

```
' eMbedded Visual Basic
colLastName.Value = "Kaminski"
```

```
// JScript
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. You cannot, however, get values from the column. For example, the following code generates an error.

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

```
// This JScript 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 rows with find and lookup

UltraLite has several modes of operation for working with data. Two of these modes, the find and lookup modes, are used for searching. The ULTable object has methods corresponding to these modes for locating particular rows in a table.

**Note**
The columns searched using Find and Lookup methods must be in the index used to open the table.

- **Find methods** move to the first row that exactly matches a specified search value, under the sort order specified when the ULTable object was opened.

  ☞ For more information about find methods, see "FindBegin method" on page 143.

- **Lookup methods** 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.

  ☞ For more information about lookup methods, see "LookupBackward method" on page 146.

### ❖ To search for a row

1. Enter find or lookup mode.

   Call the FindBegin or LookupBegin method. For example, the following code calls ULTable.FindBegin.

   ```
   ' eMbedded Visual Basic
   tCustomer.FindBegin

   // JScript
   tCustomer.FindBegin();
   ```

2. Set the search values.

   You do this by setting values in the current row. Setting these values affects the buffer, not the database. For example, the following code sets the last name column in the buffer to Kaminski.

```
' eMbedded Visual Basic
tCustomer.Columns.Item(3).Value = "Kaminski"
```

```
// JScript
tCustomer.Columns.Item(3).Value = "Kaminski";
```

For multi-column indexes, a value for the first column is required, but you can omit the other columns.

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.

```
' eMbedded Visual Basic
tCustomer.FindFirst
```

```
// JScript
tCustomer.FindFirst();
```

## Inserting, updating, and deleting rows

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

When your application changes location, 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.

Example

The following statement changes the value of the ID column in the buffer to 3.

```
' eMbedded Visual Basic
colID.Value = 3
```

```
// JScript
colID.Value = 3;
```

Using UltraLite modes

The UltraLite mode determines the purpose for which the values in the buffer will be used. UltraLite has the following four modes of operation, in addition to a default mode.

- **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** Used to locate a row whose value exactly matches the data in the buffer when one of the ULTable.Find methods is called.

- **Lookup mode** Used to locate a row whose value matches or is greater than the data in the buffer when one of the ULTable.Lookup methods is called.

## ❖ To update a row

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.

   ```
   ' eMbedded Visual Basic
   tCustomer.UpdateBegin
   // JScript
   tCustomer.UpdateBegin();
   ```

3. Set the new values for the row to be updated.

   For example, the following instruction sets the new value to Elizabeth.

   ```
   ' eMbedded Visual Basic
   ColFirstName.Value = "Elizabeth"
   // JScript
   ColFirstName.Value = "Elizabeth";
   ```

4. Execute the Update.

   ```
   'eMbedded Visual Basic
   tCustomer.Update
   // JScript
   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, the current position is undefined.

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 32.

**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 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 by the index specified when opening the table.

### ❖ To insert a row

1. Enter Insert mode.

   For example, the following instruction enters Insert mode on the table CustomerTable.

   ```
   ' eMbedded Visual Basic
   CustomerTable.InsertBegin
   ```

   ```
   // JScript
   CustomerTable.InsertBegin();
   ```

2. Set the values for the new row.

   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.

   ```
   ' eMbedded Visual Basic
   CustomerTable.Columns.Item("Fname").Value = fname
   CustomerTable.Columns.Item("Lname").Value = lname
   ```

   ```
   // JScript
   CustomerTable.Columns("Fname").Value = fname;
   CustomerTable.Columns("Lname").Value = lname;
   ```

3. Execute the insertion.

   The inserted row is permanently saved to the database when a Commit is carried out. In AutoCommit mode, a Commit is carried out as part of the Insert method.

   ```
   'eMbedded Visual Basic
   CustomerTable.Insert
   ```

   ```
   // JScript
   CustomerTable.Insert();
   ```

Deleting rows

There is no delete mode corresponding to the insert or update modes.

The following procedure deletes a row.

### ❖ To delete a row

1. Move to the row you wish to delete.

2. Execute the deletion.

```
'eMbedded Visual Basic
tCustomer.Delete
```

```
// JScript
tCustomer.Delete();
```

## Working with BLOB data

You can fetch BLOB data for columns declared BINARY or LONG BINARY using the GetByteChunk method.

☞ For more information, see "GetByteChunk method" on page 86.

Example

The following code demonstrates how to use the ULColumn.GetByteChunk method to get BLOB data.

```
' eMbedded Visual Basic
Dim offset As Integer
Dim nBytes As Integer
Dim chunk(1024) As Byte
ll = col.GetByteChunk( 0, chunk, 1024)
If nBytes <> 1024 Then
   ' only got nBytes bytes, expected 1024
Else
   ' have 1024 bytes
End if
```

```
// JScript
var offset;
var nBytes;
var chunk = new Array();
nBytes = col.GetByteChunk( 0, chunk, 1024);
if ( nBytes != 1024 ) {
  // only got nBytes bytes, expected 1024
} else {
  // have 1024 bytes
}
```

## Transaction processing in UltraLite

UltraLite provides transaction processing to ensure the integrity of the data in your database. A transaction is a logical unit of work. Either the entire transaction is executed, or none of the statements in the transaction are executed.

By default, UltraLite operates in AutoCommit mode. In AutoCommit mode, 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, the deduction from the source account and the addition to the destination account constitute a single transaction. If AutoCommit is false, you must execute a ULConnection.Commit statement to complete a transaction and make changes to your database permanent, or you may execute a ULConnection. Rollback statement to cancel all the operations of a transaction. Turning off AutoCommit improves performance.

**Note**
Synchronization causes a Commit even if you have AutoCommit set to False.

## Accessing schema information

Each ULConnection, ULTable, and ULColumn object contains a schema property. These schema objects provide information about the tables, columns, indexes, and publications in a database.

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

☞ For information about modifying the schema, see "Changing the schema of a database" on page 9.

- **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 the table.

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

- **ULColumnSchema** Information about an individual column, including its default value, name, and whether it is autoincrement.

  To obtain a ULColumnSchema object, access the ULColumn.Schema property.

- **ULIndexSchema** Information about the column, or columns, in the index. As an index has no data directly associated with it, there is no separate ULIndex object, only a ULIndexSchema object.

  The ULIndexSchema objects are available as part of the ULTableSchema.Indexes collection.

- **ULPublicationSchema** The numbers and names of tables and columns contained in a publication. Publications are also comprised of schema only, so there is a ULPublicationSchema object but no ULPublication object.

  The ULPublicationSchema objects are available as part of the ULDatabaseSchema.Publications collection.

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

  The ULResultSetSchema objects are accessible using the ULPreparedStatement.Schema property.

## Handling errors

In normal operation, UltraLite ActiveX can throw errors. Errors are expressed as SQLCODE values, negative numbers indicating the particular kind of error.

☞ For a list of error codes thrown by UltraLite ActiveX, see "ULSQLCode enumeration" on page 123.

UltraLite ActiveX throws errors only from the ULDatabaseManager and ULConnection objects. The following methods of ULDatabaseManager can throw errors.

- CreateDatabase
- CreateDatabaseWithParms
- DropDatabase
- DropDatabaseWithParms
- OpenConnection
- OpenConnectionWithParms

All other errors and exceptions within UltraLite ActiveX are routed through the ULConnection object.

☞ For more information about accessing error numbers from ULDatabaseManager and ULConnection objects, see "ULConnection class" on page 91 and "ULDatabaseManager class" on page 101.

The following sections explain how to implement error handling in eMbedded Visual Basic and JScript.

### Error handling in eMbedded Visual Basic

You can use the standard eMbedded Visual Basic error-handling features to handle errors. To enable error handling, use the On Error Resume Next statement. On Error Goto 0 causes errors to halt execution of your code.

The Err.Number property holds the SQLCODE value for an error. You can also get the last error using the ULConnection.LastErrorCode property.

One common area where errors need to be caught is in handling missing database files. In the following example, On Error Resume Next is in effect, so control flows to the next statement following the one causing the error. If the specified database does not exist, the Err object is loaded with the error number and description.

```
' eMbedded Visual Basic
On Error Resume Next
Err.Clear     ' Clear any previous errors
Set Connection = DBMgr.OpenConnection(udb)
If Err.Number <> 0 Then   ' Connection failed, no database?
  Err.Clear
  Set Connection = DBMgr.CreateDatabase(udb & usm)
  If Err.Number <> 0 Then
    MsgBox "Connect with " & udb & usm & " failed: " &
         Err.Description
    App.End
  End If
End If
```

## Error handling in JScript

There is no error handling in the version of JScript supported by Pocket IE. The default Pocket IE settings cause script errors to be ignored and the script to be silently terminated. When developing new JScript, this makes development difficult.

**Set ShowScriptErrors registry key**

Using a remote registry editor, such as the one that comes with eMbedded Visual Basic, you should create this key on your CE device:

```
[HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\
         Main]ShowScriptErrors=dword:00000001
```

With this key set, Pocket Internet Explorer will provide error notification messages for JScript code that fails. Note that the line number reported may not be reliable.

**Set ErrorResume property**

To suppress UltraLite ActiveX errors, the ULDatabaseManager and ULConnection objects provide the following two properties.

- **ErrorResume** is set to True to disable throwing subsequent errors.
- **LastErrorCode** returns the error code set by the last operation.

One common area where errors need to be caught is in handling missing database files, which are to be created from a schema. In the following example, the ULDatabaseManager.ErrorResume property is True, so control flows to the next statement following the one causing the error. If the specified database does not exist, the LastErrorCode property is loaded with the error number.

```
// JScript
DBMgr.ErrorResume = true;   // Do not throw errors
Connection = DBMgr.OpenConnection( udb );
if ( DBMgr.LastErrorCode != 0 ) {
  Connection = DBMgr.CreateDatabase( udb + usm );
  if ( DBMgr.LastErrorCode != 0 ) {
    alert("Connect with " + udb + usm + failed: " +
         DBMgr.LastErrorCode );
  }
}
```

## Authenticating users

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.

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.

☞ For more information about granting or revoking connection authority, see "GrantConnectTo method" on page 94 and "RevokeConnectFrom method" on page 95.

### ❖ To add a user or change the password for an existing user

1. Connect to the database as a user with DBA authority.
2. Grant the user connection authority with the desired password.

```
'eMbedded Visual Basic
conn.GrantConnectTo("Robert", "newPassword")

// JScript
conn.GrantConnectTo("Robert", "newPassword");
```

### ❖ To delete an existing user

1. Connect to the database as a user with DBA authority.
2. Revoke the user's connection authority as follows.

```
'eMbedded Visual Basic
conn.RevokeConnectFrom("Robert")

// JScript
conn.RevokeConnectFrom("Robert");
```

## Synchronizing data

You can UltraLite applications with a central database. Synchronization requires the MobiLink synchronization server and appropriate licensing.

This section provides a brief introduction to synchronization and describes some features of particular interest to users of UltraLite ActiveX. For a detailed explanation of synchronization, see "UltraLite Clients" [*MobiLink Clients*, page 277].

You can also find a working example of synchronization in the CustDB sample application. For eMbedded Visual Basic, this sample is described in "Tutorial: A Sample UltraLite ActiveX Application" on page 43. For JScript, this sample is described in "Tutorial: An UltraLite ActiveX Application for Pocket IE" on page 63.

UltraLite ActiveX supports TCP/IP, HTTP, and HTTPS synchronization. Synchronization is initiated by the UltraLite application. In all cases, you use methods and properties of the ULConnection object to control synchronization.

**Note**
To synchronize using encrypted synchronization (HTTPS) or to use encryption over TCP/IP you must obtain the separately-licensable security option. To order this option, see the card in your SQL Anywhere Studio package or see *http://www.sybase.com/detail?id=1015780*.

☞ For more information, see "Welcome to SQL Anywhere Studio" [*Introducing SQL Anywhere Studio*, page 4].

### ❖ To synchronize over TCP/IP or HTTP

1. Prepare the synchronization information.

   Assign values to the required properties of the ULConnection.SyncParms object.

   ☞ For information about the properties and the values that you should set, see "UltraLite Clients" [*MobiLink Clients*, page 277].

2. Synchronize.

   Call the ULConnection.Synchronize method.

## Monitoring synchronization progress

This section applies only to eMbedded Visual Basic. You cannot monitor synchronization progress using JScript.

To monitor synchronization progress, you add a synchronization dialog to your project and code accordingly.

The following procedure causes the dialog in *ULSyncStatus.ebf* to show status messages.

### ❖ To add a synchronization status dialog to your project (eMbedded Visual Basic)

1. Browse to the *Samples\UltraLiteActiveX\dbview.evb* subdirectory of your SQL Anywhere installation.

2. Copy and add *ULSyncStatus.bas* and *ULSyncStatus.ebf* to your project.

3. Add code to your UltraLite ActiveX project.

   ♦ Instead of using CreateObject to instantiate your ULDatabase Manager, use the following instruction:

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

   ♦ Ensure that your synchronization call accepts callbacks as follows.

   ```
   connection.Synchronize(True)
   ```

   ♦ Write code that captures event notifications in the Synchronization Progress Dialog.

   The following 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 following 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 following method shows users when synchronization states are changed.

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

The following method shows users when the currently synchronized table changes.

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

# Deploying UltraLite applications

When you have completed your application or when you wish to test your application, you need to deploy it to a device. This section outlines the steps needed to deploy an UltraLite application to a device.

## Deploying UltraLite ActiveX applications to Windows CE

You must carry out the following steps to deploy an UltraLite application to Windows CE:

- Deploy your application and UltraLite component.
- Deploy an initial copy of the UltraLite database or schema.

  In many situations it is sufficient to deploy an UltraLite schema file only. UltraLite creates a database file from the schema on the first connection attempt. You can then use synchronization to load an initial copy of the data.

  You must place the database or schema file so that it can be located by the application. The Database On CE and Schema On CE connection parameters define the location.

  ☞ See "Database On CE connection parameter" [*UltraLite Database User's Guide,* page 69], and "Schema On CE connection parameter " [*UltraLite Database User's Guide,* page 78].

- Deploy the MobiLink synchronization conduit for ActiveSync.

  This step is required only if the application is synchronizing using ActiveSync.

  ☞ For instructions, see "Installing the MobiLink provider for ActiveSync" [*MobiLink Clients,* page 310]

CHAPTER 3

# Tutorial: A Sample UltraLite ActiveX Application

About this chapter

This chapter provides a tutorial to guide you through the process of building an UltraLite ActiveX application for a PocketPC device.

This tutorial uses eMbedded Visual Basic. For a sample UltraLite ActiveX application using JScript, see "Tutorial: An UltraLite ActiveX Application for Pocket IE" on page 63.

Contents

# Introduction

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

☞ For more information about the table API, see the "UltraLite ActiveX API Reference" on page 79.

**Timing**

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

**Competencies and experience**

This tutorial assumes:

♦ you have Microsoft eMbedded Visual Tools installed on your computer

♦ you can write, test, and troubleshoot an eMbedded Visual Basic application

♦ you know how to create an UltraLite schema using the UltraLite Schema Painter

  ☞ For more information, see "The UltraLite Schema Painter" [*UltraLite Database User's Guide*, page 124].

♦ you have the AppForge Booster installed

  If you are missing Booster, you can get it from *http://www.appforge.com/booster.html*.

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

**Goals**

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

## Lesson 1: Create a project architecture

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

☞ For more information about database schemas, see "Creating UltraLite database schema files" on page 9.

### ❖ To create an UltraLite database schema

1. Create a directory for this tutorial.

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

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

   ☞ For more information about creating a database schema, see the "Lesson 1: Create an UltraLite database schema" [*UltraLite Database User's Guide*, page 130].

   ♦ **Schema filename** tutcustomer.usm

   ♦ **Table name** customer

   ♦ **Columns in customer**

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

   ♦ **Primary key** ascending id

## Create an eMbedded Visual Basic project

The UltraLite component for eMbedded Visual Basic development is UltraLite ActiveX.

The following procedure creates an eMbedded Visual Basic project for your application and adds a reference to the UltraLite ActiveX control.

### ❖ To create a reference to the UltraLite ActiveX control

1. Start eMbedded Visual Basic.
   - ♦ Choose Start ➤ Programs ➤ Microsoft eMbedded Visual Tools ➤ eMbedded Visual Basic 3.0.

   The New Project window appears.

2. Create a new project.

   Choose a design target for your application and click Open. The remainder of the tutorial assumes that you have chosen Windows CE for the PocketPC project.

3. Create a reference to UltraLite ActiveX.
   - ♦ Choose Project ➤ References
   - ♦ Select iAnywhere Solutions, ActiveX for UltraLite 9.0 and click OK.

     If the control does not appear in the list of available references, add the control to the list of available references.

     - Browse to the *UltraLite\UltraLiteActiveX\win32\* subdirectory of your SQL Anywhere installation.
     - Select *uldo9.dll* and click OK.

4. Save the Project:
   - ♦ Choose File ➤ Save Project.
   - ♦ Save the form as *c:\tutorial\evb\Tutorial.ebf.*
   - ♦ Save the project as *c:\tutorial\evb\Tutorial.ebp.*

## Lesson 2: Create a form

After completing the steps in "Lesson 1: Create a project architecture" on page 45, the project should display a form. The following procedure uses the form to create a user interface. This example uses labels as outputs, and text boxes and buttons as inputs, similar to a real application.

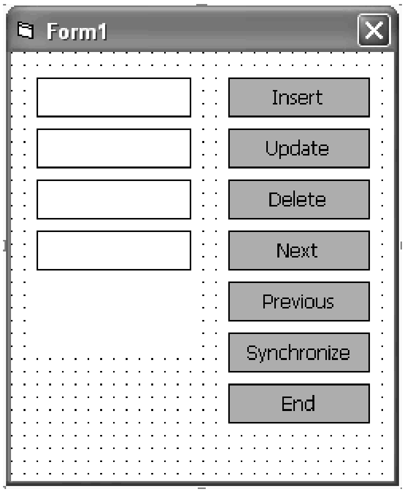
### ❖ To add controls to your project

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

| Type | Name | Caption or text |
|---|---|---|
| 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 |
| Button | btnDone | End |

2. Check the application.
   - ♦ Choose Run ➤ Execute.
     The application appears in the Windows CE emulator.
   - ♦ To end the application, click OK in the top right corner of the form.

Your form should look like the following screen capture:



## Configure the emulator to support UltraLite ActiveX

Once you add UltraLite objects to your application, you must add the UltraLite ActiveX control to the emulator in order to debug and test your application.

### ❖ To add UltraLite ActiveX to the emulator

1. Start the Control Manager.
   - In eMbedded Visual Basic, select Tools ➤ Remote Tools ➤ Control Manager.

2. Select the target emulator.
   - In the left pane, open Pocket PC and select Pocket PC Emulation.

3. Add the UltraLite control.
   - Choose Control ➤ Add New Control.
   - Browse to *ultralite\UltraLiteActiveX\ce\emulator30\uldo9.dll*, located in your SQL Anywhere directory.
   - Click OK.

     Note that *uldo9.dll* is an ActiveX file, not a control, so it will not appear in the list of controls on the device.

## Deploy the database schema

In addition to the UltraLite control, you must deploy the database schema to the emulator. The following procedure ensures that when your application first connects to a database, it uses the schema to create a database file.

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

1. Start the Windows CE File Viewer.
   - ♦ From eMbedded Visual Basic, choose Tools ➤ Remote Tools ➤ File Viewer.
   - ♦ If the File Viewer does not automatically connect to the emulator, choose Connection ➤ Add Connection and select Pocket PC Emulation.
2. Create a folder to hold your application.
   - ♦ Open the Program Files folder.
   - ♦ In File Viewer, choose File ➤ New Folder.
   - ♦ Create a folder named *tutorial*. This folder is used to hold your application files.
   - ♦ Click *tutorial* to navigate to that folder.
3. Deploy the schema file to the emulator.
   - ♦ Choose File ➤ Export File.
   - ♦ Browse to *c:\tutorial\evb* and double-click *tutcustomer.usm*.

# Lesson 3: Write the sample code

This lesson guides you through the process of writing eMbedded Visual Basic code to connect to a database, navigate within the database, and manipulate the data in the database.

This lesson also includes instructions for synchronizing your application with an Adaptive Server Anywhere database. This portion of the lesson is optional, and requires SQL Anywhere Studio.

## Write code to connect to your database

In this application, you connect to the database during the Form_Load event. You can also connect to a database using the general module.

This example uses a ULConnectionParms object to connect to a database. You can also use a connection string.

☞ For reference information, see "ULConnectionParms class" on page 98.

### ❖ Write code to connect to the UltraLite database

1. Double-click the form to open the Code window.

2. Declare the required UltraLite objects.

   Enter the following code in the General area of your form.

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

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

   In the code below, the CreateObject method is used to create the initial database manager object. The database manager tries to open a connection to the database specified by the ULConnectionParms1 object. If the database does not exist, it creates a new database using the given schema.

```
Sub Form_Load()
  ' declare variables
     Dim open_parms As String
  Dim schema_parms As String
  ' enable error handling
  On Error Resume Next
  ' specify the schema and database file
  ' locations
  open_parms = ";ce_file=\Program Files\tutorial\
         tutCustomer.udb"
  schema_parms = open_parms & ";ce_schema=\Program Files\
         tutorial\tutCustomer.usm"
  ' create a database manager and try to
   ' connect to an existing database
  Set DatabaseMgr =
         CreateObject("UltraLite.ULDatabaseManager")
  Set Connection = DatabaseMgr.OpenConnection(open_parms)
  ' if the database does not exist, create one
  If Err.Number = UlSQLCode.ulSQLE_NOERROR Then
    MsgBox "Connected to an existing database."
  ElseIf Err.Number = UlSQLCode.ulSQLE_ULTRALITE_DATABASE_
         NOT_FOUND Then
    Err.Clear
    Set Connection = DatabaseMgr.CreateDatabase(schema_
         parms)
    If Err.Number <> 0 Then
      MsgBox Err.Description
    Else
      MsgBox "Connected to a new database"
      End If
  End If
End Sub
```

4. Add code to end the application and close the connection when the End button is clicked.

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

5. Run the application.
   - Choose Run ➤ Execute.
   - After an initial message box, the form loads.
   - To terminate the application, click OK in the top right corner.

## Write code for navigation and data manipulation

The following procedures implement data manipulation and navigation.

## ❖ To open the table

1. Write code to initialize the table and move to the first row.

   This code assigns the customer table in the database to the CustomerTable variable. The call to Open opens the table so that the table data can be read or manipulated. It also positions the application before the first row in the table.

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

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

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

   If the table has no rows, the following procedure causes the application to display empty controls. Otherwise, it displays the values stored in each of the columns of the current row of the database.

   ```
   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").Value
       txtPhone.Text = CustomerTable.Columns("Phone").Value
     End If
   End Sub
   ```

3. Call DisplayCurrentRow from the Form_Activate procedure. This call ensures that the fields get updated when the application starts.

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

### ❖ To insert rows into the table

1. Write code to implement the Insert button.

   In the following procedure, 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.

   Add the following procedure to the form.

   ```
   Private Sub btnInsert_Click()
     Dim fname As String
     Dim lname As String
     Dim city As String
     Dim phone As String
     fname = txtFname.Text
     lname = txtLname.Text
     city = txtCity.Text
     phone = txtPhone.Text
     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
   ```

2. Run the application.

   After an initial message box, the form is displayed.

3. Insert two rows into the database.

   ♦ Enter a first name of Jane in the first text box and a last name of Doe in the second. Click Insert.

     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 automatically incremented value of the ID column that UltraLite assigned to the row.

   ♦ Enter a first name of John in the first text box and a last name of Smith in the second. Click Insert.

4. Click OK to end the program.

### ❖ To move through the rows of the table

1. Write code to implement the Next and Previous buttons.

   Add the following procedures to the form.

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

2. Run the application.

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

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

### ❖ To update and delete rows in the table

1. Write code to implement the Update button.

   In the code below, 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.

   Add the following procedure to the form.

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

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

2. Write code to implement the Delete button.

   In the code below, the call to Delete deletes the current row on which the application is positioned.

   Add the following procedure to the form.

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

3. Run the application.

## Write code to synchronize

The following procedure implements synchronization. Synchronization requires SQL Anywhere Studio. This portion of the tutorial is optional.

### ❖ To write code for the synchronize button

1. Write code to implement the Synchronize button.

   In the code below, the ULSyncParms object contains the synchronization parameters. For example, the ULSyncParms.UserName property specifies that when MobiLink is started, it will add a new user. The ULSyncParms.SendColumnNames property specifies that the column names will be sent to MobiLink so it can generate upload and download scripts.

   Add the following procedure 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
   ```

## Synchronize your application

The ASA 9.0 Sample database has a Customer table with columns matching those in the **customer** table in your UltraLite database. The following procedure synchronizes your database with the ASA 9.0 Sample database.

### ❖ To synchronize your application

1. From a command prompt, start the MobiLink synchronization server by running the following command line.

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

   The `-zu+` and `-za` command line options provide automatic addition of users and generation of synchronization scripts. For more information about these options, see "MobiLink Synchronization Server Options" [*MobiLink Administration Guide*, page 189].

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 9.0 Sample database.

4. Synchronize your application.

   Click Synchronize.

   The MobiLink synchronization server window displays the synchronization progress.

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

# Lesson 4: Deploy to a device

You can deploy your application to a device manually, or using the Application Install Wizard. The following sections describe both procedures.

## Deploy to a remote device manually

The following procedures deploy your application to a Windows CE device manually.

### ❖ To add the UltraLite ActiveX control to your device

1. Start the Windows CE Control manager.

   Select Tools ➤ Remote Tools ➤ Control Manager.

2. In the left pane, double-click your device type and select your device. Your device must be connected.

   The right pane shows the controls available on the selected device.

3. Choose Control ➤ Add New Control.

4. Browse to the appropriate version of the UltraLite ActiveX control, located in one of the following platform-specific subdirectories of your SQL Anywhere installation:

   - ♦ **ARM** *ultralite\UltraLiteActiveX\ce\arm\uldo9.dll*
   - ♦ **MIPS** *ultralite\UltraLiteActiveX\ce\mips\uldo9.dll*

5. Click OK.

   Alternatively, you can copy the DLL to the *\Windows* directory on the device and register it using *regsvrce.exe*.

### ❖ To deploy the database schema file to your device

1. Start the Windows CE File Viewer.

   From eMbedded Visual Basic, choose Tools ➤ Remote Tools ➤ File Viewer.

2. If the File Viewer does not automatically connect to your device, choose Connection ➤ Add Connection and select your device.

3. Create a folder to hold your application.

   - ♦ In the File Viewer, open the Program Files folder.
   - ♦ Choose File ➤ New Folder.

- ♦ Create a folder named *tutorial*. This folder is used to hold your application files.
- ♦ Click *tutorial* to navigate to that folder.

4. Deploy the schema file to your device.
    - ♦ Choose File ➤ Export File.
    - ♦ Browse to *c:\tutorial\evb* and double-click *tutcustomer.usm*.

### ❖ To deploy your application to a remote device

1. Select File ➤ Make.
2. Browse to *c:\tutorial\evb*. Name your project Tutorial. Click OK.
3. Deploy the *.vb* file to your device.
    - ♦ In the File Viewer, browse to *Program Files\tutorial*.
    - ♦ Choose File ➤ Export File.
    - ♦ Browse to *c:\tutorial\evb* and double-click *Tutorial.vb*.

You can now launch the sample application on your remote device by browsing to *Program Files\tutorial* and launching *Tutorial.vb*.

## Deploy to a remote device using the Application Install Wizard

The following procedure deploys your application to a mobile device using the Application Install Wizard. The Application Install Wizard creates an executable file and a cabinet (*.cab*) file that contains the files necessary for deployment.

### ❖ To deploy using the Application Install Wizard

1. Before using the Application Install Wizard, you must complete your project and execute it to ensure that it does not generate run-time errors.
2. Select File ➤ Make.
3. Browse to *c:\tutorial\evb*. Name your project Tutorial. Click OK.
4. Select Tools ➤ Remote Tools ➤ Application Install Wizard.

    The Application Install Wizard appears. Click Next.
5. Browse to *c:\tutorial\evb\Tutorial.ebp*. Click Next.
6. Browse to *c:\tutorial\evb\Tutorial.vb*. Click Next.
7. Browse to *c:\tutorial\evb\*. Click Next.

8. Select the processors your application will support. If you select more than one processor, each will be contained in a separate *.cab* file. Click Next.

9. Select iAnywhere Solutions, ActiveX for UltraLite. The UltraLite ActiveX control will be registered automatically upon installation of your application. Click Next.

10. Deploy the schema file to your device.
    - ♦ Click Add.
    - ♦ Browse to *c:\tutorial\tutcustomer.usm.*
    - ♦ Click Open.
    - ♦ If a prompt asks whether it is a system file, click No.
    - ♦ Check Include Device Runtimes in Cab file if you want the device runtimes to be included in your installation. The device runtimes are not always required and can make the *.cab* file very large if included. If the application works with the currently installed runtimes, there is no need to check this option.
    - ♦ Click Next.

11. Enter **tutorial** in each of the fields. Click Next.

12. Click Create Install to create the *.cab* files and setup executable.

    The wizard creates a folder, *CD1*, in the directory you specified to store the output files. *CD1* contains all the files you need to distribute your application.

13. Click Finish.

14. Run *c:\tutorial\evb\CD1\setup.exe* on your desktop computer to install your application to the device attached to it.

You can now launch your application on your device by browsing to *Program Files\tutorial* and launching *Tutorial.vb.*

## Summary

**Learning accomplishments**

During this tutorial, you:

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

**More samples**

You can find more sample applications and utilities at iAnywhere CodeXchange.

CHAPTER 4

# Tutorial: An UltraLite ActiveX Application for Pocket IE

About this chapter

This chapter provides a tutorial to guide you through the process of building an UltraLite ActiveX application for Pocket Internet Explorer. The application accesses the UltraLite ActiveX package using JScript embedded on an HTML page.

☞ For a sample UltraLite ActiveX application using eMbedded Visual Basic, see "Tutorial: A Sample UltraLite ActiveX Application" on page 43.

Contents

# Introduction

This tutorial guides you through the process of building an UltraLite ActiveX application. At the end of the tutorial you will have an application and small database on your Windows CE device that synchronizes with a database running on your desktop computer.

This tutorial describes the CustDB sample application. This application demonstrates the capabilities of UltraLite ActiveX used from Pocket Internet Explorer (Pocket IE). It runs on a Windows  CE device.

The CustDB sample is fully-functional customer application. The CustDB sample application provides you with examples of how to implement many of the techniques you will need to develop UltraLite ActiveX applications.

The CustDB sample application is written in JScript and HTML. The code for this application is located in the *Samples\UltraLiteActiveX\pie* subdirectory of your SQL Anywhere 9 installation.

**Note**
This application uses frames and so will only work on versions of Pocket IE later than 1.1.

**Timing**

The tutorial takes about 50 minutes.

**Competencies and experience**

This tutorial assumes:

- ♦ you are familiar with JScript and Pocket Internet Explorer
  - you can write, test, and troubleshoot a JScript application

The synchronization section of this tutorial requires that:

- ♦ you can use command line options and parameters

**Goals**

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

## Lesson 1: Install the UltraLite ActiveX package

The UltraLite component for JScript development is UltraLite ActiveX. The following procedure registers the UltraLite ActiveX control on a remote device.

### ❖ To register the UltraLite ActiveX control

1. Start eMbedded Visual Basic 3.0.

   Choose Start ➤ Programs ➤ Microsoft eMbedded Visual Tools ➤ eMbedded Visual Basic 3.0.

   The New Project window appears.

2. Click Cancel.

3. Choose Tools ➤ Remote Tools ➤ Control Manager.

4. In the left pane, open the folder corresponding to your device type. Select your device.

   The Control Manager connects to your device.

5. Choose Control ➤ Add New Control.

6. Browse to one of the following platform-specific files in subdirectories of your SQL Anywhere 9 installation.

   ♦ **ARM** *UltraLite\UltraLiteActiveX\ce\arm\uldo9.dll*

   ♦ **MIPS** *UltraLite\UltraLiteActiveX\ce\mips\uldo9.dll*

   ♦ **Emulator** *UltraLite\UltraLiteActiveX\ce\emulator30\uldo9.dll*

7. Click Open.

   The ULConnectionParms class and the ULDatabaseManager class are added to your device.

Alternatively, you can copy the appropriate DLL for your device to *\Windows\uldo9.dll* and register it using *regsvrce.exe*. If you do not have *regsvrce.exe* on your device, you can copy it to your device from the Microsoft Windows CE SDK.

## Lesson 2: Deploy to a device

The following procedure copies the HTML files containing embedded JScript to your remote device. Alternatively, you can access the files remotely from your hard drive or server using a web server and an internet connection.

### ❖ To copy the HTML files to your device

1. Start the File Viewer.

   From eMbedded Visual Basic 3.0, choose Tools ➤ Remote Tools ➤ File Viewer.

2. In the left pane, open the folder corresponding to your device type. Select your device.

   The File Viewer connects to your device.

3. Choose File ➤ New Folder. Create a folder named *pie* in the root of your device.

4. Choose File ➤ Export File.

   Copy the contents of the *Samples\UltraLiteActiveX\pie* subdirectory of your SQL Anywhere 9 installation to the *pie* directory on your device.

## Lesson 3: Create and deploy an UltraLite database schema

The database schema is a description of the database. It describes the tables, indexes, keys, and publications within the database, and all the relationships between them.

A database schema may be created using the UltraLite Schema Painter or the *ulinit* utility. The following procedure uses ulinit to create a database schema based on an Adaptive Server Anywhere database.

### ❖ To create the database schema

1. Create a directory on your computer for this tutorial.

   The remainder of the tutorial assumes that this directory is *c:\tutorial\pie*. If you create a directory with a different name, use that directory instead of *c:\tutorial\pie* throughout the tutorial.

2. Copy the contents of the *Samples\UltraLiteActiveX\pie* subdirectory of your SQL Anywhere 9 installation to *c:\tutorial\pie*.

3. Create the consolidated database.

   Open a command prompt and navigate to *c:\tutorial\pie*. Run the following command:

   ```
   dbinit custdbsrv.db
   ```

4. Start the MobiLink synchronization server.

   Run the following command:

   ```
   runml.bat
   ```

5. Create the UltraLite schema using the ulinit utility.

   ☞ For more information about the ulinit utility, see "The ulinit utility" [*UltraLite Database User's Guide*, page 112].

   Run the following command:

   ```
   makeschemas.bat
   ```

### ❖ To deploy the database schema to your device

1. Start the File Viewer.

   From eMbedded Visual Basic 3.0, choose Tools ➤ Remote Tools ➤ File Viewer.

2. The File Viewer connects to your device.
   - ♦ Choose Connection ➤ Add New Connection.
   - ♦ Open the folder corresponding to your device type.
   - ♦ Select your device.
3. Choose File ➤ New Folder. Create a folder named *UltraLiteDB*.
4. Choose File ➤ Export File.
5. Browse to *c:\tutorial\pie\ul_custdb.usm*. Click Open to export the file to your device.

## Lesson 4: Create a form interface

The form interface consists of the visual elements of your application. The following procedure explains the form interface in *main.htm*. You can optionally create a new text file and create the form interface yourself.

### ❖ To add controls to your project

1. Open *c:\tutorial\pie\main.htm* in a text editor.

2. Scroll down to the following lines in *main.htm*.

```
<br>
Customer: <INPUT TYPE="text" NAME="txt_Custname" SIZE=15
         MAXLENGTH=40>
```

   These lines create the text boxes given in the table below.

| Caption | INPUT TYPE | NAME |
|---|---|---|
| Customer: | Text | txt_Custname |
| Product: | Text | txt_Prodname |
| Quantity: | Text | txt_Quant |
| Prince: | Text | txt_Price |
| Discount: | Text | txt_Discount |
| Status: | Text | txt_Status |
| Notes: | Text | txt_Notes |

3. Scroll down to the following lines in *main.htm*.

```
<br>
<INPUT NAME="b_Previous" TYPE="BUTTON" VALUE=" <Back "
         onClick="MoveBack()">
```

   These lines create the buttons given in the table below.

| INPUT TYPE | NAME | VALUE | OnClick |
|---|---|---|---|
| Button | b_Previous | < Back | MoveBack() |
| Button | b_Next | Next > | MoveForward() |
| Button | b_Approve | Approve... | ApproveOrder() |
| Button | b_Deny | Deny... | DenyOrder() |
| Button | b_Add | Add... | addNewOrder() |
| Button | b_Delete | Delete | deleteOrder() |
| Button | b_Synchronize | Synchronize | syncData() |

## Lesson 5: Write the JScript sample code

This lesson guides you through the process of writing JScript code to connect to a database, navigate within the database, and manipulate the data in the database.

The synchronization portion of this lesson require SQL Anywhere Studio.

### Write code to connect to an UltraLite database

In Pocket IE, UltraLite ActiveX is accessed using JScript embedded on an HTML page. When a page containing a connection script is loaded, a new database connection is created. When the browser moves to another page, all of the objects created on the previous page are discarded. The database connection is thus lost when the page is unloaded.

Losing application state, including database connections, whenever a form changes, is slow and expensive. To avoid this, it is recommended that Pocket IE applications use HTML frames to maintain application state. The database connection can be kept by the frameset document. Pocket IE requires at least two frames within a frameset, and draws a three (3) pixel border between the frames.

In the CustDB sample, the minimal frame defined in *topline.htm* is a placeholder that displays the application name at the top of the screen. The rest of the screen is used to display the other pages of the application.

To swap forms in and out of the frameset, a JScript function can use the **document.location.replace** method. For example, the following code fragment closes the current connection and returns to the connect form.

```
<SCRIPT>
function exitApp() {
  if ( top.Connection != null ) {
    top.Connection.Close();
  }
  document.location.replace("connect.htm");
}
</SCRIPT>
<INPUT NAME="b_Done" TYPE="BUTTON" VALUE="Done"
         onClick="exitApp()">
```

The following procedure connects to an UltraLite database. For more information about connecting to an UltraLite database, see “Connecting to an UltraLite database” on page 11.

❖ **To connect to the UltraLite database**

1. Start Pocket Internet Explorer.
2. Open *login.htm*.
3. Tap the hyperlink.

   The hyperlink links to *frames.htm*.

   The code in *frames.htm* creates a new database manager object and uses it to open a connection to the database specified by a connection string. If the database does not exist, it creates a new one.

   ☞ For more information, see "Connecting to an UltraLite database" on page 11.

```
<SCRIPT LANGUAGE="JScript">
var Connection;      // UltraLite Connection
var DatabaseMgr;  // UltraLite Database Manager
var CS;        // Current State object

// Initialize connection. Create database if not already
         present.
function initDatabase() {
  var udb, usm;
  udb = "file_name=\\UltraLiteDB\\ul_custapi.udb";
  usm = ";schema_file=\\UltraLiteDB\\ul_custdb.usm";
  CS = new CurrState( 0 );
  DatabaseMgr = new ActiveXObject(
         "UltraLite.ULDatabaseManager" );
  var bval = DatabaseMgr.ErrorResume;
  DatabaseMgr.ErrorResume = true;
  Connection = DatabaseMgr.OpenConnection( udb );
  if ( DatabaseMgr.LastErrorCode != 0 ) {
      Connection = DatabaseMgr.CreateDatabase( udb + usm );
  }
}
// Initialize the database.
initDatabase();
</SCRIPT>
```

4. A script prompts you to enter an employee ID. Accept the default value of 50.

## Write code to synchronize the database

You must synchronize the database in order to obtain the list of products (which cannot be modified at the remote database) and an initial list of orders.

The following procedure synchronizes the database.

❖ **To synchronize the sample database**

1. Tap Synchronize.

   The code below runs.

   The ULConnection.Synchronize method takes a single parameter, *show-progress*. In JScript applications, *show-progress* is set to false because Pocket Internet Explorer does not allow messages to be displayed.

   ☞ For more information about synchronizing an UltraLite database, see "Synchronizing data" on page 39.

   ```
   function syncData() {
     var conn = top.Connection;
     var parms = conn.SyncParms;
     // Set Sync Params
     parms.Stream = 2;  // ulTCPIP = 2, ulHTTP = 1, ulHTTPS = 3
     parms.StreamParams = "";
     parms.UserName = "50";      // m_EmpIDStr;
     parms.Version = "custdb 9.0";
     conn.ErrorResume = true;
     conn.Synchronize(false);
     conn.ErrorResume = false;
     if ( conn.LastErrorCode != 0 ) {
         SetStatus("Sync failed: " + conn.LastErrorDescription
            );
         return;
     }
     conn.Commit();        // Save updates.
     SetStatus("Synchronized");
     SkipToValidOrder();
     SetOrderData();
   }
   ```

2. When synchronization is complete, the word Synchronized appears at the bottom of the screen.

## Write code to display order information

When *main.htm* loads, the following functions run to obtain and display the order information.

☞ For more information about the GetTable method, see "GetTable method" on page 94.

```
function OpenGetOrder() {
 var conn = top.Connection;
 var cs = top.CS;
 var empid;
 if ( conn == null ) {
    alert("Not yet connected!");
  return;
 }
 conn.AutoCommit = false;
 var table = conn.GetTable("ULIdentifyEmployee_nosync");
 table.Open();
 if ( table.RowCount == 0 ) {
    empid = window.prompt("Enter employee ID #: ", "50");
   table.InsertBegin();
    table.Columns("emp_id").value = empid;
   table.Insert();
    conn.Commit();
 }
table.MoveFirst();
 cs.SetEmployeeID( table.Columns("emp_id").value );
 table.Close();
 ProductList = conn.GetTable("ULProduct");
 ProductList.Open();
 CustomerList = conn.GetTable("ULCustomer");
 CustomerList.Open();
 OrderList = conn.GetTable("ULOrder");
 OrderList.Open();
 SetOrderData();
 SkipToValidOrder();
}
function UpdateForm() {
 var cs = top.CS;
 if ( cs.GetNoOrder() ) {
    txt_Custname.value = "";
    txt_Prodname.value = "";
    txt_Quant.value = "";
    txt_Price.value = "";
    txt_Discount.value = "";
    txt_Status.value = "";
    txt_Notes.value = "";
  return;
 }
txt_Custname.value = cs.GetCustName();
 txt_Prodname.value = cs.GetProdName();
 txt_Quant.value = cs.GetQuantity();
 txt_Price.value = cs.GetPrice();
 txt_Discount.value = cs.GetDiscount();
 txt_Status.value = cs.GetStatus();
 txt_Notes.value = cs.GetNotes();
 if ( cs.FirstOrder() == cs.GetOrderID() ) {
    SetStatus("First Order");
 } else if ( cs.LastOrder() == cs.GetOrderID() ) {
    SetStatus("Last Order");
 }
}
```

## Write code for navigation and data manipulation

The following procedures implement data manipulation and navigation.

### ❖ To scroll through the list of orders

1. Tap Back or Next.

   One of the following functions runs:

```
function MoveBack() {
  var cs = top.CS;
  var posn = cs.GetBookmark();
  if ( MoveOrder( -1 ) ) {
      cs.SetBookmark(posn - 1);
      UpdateForm();
  }
}
function MoveNext() {
  var cs = top.CS;
  var posn = cs.GetBookmark();
  if ( MoveOrder(1) ) {
    cs.SetBookmark(posn + 1);
    UpdateForm();
  }
}
```

### ❖ To add an order to the database

1. Tap Add.

   The following script runs, replacing *main.htm* with *add.htm* in the frame.

```
function addNewOrder() {
    document.location.replace("add.htm");
}
```

2. Fill in the form with the desired values. Click OK.

The following script runs:

☞ For more information about inserting rows, see "Inserting, updating, and deleting rows" on page 29.

```
function addCustomer() {
  var custName = window.prompt("Enter new customer name: ",
         "");
  if ( custName == "" ) return;
  var conn = top.Connection;
  var custlist = conn.GetTable( "ULCustomer" )
  custlist.Open("ULCustomerName");
  custlist.FindBegin();
  custlist.Columns("cust_name").value = custName;
  if ( custlist.FindFirst() ) {  // already present. Done.
    custlist.Close();
      return;
  }
var custid = NextCustomerID();
  if ( custid == -1 ) {
      alert("No more customer IDs, cannot add. Replenish
         ULCustomerIDPool");
      custlist.Close();
      return;
  }
  custlist.InsertBegin();
  custlist.Columns("cust_id").value = custid;
  custlist.Columns("cust_name").Value = custName;
  custlist.Insert();
  custlist.Close();
  conn.Commit();
  // Only way to show new customer is to reload this form!
  document.location.reload();
}
```

## ❖ To approve or deny an order

1. Click Approve or Deny.

   One of the following scripts runs, loading *approve.htm* or *deny.htm* in the frame.

   ```
   function ApproveOrder() {
     var cs = top.CS;
     document.location.replace("approve.htm");
   }
   function DenyOrder() {
     document.location.replace("deny.htm");
   }
   ```

2. Enter a note and click OK.

   If you are approving an order, the following code runs. If you are denying an order, the code is the same but the status is instead set to Denied.

☞ For more information about updating rows, see "Inserting, updating, and deleting rows" on page 29.

```
function OK_Approve() {
  var cs = top.CS;
  var conn = top.Connection;
  var notes = txt_Notes.value;
  var order = conn.GetTable("ULOrder");
  order.Open();
  order.FindBegin();
  order.Columns("order_id").value = cs.GetOrderID();
  if ( order.FindFirst() ) {
    order.UpdateBegin();
    order.Columns("status").value = "Approved";
    order.Columns("notes").value = notes;
    order.Update();
    if (conn.LastErrorCode != 0 ) {
      alert("Failed: " + conn.LastErrorDescription );
    }
    conn.Commit();
  } else {
    alert("Cannot find order " + cs.GetOrderID() );
  }
  order.Close();
  document.location.replace("main.htm");
}
```

CHAPTER 5

# UltraLite ActiveX API Reference

About this chapter

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

Contents

## IULColumns collection

A collection of ULColumn objects that provides you with metadata about the column.

### Properties

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

Example

You can enumerate the columns in a IULColumns collection using the For Each statement in eMbedded Visual Basic or the for ... in statement in JScript.

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

```
// JScript
var col : UltraLite.ULColumn;
var collection = table.Columns;
for ( col in collection ) {
  if ( col.IsNull ) {
    alert ( col.Schema.Name + "is null" );
  }
}
```

# IULIndexSchemas collection

A collection of ULIndexSchema objects that provides you with ULIndexSchema information.

## Properties

| Prototype | Description |
|---|---|
| Count as long (read-only) | Gets the number of indexes in the collection. |
| Item (Index) as ULIndexSchema (read-only) | Gets 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 the indexes on a table using the For Each statement in eMbedded Visual Basic or the for ... in statement in JScript.

```
' eMbedded Visual Basic
Dim index As ULIndexSchema
For Each index In TableSchema.Indexes
    'use index
Next
```

```
// JScript
var index : UltraLite.ULIndexSchema;
var collection = TableSchema.Indexes;
for ( index in collection ) {
  ' use index
}
```

## IULPublicationSchemas collection

A collection of ULPublicationSchema objects that provides you with information about ULPublicationSchema.

### Properties

| Prototype | Description |
|---|---|
| Count as long (read-only) | Gets the number of publications in the collection. |
| Item (Index) as ULPublicationSchema (read-only) | Gets a publication from the collection. |

Example

You can enumerate all the publications using the For Each statement in eMbedded Visual Basic or the for ... in statement in JScript.

```
’ eMbedded Visual Basic
Dim pub As ULPublicationSchema
For Each pub In connection.schema.publications
    ’use pub
Next
```

```
// JScript
var ps : UltraLite.ULPublicationSchema;
var collection = connection.schema.publications;
for ( pub in collection ) {
  ’ use pub
}
```

## ULAuthStatusCode enumeration

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

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

## ULColumn class

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

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

☞ For information about the table object, see "ULTable class" on page 142.

### Properties

| Prototype | Description |
|---|---|
| IsNull As Boolean (read only) | Indicates whether the column value is NULL. |
| Schema As ULColumn-Schema (read only) | Gets the object representing the schema of the column. |
| Value As Variant | Gets or sets the data value of this column in the current row as Variant. |

### AppendByteChunk method

Prototype

**AppendByteChunk(** _
  *byteArray* **,** _
  [ *chunkSize* ] _
**)** As Boolean
Member of **UltraliteActiveX.ULColumn**

Description

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

Parameters

**byteArray** The array of bytes to be appended.

**chunkSize** The number of bytes to be appended. If not provided, the length of byteArray is used.

Returns

**True** if successful.

**False** if unsuccessful.

| | |
|---|---|
| Errors set | **ulSQLE_CONVERSION_ERROR** The error occurs if the column data type is not LONG BINARY or BINARY. |
| Example | The following examples append data to the edata column. |

```
' eMbedded Visual Basic
Dim data (512) As Byte
' ...
table.Columns("edata").AppendByteChunk(data)
```

```
// JScript
var data = new Array();
// ...
table.Columns("edata").AppendByteChunk(data);
```

## AppendStringChunk method

| | |
|---|---|
| Prototype | **AppendStringChunk(** *chunk* **)**<br>Member of **UltraLiteActiveX.ULColumn** |
| Description | Appends the string to the column if the type is TypeLongString or TypeString. |
| Parameters | **data** A string to append to the existing string in a table. |
| Errors set | **ulSQLE_CONVERSION_ERROR** The error occurs if the column data type is not VARCHAR. |

## GetByteChunk method

| | |
|---|---|
| Prototype | **GetByteChunk (_**<br>*offset* As Long**, _**<br>*pByteArray***, _**<br>[ *chunkSize* ] _<br>**)** As Long<br>Member of **UltraliteActiveX.ULColumn** |
| Description | Fills the buffer passed in, which should be an array, with the binary data in the column. Suitable for BLOBS. |
| Parameters | **offset** The offset into the underlying array of bytes. The source offset must be greater than or equal to 0, otherwise a ulSQLE_INVALID_PARAMETER error will be raised.<br>**pByteArray** A variant. Array data is passed by reference as array.<br>**chunkSize** An optional parameter representing an array of bytes expressed as Long type. |
| Returns | The number of bytes read. |
| Errors set | **ulSQLE_CONVERSION_ERROR** The error occurs if the column data type |

isn't BINARY or LONG BINARY.

**ulSQLE_INVALID_PARAMETER** The error occurs if the column data type is BINARY and the offset is not 0 or 1, or, the data length is less than 0.

The error also occurs if the column data type is LONG BINARY and the offset is less than 1 or, the data length is less than 0.

Example

In the following example, edata is a column name.

```
' eMbedded Visual Basic
Dim data (512) As Byte
' ...
table.Columns.Item("edata").GetByteChunk(0,data)
```

```
// JScript
var data = new Array();
// ...
table.Columns.Item("edata").GetByteChunk(0, data);
```

## GetStringChunk method

Prototype

**GetStringChunk(** _
  *offset* As Long**,** _
  *pStringObj***,** _
  [ *chunkSize* ] _
**)** As Long
Member of **UltraliteActiveX.ULColumn**

Description

Fills the string passed in with the binary data from the column. Suitable for LONG VARCHAR columns.

Parameters

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

**pStringObj** The string array you want filled. This variant is passed by reference.

**chunkSize** An optional parameter representing the number of characters to retrieve.

Returns

The number of characters copied. Room is left for a null termination character and the length does not include that character.

Errors

**ulSQLE_CONVERSION_ERROR** The error occurs if the column data type isn't TypeString or TypeLongString.

**ulSQLE_INVALID_PARAMETER** The error occurs if the column data type is CHAR and the src_offset is greater than 64K.

The error also occurs if src_offset is less than 1 or string length is less than 0.

Example

```
' eMbedded Visual Basic
Dim cd As ULColumn
Dim S As Strong
Dim l, offset As Long
S=String(512, vbNulChar)
offset=0
Do
    l=cd.GetStringChunk(offset, S, 512)
    If l=0 then Exit Do
    'use string ins
Loop
```

```
// JScript
var cd;
var s;
var l, offset;
l = 0;
While (!l) {
  l = cd.GetStringChunk(offset, s, 512);
}
```

## SetByteChunk method

Prototype

**SetByteChunk (** _
  *ByteArray* **,** _
  [ *length* ] _
**)** As Boolean
Member of **UltraliteActiveX.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.

Errors set

**ulSQLE_CONVERSION_ERROR** The error occurs if the column data type is not BINARY or LONG BINARY.

**ulSQLE_INVALID_PARAMETER** The error occurs if the data length is less than 0 or greater than 64K.

Example

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

```
' eMbedded Visual Basic
Dim data (1 to 512) As Byte
' ...
table.Columns.Item("edata").SetByteChunk(data,232)
```

```
// JScript
var data = new Array();
// ...
table.Columns.Item("edata").SetByteChunk(data,232);
```

## SetToDefault method

| | |
|---|---|
| Prototype | **SetToDefault( )**<br>Member of **UltraliteActiveX.IColumn** |
| Description | Sets the current column to its default value as defined by the database schema. For example, an autoincrement column will be assigned the next available value. |

# ULColumnSchema class

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

## Properties

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

## ULConnection class

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

### Properties

The following are properties of ULConnection:

| Prototype | Description |
|---|---|
| AutoCommit As Boolean | Indicates the AutoCommit value. If true, all data changes are committed immediately after they are made. Otherwise, changes are not committed to the database until Commit is called. By default, this property is True. |
| CollationName As String (read-only) | Gets the database character set and sort order. |
| DatabaseID As Long | Gets or sets the database ID, which determines the starting value for global autoincrement columns. If the database ID has never been set, the value is -1. |
| DatabaseManager As ULDatabaseManager (read-only) | Gets the owning database manager object. |
| DatabaseNew As Boolean (read-only) | Indicates whether the database was newly created for this connection or not. |
| ErrorResume As Boolean | Indicates the error handling method. |
| GlobalAutoIncrementUsage As Long (read-only) | Gets the percentage of available global autoincrement values that have been used. |
| IsCaseSensitive As Boolean (read-only) | Indicates whether the database is case sensitive or not. |
| LastErrorCode As SQLCodeConstants | Gets the last error number, and allows you to clear the previous error code. |
| LastErrorDescription As String (read-only) | Gets the last error description. |
| LastIdentity As Long (read-only) | Gets the most recent value inserted into a column with a default of autoincrement or global autoincrement. |

| Prototype | Description |
|---|---|
| OpenParms As String (read-only) | Gets the string used to open the connection to the database. |
| Schema As ULDatabaseSchema (read-only) | Gets the ULDatabaseSchema object which represents the definition of the database. |
| SQLErrorOffset As Integer (read-only) | If PrepareStatement raises an error, indicates the 1-based offset in the SQL statement where the error was noted. If this value is less than or equal to 0, no offset information is available. |
| SyncParms As ULSyncParms (read-only) | Gets the synchronization parameters object. |
| SyncResult As ULSyncResult (read-only) | Gets the results of the most recent synchronization. |

## CancelSynchronize method

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

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

## ChangeEncryptionKey method

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

Description Encrypt the database with the specified key.

Parameters **newkey** The new encryption key value for the database.

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

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

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

## Close method

| | |
|---|---|
| Prototype | **Close( )**<br>Member of **UltraliteActiveX.ULConnection** |
| Description | Closes the connection to the database. No methods on the ULConnection object or any other database object for this connection should be called after this method is called. If a connection is not explicitly closed, it will be implicitly closed when the application terminates. |

## Commit method

| | |
|---|---|
| Prototype | **Commit( )**<br>Member of **UltraliteActiveX.ULConnection** |
| Description | Commits outstanding changes to the database. This is only useful if AutoCommit is false.<br><br>For more information, see Autocommit under ULConnection “Properties” on page 91 |

## CountUploadRows method

| | |
|---|---|
| Prototype | **CountUploadRows(**<br>  [ *mask* As Long = 0 ], _<br>  [ *threshold* As Long = -1 ] _<br>**)** As Long<br>Member of **UltraliteActiveX.ULConnection** |
| Description | Returns the number of rows that need to be uploaded when synchronization next takes place. |
| Parameters | **mask** An optional, unique identifier that refers to the publications to check. Use 0 for all publications. If not specified, then the value is zero.<br><br>**threshold** An optional parameter representing the maximum number of rows to count. Use -1 to indicate no maximum. If not specified, this value is -1. |
| Returns | Returns the number of rows that need to be uploaded in next synchronization. |

## GetNewUUID method

| | |
|---|---|
| Prototype | **GetNewUUID( )** As String<br>Member of **UltraliteActiveX.ULConnection** |
| Description | Returns a new universally unique identifier. The value is a string of the form |

*xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*, and is typically stored in a column of data type UNIQUEIDENTIFIER.

Returns — Each call returns a new UUID.

## GetTable method

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

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

Parameters — **name** The name of the table sought.

Returns — Returns the ULTable object.

## GrantConnectTo method

Prototype —

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

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

Parameters — **userid** The user ID being granted authority to connect.

**password** The password the user ID must specify to connect.

## LastDownloadTime method

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

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

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

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

## PrepareStatement method

Prototype — **PrepareStatement(** *sqlStatement* As String **)** As ULPreparedStatement
Member of **UltraliteActiveX.ULConnection**

Description — Prepares a SQL statement for execution.

Parameters — **sqlStatement** The SQL statement to prepare.

Returns

Returns a ULPreparedStatement. If there was a problem preparing the statement, an error will be raised. The offset into the statement where the error occurred can be determined from the SQLErrorOffset property.

## ResetLastDownloadTime method

Prototype

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

Description

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

Parameters

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

## RevokeConnectFrom method

Prototype

**RevokeConnectFrom(** *userID* As String **)**
Member of **UltraliteActiveX.ULConnection**

Description

Revokes the specified user’s ability to connect to the database.

Parameters

**userid** The user ID whose authority to connect is being revoked.

## Rollback method

Prototype

**Rollback( )**
Member of **UltraliteActiveX.ULConnection**

Description

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

## RollbackPartialDownload method

Roll back the changes from a failed synchronization.

Prototype

**RollbackPartialDownload ( )**
Member of **UltraliteActiveX.ULConnection**

Description

When a communication error occurs during the download phase of synchronization, UltraLite can apply the downloaded changes, so that the synchronization can be resumed from the place it was interrupted. If the download changes are not needed (the user or application does not want to resume the download at this point), RollbackPartialDownload rolls back the failed download transaction.

See also

- “Resuming failed downloads” [*MobiLink Administration Guide*, page 74]
- “Keep Partial Download synchronization parameter” [*MobiLink Clients*, page 321]

- ♦ "Partial Download Retained synchronization parameter" [*MobiLink Clients*, page 324]
- ♦ "Resume Partial Download synchronization parameter" [*MobiLink Clients*, page 327]

## StartSynchronizationDelete method

| | |
|---|---|
| Prototype | **StartSynchronizationDelete( )**<br>Member of **UltraliteActiveX.ULConnection** |
| Description | Once StartSynchronizationDelete is called, all delete operations are again synchronized. |

## StopSynchronizationDelete method

| | |
|---|---|
| Prototype | **StopSynchronizationDelete( )**<br>Member of **UltraliteActiveX.ULConnection** |
| Description | Prevents delete operations from being synchronized. This is useful for deleting old information from an UltraLite database to save space, while not deleting this information on the consolidated database. |

## StringToUUID method

Prototype

**StringToUUID(** *s_uuid* As String **)**
Member of **UltraliteActiveX.ULConnection**

Description

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

**Not needed in newer databases**
In databases created before version 9.0.2, the UNIQUEIDENTIFIER data type is defined as a user-defined data type and functions are needed to convert between binary and string representations of UUID values.

In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type is a native data type and UltraLite carries out conversions as needed. The StringToUUID function is therefore not needed.

☞ For more information, see "UNIQUEIDENTIFIER data type [Binary]" [*ASA SQL Reference*, page 75].

Parameters

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

Example

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

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

## Synchronize method

Prototype

**Synchronize(** [*show-progress* As Boolean ] **)**
Member of **UltraliteActiveX.ULConnection**

Description

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

Parameters

**show-progress** An optional parameter whose value may be true or false. Set this to true to show the progress of synchronization as it happens. Default is false.

## UUIDToString method

Prototype

**UUIDToString(** *buffer_16_bytes* **)** As String
Member of **UltraliteActiveX.ULConnection**

Description

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

**Not needed in newer databases**

In databases created before version 9.0.2, the UNIQUEIDENTIFIER data type is defined as a user-defined data type and functions are needed to convert between binary and string representations of UUID values.

In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type is a native data type and UltraLite carries out conversions as needed. The UUIDToString function is therefore not needed.

☞ For more information, see "UNIQUEIDENTIFIER data type [Binary]" [*ASA SQL Reference*, page 75].

Parameters

**buffer_16_bytes** An array of 16 bytes containing a UUID.

Returns

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

# ULConnectionParms class

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

## Properties

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

In UltraLite ActiveX, you can use the ULConnectionParms object and set your connection properties in your code. You use the ULConnectionParms object in conjunction with the **ULDatabaseManager.CreateDatabaseWithParms** and **ULDatabaseManager.OpenConnectionWithParms** methods.

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

☞ For more information about the meaning of these parameters, see "Connection Parameters" [*UltraLite Database User's Guide,* page 63].

| Prototype | Description |
|---|---|
| AdditionalParms As String (read-write) | Additional parameters specified as **name**=*value* pairs separated with semi-colons.<br><br>☞ See "Additional Parms connection parameter" [*UltraLite Database User's Guide,* page 68]. |
| CacheSize As String (read-write) | The size of the cache. CacheSize values are specified in bytes. Use the suffix k or K for kilobytes and use the suffix m or M for megabytes. The default cache size is sixteen pages. Given a default page size of 4 KB, the default cache size is 64 KB.<br><br>☞ See "Cache Size connection parameter " [*UltraLite Database User's Guide,* page 73]. |

| Prototype | Description |
|---|---|
| ConnectionName As String (read-write) | A name for the connection. This is needed only if you create more than one connection to the database.<br>☞ See "Connection Name connection parameter" [*UltraLite Database User's Guide*, page 74]. |
| DatabaseOnCE As String (read-write) | The filename of the database deployed to PocketPC.<br>☞ See "Database On CE connection parameter" [*UltraLite Database User's Guide*, page 69]. |
| DatabaseOnDesktop As String (read-write) | The filename of the database during development.<br>☞ See "Database On Desktop connection parameter" [*UltraLite Database User's Guide*, page 70]. |
| EncryptionKey As String (read-write) | A key for encrypting the database. OpenConnection and OpenConnectionWithParms must use the same key as specified during database creation. Suggestions for keys are:<br>1. Select an arbitrary, lengthy string<br>2. Select strings with a variety of numbers, letters and special characters, so as to decrease the chances of key penetration.<br>☞ See "Encryption Key connection parameter " [*UltraLite Database User's Guide*, page 75]. |
| ParmsUsed As String (read-only) | The parameters used by the ULDatabaseManager. Useful for debugging purposes. |
| Password As String (read-write) | The password for an authenticated user. Databases are initially created with one authenticated user password *SQL*. Passwords are case-insensitive if the database is case-insensitive and case-sensitive if the database is case-sensitive. The default value is *SQL*.<br>☞ See "Password connection parameter" [*UltraLite Database User's Guide*, page 76]. |

| Prototype | Description |
|---|---|
| ReserveSize As Integer (read-write) | The amount of file system space to reserve for storage of UltraLite persistent data.<br>☞ See "Reserve Size connection parameter " [*UltraLite Database User's Guide,* page 84]. |
| SchemaOnCE As String (read-write) | The schema filename deployed to PocketPC.<br>☞ See "Schema On CE connection parameter " [*UltraLite Database User's Guide,* page 78]. |
| SchemaOnDesktop As String (read-write) | The schema filename during development.<br>☞ See "Schema On Desktop connection parameter " [*UltraLite Database User's Guide,* page 79]. |
| UserID As String (read-write) | The authenticated user for the database. Databases are initially created with one authenticated user DBA. The UserID is case-insensitive if the database is case-insensitive and case-sensitive if the database is case-sensitive. The default value is *DBA*.<br>☞ See "User ID connection parameter" [*UltraLite Database User's Guide,* page 76]. |

## ULDatabaseManager class

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

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

### Properties

The following are properties of ULDatabaseManager:

| Prototype | Description |
|---|---|
| ErrorResume As Boolean | The error handling method. The default is false. If set to true, an error will not be raised when ULDatabase-Manager methods fail. |
| LastErrorCode As SQLCodeCon-stants | Gets the last error number, and allows you to clear the previous error code. |
| Version As String (read-only) | Gets the version string of the UltraLite component. |

### CreateDatabase method

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

Prototype

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

Description

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

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

☞ For more information about ApplyFile, see "ULDatabaseSchema class" on page 109 and "ApplyFile method" on page 110.

Parameters

**parms** A semicolon-separated list of database creation parameters.

☞ For information about connection parameters, see "Connection Parameters" [*UltraLite Database User's Guide,* page 63].

Returns

Returns a connection to a newly created UltraLite database.

Examples

The example below uses CreateObject to create and open a new database.

```
' eMbedded Visual Basic
open_parms = "file_name=\tutCustomer.udb"
schema_parms = open_parms & ";" & "schema_name=\tutCustomer.usm"
Set DatabaseMgr = CreateObject("UltraLite.ULDatabaseManager")
Set Connection = DatabaseMgr.CreateDatabase(schema_parms)
```

```
// JScript
open_parms = "file_name=\\tutCustomer.udb";
schema parms = open_parms + ";" + "schema_name=\\
         tutCustomer.usm"
DatabaseMgr = new ActiveXObject("UltraLite.ULDatabaseManager")
Set Connection = DatabaseMgr.CreateDatabase(schema_parms);
```

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

This functionality is only available with eMbedded Visual Basic.

☞ For more information about showing synchronization progress, see "Monitoring synchronization progress" on page 39.

```
'eMbedded Visual Basic
Set DBMgr =
         CreateObjectWithEvents("UltraLite.ULDatabaseManager",
         "UL_")
```

☞ For information about connection parameters, see "OpenConnection method" on page 107.

## CreateDatabaseWithParms method

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

Prototype

**CreateDatabaseWithParms(** *parms* As ULConnectionParms **)**
As ULConnection
Member of **UltraliteActiveX.ULDatabaseManager**

Description

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

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

Parameters

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

Returns

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

Examples

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

The example below uses CreateDatabaseWithParms to create and open a new database.

```
’ eMbedded Visual Basic
’ Use CreateObject in to get an instance of the
         ULDatabaseManager object
Set DatabaseMgr = CreateObject("UltraLite.ULDatabaseManager")
’ Use CreateObject to get an instance of the ULConnectionParms
         object
Dim LoginParms As ULConnectionParms
Set LoginParms = CreateObject("UltraLite.ULConnectionParms")
LoginParms.DatabaseOnCE = "/tutorial/tutorial.udb"
LoginParms.SchemaOnCE = "/tutorial/tutorial.usm"
’ Drop the existing database and create a new database
Call DatabaseMgr.DropDatabaseWithParms( LoginParms )
Set Connection = DatabaseMgr.CreateDatabaseWithParms(LoginParms)
```

```
// JScript
’ get an instance of the ULDatabaseManager object
DatabaseMgr = new ActiveXObject("UltraLite.ULDatabaseManager");
var LoginParms;
LoginParms = new ActiveXObject("UltraLite.ULConnectionParms");
LoginParms.DatabaseOnCE = "//tutorial//tutorial.udb";
LoginParms.SchemaOnCE = "//tutorial//tutorial.usm";
’ Drop the existing database and create a new database
DatabaseMgr.DropDatabaseWithParms( LoginParms );
Connection = DatabaseMgr.CreateDatabaseWithParms( LoginParms );
```

## DropDatabase method

The DropDatabase method deletes a database file.

Prototype

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

Description

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

Parameters

**parms** The filename for the database.

Example

The following example drops a database:

```
' eMbedded Visual Basic
open_parms = "ce_file=\tutCustomer.udb"
DropDatabase(open_parms)
```

```
// JScript
open_parms = "ce_file=\\tutCustomer.udb";
DropDatabase( open_parms );
```

## DropDatabaseWithParms method

The DropDatabaseWithParms method deletes a database file.

Prototype

**DropDatabaseWithParms(** *parms* As ULConnectionParms **)**
Member of **UltraliteActiveX.ULDatabaseManager**

Description

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

Parameters

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

Example

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

```
' eMbedded Visual Basic
Call DatabaseMgr.DropDatabaseWithParms( LoginParms )
```

```
// JScript
DatabaseMgr.DropDatabseWithParms( LoginParms );
```

## OnReceive event

Prototype

**OnReceive(**
  *nBytes* As Long**,** _
  *nInserts* As Long**,** _
  *nUpdates* As Long**,** _
  *nDeletes* As Long _
**)**
Member of **UltraliteActiveX.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

```
’eMbedded Visual Basic
Private Sub OnReceive(ByVal nBytes As Long, ByVal nInserts As
         Long, ByVal nUpdates As Long, ByVal nDeletes As Long)
    prLine "OnReceive " & nBytes & " bytes, " & Inserts & "
         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 **UltraliteActiveX.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

```
' eMbedded Visual Basic
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 **UltraliteActiveX.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

```
' eMbedded Visual Basic
Private Sub _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 **UltraliteActiveX.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 ULDatabaseSchema.GetTableName method.

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

Example

```
' eMbedded Visual Basic
Private Sub _OnTableChange(ByVal newTableIndex As Long, ByVal
         numTables As Long)
    prLine "OnTableChange index:" & newTableIndex & ", #tables="
         & numTables
End Sub
```

## OpenConnection method

Prototype

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

Description

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

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

Parameters

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

Returns

The ULConnection object is returned if the connection was successful.

Example

The example below shows how to use connection parameters in the OpenConnection method.

```
' eMbedded Visual Basic
open_parms = "ce_file = \tutCustomer.udb"
Set DatabaseMgr = CreateObject("UltraLite.ULDatabaseManager")
Set Connection = DatabaseMgr.OpenConnection(open_parms)
```

```
// JScript
open_parms = "ce_file = \\tutCustomer.udb";
DatabaseMgr = new ActiveXObject("UltraLite.ULDatabaseManager");
Connection = DatabaseMgr.OpenConnection(open_parms);
```

## OpenConnectionWithParms method

Prototype

**OpenConnectionWithParms(** *connparms* As ULConnectionParms**)**
As ULConnection
Member of **UltraliteActiveX.ULDatabaseManager**

Description

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

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

Parameters

**connparms** The parameters defining this connection.

Returns

The ULConnection object is returned if the connection was successful.

Example

The following example assumes you have created a ULConnectionParms object **LoginParms** and have specified the database locations and schema locations.

```
' eMbedded Visual Basic
Set DatabaseMgr = CreateObject("UltraLite.ULDatabaseManager")
Set Connection = DatabaseMgr.OpenConnection(LoginParms)
```

```
// JScript
DatabaseMgr = new ActiveXObject("UltraLite.ULDatabaseManager");
Connection = DatabaseMgr.OpenConnection(LoginParms);
```

## ULDatabaseSchema class

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

### Properties

The following are properties of ULDatabaseSchema:

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

| **Prototype** | **Description** |
|---|---|
| TimestampFormat As String (read-only) | Gets the format for timestamps retrieved from the database. |
| TimestampIncrement (read-only) | Gets the database timestamp increment. |

## ApplyFile method

Prototype

**ApplyFile(** *parms* As String **)** As Boolean
Member of **UltraliteActiveX.ULDatabaseSchema**

Description

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

**Caution**
Although ApplyFile is very safe, ApplyFile can cause data loss under a number of circumstances including (1) if columns are deleted, or (2) if the data type for a column is changed to an incompatible type, or (3) if you upgrade an 8.0.2 database using ApplyFile in UltraLite 9.0.

Parameters

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

Returns

**True** if successful.

**False** if unsuccessful.

Example

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

## ApplyFileWithParms method

Prototype

**ApplyFileWithParms(** *parms* As ULConnectionParms ) As Boolean
Member of **UltraliteActiveX.ULDatabaseSchema**

Description

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

**Caution**
Although ApplyFile is very safe, ApplyFileWithParms can cause data loss under a number of circumstances including (1) if columns are deleted, or (2) if the data type for a column is changed to an incompatible type, or (3) if you upgrade an 8.0.2 database using ApplyFile in UltraLite 9.0.

Parameters

**parms** The object identifying the schema file to apply.

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) | Gets the number of columns in the index |
| ColumnName (position As Long) As String (read-only) | Gets the column name in position of index. |
| ForeignKey As Boolean (read-only) | Indicates whether this is a foreign key. |
| IsColumnDescending( position As Long) As Boolean (read-only) | Indicates whether a column is sorted descending. False if ascending. |
| Name As String (read-only) | Gets the name of the index |
| PrimaryKey As Boolean (read-only) | Gets whether this is the primary key for this table. |
| ReferencedIndexName As String (read-only) | Gets the name of the index referenced by this index if it is a foreign key |
| ReferencedTableName As String (read-only) | Gets the name of the table referenced by this index if it is a foreign key |
| UniqueIndex As Boolean (read-only) | Indicates whether values in the index must be unique. |
| UniqueKey As Boolean (read-only) | Indicates whether the index is a unique constraint on a table. If True, the columns in the index are unique and do not permit NL values |

## ULPreparedStatement class

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

### Properties

| Prototype | Description |
|---|---|
| HasResultSet As Boolean (read-only) | Indicates whether the prepared statement generates a result set.<br>True if the statement has a result set, otherwise, false.<br>If true, ExecuteQuery should be called instead of ExecuteStatement. |
| Schema As ULResultSetSchema (read-only) | Gets the schema describing results of statement. |

### AppendByteChunk method

Prototype

**AppendByteChunk( _**
  *parameter_id* As Long**,** _
  *Array***,** _
  [ *chunkSize* ] _
**)** As Boolean
Member of **UltraliteActiveX.ULPreparedStatement**

Description

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

Parameters

**parameter_id** The 1-based parameter number to set.

**data** The array of bytes to append.

**chunkSize** The number of bytes to append. If not provided, the length of byteArray is used.

Returns

**True** if successful.

|  |  |
|---|---|
|  | **False** if unsuccessful. |
| Errors set | **ulSQLE_CONVERSION_ERROR** The error occurs if the column data type is not LONG BINARY or BINARY. |

## AppendStringChunk method

| | |
|---|---|
| Prototype | **AppendStringChunk(**<br>*parameter_id* As Long **,**<br>*chunk* **)**<br>Member of **UltraLiteActiveX.ULPreparedStatement** |
| Description | Appends the string to the column if the type is ulTypeLongString. |
| Parameters | **parameter_id** The 1-based parameter number to set. |
|  | **chunk** A string to append to the existing string in a table. |
| Errors set | **ulSQLE_CONVERSION_ERROR** The error occurs if the column data type is not VARCHAR. |

## Close method

| | |
|---|---|
| Prototype | **Close( )**<br>Member of **UltraLiteActiveX.ULPreparedStatement** |
| Description | Frees resources associated with the ULPreparedStatement. |

## ExecuteQuery method

| | |
|---|---|
| Prototype | **ExecuteQuery( )** As ULResultSet<br>Member of **UltraliteActiveX.ULPreparedStatement** |
| Description | Executes the query and returns a result set. |
| Returns | A ULResultSet object. The ULResultSet is the data you requested in your SELECT statement. To describe the product of your query, see "ULResultSetSchema class" on page 122 |

## ExecuteStatement method

| | |
|---|---|
| Prototype | **ExecuteStatement( )** As Long<br>Member of **UltraliteActiveX.ULPreparedStatement** |
| Description | Executes the statement. |
| Returns | The number of rows updated. |

## SetNullParameter method

| | |
|---|---|
| Prototype | **SetNullParameter(** *parameter_id* As Long **)**<br>Member of **UltraliteActiveX.ULPreparedStatement** |
| Description | Set the parameter to NL. |
| Parameters | **parameter_id** The 1-based parameter number to set. |

## SetParameter method

| | |
|---|---|
| Prototype | **SetParameter(**<br>*parameter_id* As Long<br>*val*<br>**)**<br>Member of **UltraliteActiveX.ULPreparedStatement** |
| Description | Set execution parameter to the value passed in. |
| Parameters | **parameter_id** The 1-based parameter number to set.<br><br>**val** The value you want for the execution parameter. |

# ULPublicationSchema class

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

## Properties

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

## ULResultSet class

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

### Properties

| Prototype | Description |
|---|---|
| BOF As Boolean (read-only) | Indicates whether the current row position is before the first row. Returns True if the current row position is before the first row, otherwise false. |
| EOF As Boolean (read-only) | Indicates whether the current row position is after the last row. EOF is true if beyond the last row, otherwise false. |
| IsNull (columnID As Long) As Boolean (read-only) | Indicates whether the value from the specified column is SQL NULL. True if the column is null, else, IsNull is false. |
| RowCount As Long (read-only) | The number of rows in the result set. |
| Schema As ULResultSetSchema (read-only) | The schema description for this result set. |
| Value(columnID As Long) (read-only) | The value of the given column |

### Close method

Prototype

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

Description

Frees all resources associated with this object.

## GetByteChunk method

Prototype

**GetByteChunk(**
*index* As Long**,** _
*offset* As Long**,** _
*data***,** _
[ *data_len* As Long ] _
**)** As Long
Member of **UltraliteActiveX.ULResultSet**

Description

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

Parameters

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

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

**data** A pointer to an array of bytes.

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

Returns

The number of bytes read.

Errors set

**ulSQLE_CONVERSION_ERROR** The error occurs if the column data type is not BINARY or LONG BINARY.

**ulSQLE_INVALID_PARAMETER** The error occurs if the column data type is BINARY and the offset is not 0 or 1, or, the data length is less than 0.

The error also occurs if the column data type is LONG BINARY and the offset is less than 1.

Example

In the following example, edata is a column name.

```
’ eMbedded Visual Basic
Dim data (512) As Byte
...
table.Columns.Item("edata").GetByteChunk(0,data)
```

```
// JScript
var data = new Array();
// ...
table.Columns.Item("edata").GetByteChunk(0,data);
```

## GetStringChunk method

| | |
|---|---|
| Prototype | **GetStringChunk(**<br>*index* As Long**,** _<br>*offset* As Long**,** _<br>*pStringObj***,** _<br>[ *chunkSize* ] **)** As Long<br>Member of **UltraliteActiveX.ULResultSet** |
| Description | Fills the string passed in with the binary data in the column. Suitable for Long Varchars. |
| Parameters | **index** The 1-based column ID of the target column.<br><br>**offset** The character offset into the underlying data from which you start getting the string.<br><br>**pStringObj** The string you want returned. This variant is passed by reference.<br><br>**chunkSize** An optional parameter representing the number of characters to retrieve. |
| Returns | The number of characters copied. Room is left for a null termination character and the length does not include that character.<br><br>Gets BLOB data from a binary or long binary column. |
| Errors set | **ulSQLE_CONVERSION_ERROR** The error occurs if the column data type is not CHAR or LONG VARCHAR.<br><br>**ulSQLE_INVALID_PARAMETER** The error occurs if the column data type is CHAR and the src_offset is greater than 64K.<br><br>The error occurs if offset is less than 1 or string length is less than 0. |

## MoveAfterLast method

| | |
|---|---|
| Prototype | **MoveAfterLast( )**<br>Member of **UltraliteActiveX.ULResultSet** |
| Description | Moves to a position after the last row of the ULResultSet. |
| Returns | **True** if successful.<br><br>**False** if unsuccessful. The method fails, for example, if there are no rows. |

## MoveBeforeFirst method

| | |
|---|---|
| Prototype | **MoveBeforeFirst( )**<br>Member of **UltraliteActiveX.ULResultSet** |

| | |
|---|---|
| Description | Moves to a position before the first row. |
| Returns | **True** if successful. |
| | **False** if unsuccessful. The method fails, for example, if there are no rows. |

## MoveFirst method

| | |
|---|---|
| Prototype | **MoveFirst( )** As Boolean<br>Member of **UltraliteActiveX.ULResultSet** |
| Description | Moves to the first row. |
| Returns | **True** if successful. |
| | **False** if unsuccessful. The method fails, for example, if there are no rows. |

## MoveLast method

| | |
|---|---|
| Prototype | **MoveLast( )**<br>Member of **UltraliteActiveX.ULResultSet** |
| Description | Moves to the last row. |
| Returns | **True** if successful. |
| | **False** if unsuccessful. The method fails, for example, if there are no rows. |

## MoveNext method

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

## MovePrevious method

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

## MoveRelative method

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

## IsNull method

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

## ULResultSetSchema class

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

### Properties

| Prototype | Description |
|---|---|
| ColumnCount As Long(read-only) | Gets the number of columns in the result set |
| ColumnName As String (read-only) | Gets the name of the column in the result set. |
| ColumnPrecision As Integer (read-only) | Gets the precision of the datatype for the column if it is numeric. |
| ColumnScale As Integer (read-only) | Gets the scale of the datatype for the column if it is numeric. |
| ColumnSize As Integer (read-only) | Gets the size of the datatype for the column. |
| ColumnSQLType As ULSQL-Type (read-only) | Gets the ULSQLType of the column. |
| Name (column_id As Long) as String (read-only) | Gets the name of the publicationgiven the column name from the ordinal ID |
| Precision(columnID As Long) As Long (read-only) | Gets the numeric precision of the column. |

### GetColumnID method

Prototype

**GetColumnID(** *col_name* As String **)** As Long
Member of **UltraliteActiveX.ULResultSetSchema**

Description

Get the column id for a named column.

Parameters

**col_name** The column name for which an id is sought.

Returns

GetColumnID returns the column ID for the named column.

## ULSQLCode enumeration

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

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

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

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

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

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

## ULSQLType enumeration

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

| Constant | UltraLite Database Type | Value |
|---|---|---|
| ulTypeLong | Integer | 1 |
| ulTypeUnsignedLong | SmallInt | 2 |
| ulTypeShort | UnsignedInteger | 3 |
| ulTypeUnsignedShort | UnsignedSmallInt | 4 |
| ulTypeBig | Big | 5 |
| ulTypeUnsignedBig | UnsignedBig | 6 |
| ulTypeByte | Byte | 7 |
| ulTypeBit | Bit | 8 |
| ulTypeDateTime | Time | 9 |
| ulTypeDate | Date | 10 |
| ulTypeTime | Timestamp | 11 |
| ulTypeDouble | Double | 12 |
| ulTypeReal | Real | 13 |
| ulTypeNumeric | (Var)Binary | 14 |
| ulTypeBinary | LongBinary | 15 |
| ulTypeString | (Var)Char | 16 |
| ulTypeLongString | LongVarchar | 17 |
| ulTypeLongBinary | Numeric | 18 |

## ULStreamErrorCode enumeration

The ULStreamErrorCode constants identify communications errors during synchronization.

☞ For more information about these errors, see "MobiLink Communication Error Messages" [*ASA Error Messages*, page 549].

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

| Constant | Value |
|---|---|
| ulStreamErrorCodeSecureCertificateRef | 23 |
| ulStreamErrorCodeSecureCertificateNotTrusted | 24 |
| ulStreamErrorCodeSecureDuplicateContext | 25 |
| ulStreamErrorCodeSecureSetIo | 26 |
| ulStreamErrorCodeSecureSetIoSemantics | 27 |
| ulStreamErrorCodeSecureCertificateChainFunc | 28 |
| ulStreamErrorCodeSecureCertificateChainRef | 29 |
| ulStreamErrorCodeSecureEnableNonBlocking | 30 |
| ulStreamErrorCodeSecureSetCipherSuites | 31 |
| ulStreamErrorCodeSecureSetChainNumber | 32 |
| ulStreamErrorCodeSecureCertificateFileNotFound | 33 |
| ulStreamErrorCodeSecureReadCertificate | 34 |
| ulStreamErrorCodeSecureReadPrivateKey | 35 |
| ulStreamErrorCodeSecureSetPrivateKey | 36 |
| ulStreamErrorCodeSecureCertificateExpiryDate | 37 |
| ulStreamErrorCodeSecureExportCertificate | 38 |
| ulStreamErrorCodeSecureAddCertificate | 39 |
| ulStreamErrorCodeSecureTrustedCertificateFileNotFound | 40 |
| ulStreamErrorCodeSecureTrustedCertificateRead | 41 |
| ulStreamErrorCodeSecureCertificateCount | 42 |
| ulStreamErrorCodeSecureCreateCertificate | 43 |
| ulStreamErrorCodeSecureImportCertificate | 44 |
| ulStreamErrorCodeSecureSetRandomRef | 45 |
| ulStreamErrorCodeSecureSetRandomFunc | 46 |
| ulStreamErrorCodeSecureSetProtocolSide | 47 |
| ulStreamErrorCodeSecureAddTrustedCertificate | 48 |
| ulStreamErrorCodeSecureCreatePrivateKeyObject | 49 |
| ulStreamErrorCodeSecureCertificateExpired | 50 |

| Constant | Value |
|---|---|
| ulStreamErrorCodeSecureCertificateCompanyUnit | 51 |
| ulStreamErrorCodeSecureCertificateCommonName | 52 |
| ulStreamErrorCodeSecureHandshake | 53 |
| ulStreamErrorCodeHttpVersion | 54 |
| ulStreamErrorCodeSecureSetReadFunc | 55 |
| ulStreamErrorCodeSecureSetWriteFunc | 56 |
| ulStreamErrorCodeSocketHostNameNotFound | 57 |
| ulStreamErrorCodeSocketGetHostByAddr | 58 |
| ulStreamErrorCodeSocketLocalhostNameNotFound | 59 |
| ulStreamErrorCodeSocketCreateTcpip | 60 |
| ulStreamErrorCodeSocketCreateUdp | 61 |
| ulStreamErrorCodeSocketBind | 62 |
| ulStreamErrorCodeSocketCleanup | 63 |
| ulStreamErrorCodeSocketClose | 64 |
| ulStreamErrorCodeSocketConnect | 65 |
| ulStreamErrorCodeSocketGetName | 66 |
| ulStreamErrorCodeSocketGetOption | 67 |
| ulStreamErrorCodeSocketSetOption | 68 |
| ulStreamErrorCodeSocketListen | 69 |
| ulStreamErrorCodeSocketShutdown | 70 |
| ulStreamErrorCodeSocketSelect | 71 |
| ulStreamErrorCodeSocketStartup | 72 |
| ulStreamErrorCodeSocketPortOutOfRange | 73 |
| ulStreamErrorCodeLoadNetworkLibrary | 74 |
| ulStreamErrorCodeActsyncNoPort | 75 |
| ulStreamErrorCodeHttpExpectedPost | 89 |

## ULStreamErrorContext enumeration

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

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

## ULStreamErrorID enumeration

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

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

## ULStreamType enumeration

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

| Constant | Value | Description |
|---|---|---|
| ulTCPIP | 1 | TCP/IP stream |
| ulHTTP | 2 | HTTP stream |
| ulHTTPS | 3 | HTTPS synchronization |

## ULSyncParms class

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

### Properties

The following are properties of ULSyncParms:

| Prototype | Description |
|---|---|
| CheckpointStore As Boolean | If true, adds checkpoints of the database during synchronization to limit database growth during the synchronization process. This is most useful for large downloads with many updates.<br>See "Checkpoint Store synchronization parameter" [*MobiLink Clients,* page 318]. |
| DownloadOnly As Boolean | Indicates if a synchronization only downloads data.<br>See "Download Only synchronization parameter" [*MobiLink Clients,* page 320]. |
| KeepPartialDownload As Boolean | If the synchronisation fails during download because of a communications error, apply those changes that were successfully downloaded, rather than rolling back all the changes.<br>See "Keep Partial Download synchronization parameter" [*MobiLink Clients,* page 321]. |
| NewPassword As String | Change a user password to this new password string on the next synchronization.<br>See "New Password synchronization parameter" [*MobiLink Clients,* page 322]. |
| Password As String | The password corresponding to a given user name.<br>See "Password synchronization parameter" [*MobiLink Clients,* page 324]. |

| Prototype | Description |
|---|---|
| PublicationMask As Long | Specify the publications to synchronize. The default is to synchronize all data.<br>See "Publication synchronization parameter" [*MobiLink Clients*, page 326]. |
| ResumePartialDownload As Boolean | Resume a synchronization that failed during download because of a communications error, applying only those changes that were scheduled to be downloaded in the failed synchronization.<br>See "Resume Partial Download synchronization parameter" [*MobiLink Clients*, page 327]. |
| SendColumnNames As Boolean | If SendColumnNames is true, column names are sent to the MobiLink synchronization server. Column names must be sent to the MobiLink synchronization server for automatic script generation.<br>See "Send Column Names synchronization parameter" [*MobiLink Clients*, page 330]. |
| SendDownloadAck As Boolean | If SendDownloadAck is true, a download acknowledgement is sent during synchronization.<br>See "Send Download Acknowledgement synchronization parameter" [*MobiLink Clients*, page 331]. |
| Stream As ULStreamType constants | Set the type of stream to use during synchronization.<br>See "Stream Type synchronization parameter" [*MobiLink Clients*, page 332]. |
| StreamParms As String | Set network protocol options for the given stream type.<br>See "Stream Parameters synchronization parameter" [*MobiLink Clients*, page 334] and "Network protocol options for UltraLite synchronization clients" [*MobiLink Clients*, page 341]. |

| Prototype | Description |
|---|---|
| UploadOnly As Boolean | Indicates whether a synchronization only uploads data.<br>See "Upload Only synchronization parameter" [*MobiLink Clients*, page 337]. |
| UserName As String | The MobiLink user name for synchronization.<br>See "User Name synchronization parameter" [*MobiLink Clients*, page 338]. |
| Version As String | The synchronization script version to run.<br>See "Version synchronization parameter" [*MobiLink Clients*, page 339]. |

## AddAuthenticationParm method

| | |
|---|---|
| Prototype | **AddAuthenticationParm(** BSTR parm **)**<br>Member of **UltraliteActiveX.ULSyncParms** |
| Description | Adds a parameter to be passed to the authenticate_parms MobiLink synchronization script. |
| Parameters | **parm** The parameter being added. |
| Returns | No return value. |
| See also | "Authentication Parameters synchronization parameter" [*MobiLink Clients*, page 316]<br>"authenticate_parameters connection event" [*MobiLink Administration Guide*, page 334] |

## ClearAuthenticationParms method

| | |
|---|---|
| Prototype | **ClearAuthenticationParms( )**<br>Member of **UltraliteActiveX.ULSyncParms** |
| Description | Clears all parameters that were to be passed to the authenticate_parms MobiLink synchronization script. |
| Returns | No return value. |
| See also | "Authentication Parameters synchronization parameter" [*MobiLink Clients*, page 316] |

“authenticate_parameters connection event” [*MobiLink Administration Guide,* page 334]

# ULSyncResult class

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

## Properties

The following are properties of ULSyncResult:

| Prototype | Description |
|---|---|
| AuthStatus As AuthStatusCode (read-only) | Gets the authorization status code for the last synchronization.<br>See "Authentication Status synchronization parameter" [*MobiLink Clients,* page 317]. |
| AuthValue As Long (read-only) | Gets the MobiLink authentication value.<br>See "Authentication Value synchronization parameter" [*MobiLink Clients,* page 318]. |
| PartialDownloadRetained (read-only) | Indicates that the synchronization failed during download, and that a partial download was kept.<br>See "Partial Download Retained synchronization parameter" [*MobiLink Clients,* page 324]. |
| IgnoredRows As Boolean (read-only) | Indicates whether rows were ignored during the last synchronization.<br>See "Ignored Rows synchronization parameter" [*MobiLink Clients,* page 320]. |
| StreamErrorCode As ULStreamErrorCode (read-only) | Gets the error code reported by the synchronization stream. |
| StreamErrorContext As ULStreamErrorContext (read-only) | Gets the basic network operation performed. |
| StreamErrorID As ULStreamErrorID (read-only) | Gets the network layer reporting the error. |
| StreamErrorSystem As Long (read-only) | Gets the stream error system-specific code. |

| Prototype | Description |
|---|---|
| Timestamp as Variant (read-only) | Gets the timestamp of the last synchronization. |
| UploadOK As Boolean (read-only) | Indicates whether data was uploaded successfully in the last synchronization.<br>See “Version synchronization parameter” [*MobiLink Clients,* page 339]. |

## ULSyncState enumeration

| Constant | Description |
|---|---|
| ulSyncStateStarting | No synchronization actions have been taken yet. |
| ulSyncStateConnecting | The synchronization stream has been built, but not yet opened. |
| ulSyncStateSendingHeader | The synchronization stream has been opened and the header is about to be sent. |
| ulSyncStateSendingTable | A table is being sent. |
| ulSyncStateSendingData | Data for the current table is being sent. |
| ulSyncStateFinishingUpload | The upload is completing. The final count of rows sent is included with this event. |
| ulSyncStateReceivingUploadAck | An acknowledgement that the upload is complete is being received. |
| ulSyncStateReceivingTable | A table is being received. |
| ulSyncStateReceivingData | Data for the current table is being received. |
| ulSyncStateCommittingDownload | The download is being committed. The final count of rows received is included with this event. |
| ulSyncStateSendingDownloadAck | An acknowledgement that the download is complete is being sent. |
| ulSyncStateDisconnecting | The synchronization stream is about to be closed. |
| ulSyncStateDone | Synchronization has successfully completed. The SyncResult object has been updated. |
| ulSyncStateError | Synchronization has completed but an error occurred. Check SyncResult and SQLCode for details. |

| Constant | Description |
|---|---|
| ulSyncStateRollingBackDownload | Synchronization is rolling back the download because an error was encountered during the download. The error will be reported with a subsequent ulSyncStateError progress report. |
| ulSyncStateCancelled | Synchronization has been canceled. |

## ULTable class

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

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

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

### Properties

| Prototype | Description |
|---|---|
| BOF As Boolean (read-only) | Indicates whether the current row position is before the first row. Returns True if the current row position is before the first row, otherwise false. |
| Columns As IColumns (read-only) | Gets a collection of column objects |
| EOF As Boolean (read-only) | Indicates whether the current row position is after the last row. Returns True if the current row position is before the first row, otherwise false. |
| IsOpen As Boolean (read-only) | Indicates whether or not the table is currently open. |
| RowCount As Long (read-only) | Gets the number of rows in the table. |
| Schema As ULTableSchema (read-only) | Gets information about the table schema. |

### Close method

Prototype

**Close( )**
Member of **UltraliteActiveX.ULTable**

Description

Frees resources associated with the table. This method should be called after all processing involving the table is complete.

## Delete method

Prototype

**Delete( )**
Member of **UltraliteActiveX.ULTable**

Description

Deletes the current row from the table.

## DeleteAllRows method

Prototype

**DeleteAllRows( )**
Member of **UltraliteActiveX.ULTable**

Description

Deletes all rows in the table.

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

## FindBegin method

Prototype

**FindBegin( )**
Member of **UltraliteActiveX.ULTable**

Description

Prepares a table for a find.

## FindFirst method

Prototype

**FindFirst(** [*num_columns* As Long = 32767] **)** As Boolean
Member of **UltraliteActiveX.ULTable**

Description

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

The current index is that used to specify the sort order of the table. It is specified when your application calls the Open method. The default index is the primary key.

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

*Note* : Requires that FindBegin be called prior to using this method.

Parameters

**num_columns** An optional parameter referring to the number of columns to be used in the FindFirst. For example, if 2 is passed, the first two columns are used for the FindFirst. If num_columns exceeds the number of columns indexed, all columns are used in FindFirst.

Returns

**True** if successful.

**False** if unsuccessful.

## FindLast method

Prototype

**FindLast(** [ *num_columns* As Long = 32767 ] **)** As Boolean
Member of **UltraliteActiveX.ULTable**

Description

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

The current index is used to specify the sort order of the table. It is specified when your application calls the **Open** method. The default index is the primary key.

☞ For more information, see "Open method" on page 149.

To specify the value to search for, set the column value for each column in the index for which you want to find the value. The cursor is left on the last row found that exactly matches the index value. On failure the cursor position is before the first row (**BOF)**.

**Note**
Requires that FindBegin be called prior to using this method.

Parameters

**num_columns** An optional parameter referring to the number of columns to be used in the FindLast. For example, if 2 is passed, the first two columns are used for the FindLast. If num_columns exceeds the number of columns indexed, all columns are used in FindLast.

Returns

**True** if successful.

**False** if unsuccessful.

## FindNext method

Prototype

**FindNext(** [ *num_columns* As Long = 32767 ] **)** As Boolean
Member of **UltraliteActiveX.ULTable**

Description

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

The current index is that used to specify the sort order of the table. It is specified when your application calls the **Open** method. The default index is the primary key.

☞ For more information, see "Open method" on page 149.

The cursor is left on the first row found that exactly matches the index value. On failure, the cursor position is after the last row (**EOF)**.

*Note* : Must be preceded by FindFirst or FindLast.

Parameters **num_columns** An optional parameter referring to the number of columns to be used in the FindNext. For example, if 2 is passed, the first two columns are used for the FindNext. If num_columns exceeds the number of columns indexed, all columns are used in FindNext.

Returns **True** if successful.

**False** if unsuccessful (EOF).

## FindPrevious method

Prototype **FindPrevious(** [ *num_columns* As Long = 32767 ] **)** As Boolean
Member of **UltraliteActiveX.ULTable**

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

The current index is that used to specify the sort order of the table. It is specified when your application calls the **Open** method. The default index is the primary key.

☞ For more information, see "Open method" on page 149.

On failure it is positioned before the first row (**BOF)**.

Parameters **num_columns** An optional parameter referring to the number of columns to be used in the FindPrevious. For example, if 2 is passed, the first two columns are used for the FindPrevious. If num_columns exceeds the number of columns indexed, all columns are used in FindPrevious.

Returns **True** if successful.

**False** if unsuccessful (BOF).

## Insert method

Prototype **Insert( )** As Boolean
Member of **UltraliteActiveX.ULTable**

Description Inserts a row in the table with values specified in previous **Set** methods. Must be preceded by **InsertBegin**. Set for each ULColumn object.

Returns **True** if successful.

**False** if unsuccessful (BOF).

## InsertBegin method

Prototype

**InsertBegin( )**
Member of **UltraliteActiveX.ULTable**

Description

Prepares a table for inserting a new row, setting column values to their defaults.

Examples

In this example, InsertBegin sets insert mode to allow you to begin assigning data values to CustomerTable columns.

```
' eMbedded Visual Basic
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
```

```
// JScript
CustomerTable.InsertBegin();
CustomerTable.Columns("Fname").Value = fname;
CustomerTable.Columns("Lname").Value = lname;
If ( Len(city) > 0 ) {
  CustomerTable.Columns("City").Value = city;
}
If ( Len(phone) > 0 ) {
  CustomerTable.Columns("phone").Value = phone;
}
CustomerTable.Insert();
```

See also

"UpdateBegin method" on page 150

## LookupBackward method

Prototype

**LookupBackward(** [ *num_columns* As Long = 32767 ] **)** As Boolean
Member of **UltraliteActiveX.ULTable**

Description

Move backwards through the table starting from the end, looking for the first row that matches or is less than a value or set of values in the current index.

The current index is that used to specify the sort order of the table. It is specified when your application calls the **Open** method. The default index is the primary key.

☞ For more information, see "Open method" on page 149.

To specify the value to search for, set the column value for each column in the index. The cursor is left on the last row that matches or is less than the index value. On failure (that is, if no row is less than the value being looked for), the cursor position is before the first row (**BOF)**.

Parameters **num_columns** For composite indexes, the number of columns to use in the lookup.

Returns **True** if successful.

**False** if unsuccessful.

## LookupBegin method

Prototype **LookupBegin( )**
Member of **UltraliteActiveX.ULTable**

Description Prepares a table for a lookup.

## LookupForward method

Prototype **LookupForward(** [*num_columns* As Long = 32767 ] **)** As Boolean
Member of **UltraliteActiveX.ULTable**

Description Move forward through the table starting from the beginning, looking for the first row that matches or is greater than a value or set of values in the current index.

The current index is that used to specify the sort order of the table. It is specified when your application calls the **Open** method. The default index is the primary key.

☞ For more information, see "Open method" on page 149.

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure (that is, if no rows are greater than the value being looked for), the cursor position is after the last row (**EOF**).

Parameters **num_columns** For composite indexes, the number of columns to use in the lookup.

Returns **True** if successful.

**False** if unsuccessful.

## MoveAfterLast method

Prototype **MoveAfterLast( )** As Boolean
Member of **UltraliteActiveX.ULTable**

Description Moves to a position after the last row.

| | |
|---|---|
| Returns | **True** if successful. |
| | **False** if the operation fails. |

## MoveBeforeFirst method

| | |
|---|---|
| Prototype | **MoveBeforeFirst( )** As Boolean<br>Member of **UltraliteActiveX.ULTable** |
| Description | Moves to a position before the first row. |
| Returns | **True** if successful. |
| | **False** if the operation fails. |

## MoveFirst method

| | |
|---|---|
| Prototype | **MoveFirst( )** As Boolean<br>Member of **UltraliteActiveX.ULTable** |
| Description | Moves to the first row. |
| Returns | **True** if successful. |
| | **False** if there is no data in the table. |

## MoveLast method

| | |
|---|---|
| Prototype | **MoveLast( )** As Boolean<br>Member of **UltraliteActiveX.ULTable** |
| Description | Moves to the last row. |
| Returns | **True** if successful. |
| | **False** if there is no data in the table. |

## MoveNext method

| | |
|---|---|
| Prototype | **MoveNext( )** As Boolean<br>Member of **UltraliteActiveX.ULTable** |
| Description | Moves to the next row. |
| Returns | **True** if successful. |
| | **False** if there is no more data in the table. For example, MoveNext fails if there are no more rows. |

## MovePrevious method

| | |
|---|---|
| Prototype | **MovePrevious( )** As Boolean<br>Member of **UltraliteActiveX.ULTable** |

Description

Moves to the previous row.

Returns

**True** if successful.

**False** if there is no more data in the table. For example, MovePrevious fails if there are no rows.

## MoveRelative method

Prototype

**MoveRelative(** *index* As Long **)** As Boolean
Member of **UltraliteActiveX.ULTable**

Description

Moves a certain number of rows relative to the current row.

Parameters

**index** The number of rows to move. The value can be positive, negative, or zero. Zero is useful if you want to repopulate a row buffer.

Returns

**True** if successful.

**False** if the move failed, as may happen, for example, if the cursor is positioned beyond the first or last row.

## Open method

Prototype

**Open(**
 [ *index_id* ], **_)**
Member of **UltraliteActiveX.ULTable**

Description

Opens the table so it can be read or manipulated. By default, the rows are ordered by primary key. By supplying an index , the rows can be ordered in other ways.

The cursor is positioned before the first row in the table.

Parameters

**indexID** An optional parameter referring to the ID of the index.

## Truncate method

Prototype

**Truncate( )**
Member of **UltraliteActiveX.ULTable**

Description

Removes all data from this table. The changes are not synchronized, so that on synchronization, it does not affect the data in the consolidated database.

☞ For more information, see "StartSynchronizationDelete method" on page 96.

## Update method

Prototype

**Update( )**
Member of **UltraliteActiveX.ULTable**

Description

Updates a row in the table with values specified in **ULColumn** methods.

*Note* : Must be preceded by a call to UpdateBegin.

## UpdateBegin method

Prototype

**UpdateBegin( )**
Member of **UltraliteActiveX.ULTable**

Description

Prepares a table for modifying the contents of the current row.

Example

```
' eMbedded Visual Basic
Table.UpdateBegin
Table.Columns( "ColName" ).Value = "New Value"
Table.Update
```

```
  // JScript
  Table.UpdateBegin();
  Table.Columns("ColName").Values = "NewValue";
  Table.Update();
```

## ULTableSchema class

The ULTableSchema object allows you to obtain the attributes of a table.

### Properties

The ULTableSchema represents metadata about the table. The following are properties of the ULTableSchema class:

| Prototype | Description |
|---|---|
| ColumnCount As Integer (read-only) | The number of columns in this table |
| Indexes As IIndexSchemas | The number of indexes on this table |
| Name As String (read-only) | This table’s name |
| NeverSynchronized As Boolean (read-only) | Indicates if the table is always excluded from synchronization. |
| PrimaryKey As ULIndexSchema (read-only) | The primary key for this table. |
| UploadUnchangedRows As Boolean (read-only) | Indicates if all rows in the table should be uploaded on synchronization, rather than just the rows changed since the last synchronization. |

### InPublication method

Prototype

**InPublication(** *publicationName* **)** As Boolean
Member of **UltraliteActiveX.ULTableSchema**

Description

Indicates whether this table is part of the specified publication.

Parameters

**publicationName** The name of the publication you are checking.

Returns

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

**False** if the table is not part of the publication.

# Index