SYBASE®
iAnywhere.

UltraLite®
AppForge Programming

Copyright and trademarks

Contents

About This Manual

Subject

This manual describes UltraLite for AppForge. With UltraLite for AppForge you can develop and deploy database applications to handheld, mobile, or embedded devices, running Palm OS, Symbian OS, or Windows CE.

Audience

This manual is intended for AppForge 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.

SQL Anywhere 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 documentation

The complete SQL Anywhere documentation is available in two forms: an online form that combines all books, and as separate PDF files for each book. Both forms of the documentation contain identical information and consist of the following books:

♦ **SQL Anywhere 10 - Introduction**   This book introduces SQL Anywhere 10—a product that provides data management and data exchange technologies, enabling the rapid development of database-powered applications for server, desktop, mobile, and remote office environments.

♦ **SQL Anywhere 10 - Changes and Upgrading**   This book describes new features in SQL Anywhere 10 and in previous versions of the software, as well as upgrade instructions.

♦ **SQL Anywhere Server - Database Administration**   This book covers material related to running, managing, and configuring SQL Anywhere databases. It describes database connections, the database server, database files, backup procedures, security, high availability, and replication with Replication Server, as well as administration utilities and options.

♦ **SQL Anywhere Server - SQL Usage**   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.

♦ **SQL Anywhere Server - SQL Reference**   This book provides a complete reference for the SQL language used by SQL Anywhere. It also describes the SQL Anywhere system views and procedures.

♦ **SQL Anywhere Server - Programming**   This book describes how to build and deploy database applications using the C, C++, and Java programming languages, as well as Visual Studio .NET. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by these tools.

♦ **SQL Anywhere 10 - Error Messages**   This book provides a complete listing of SQL Anywhere error messages together with diagnostic information.

♦ **MobiLink - Getting Started**   This manual introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.

♦ **MobiLink - Server Administration**   This manual describes how to set up and administer MobiLink server-side utilities and functionality.

♦ **MobiLink - Client Administration**   This manual describes how to set up, configure, and synchronize MobiLink clients. MobiLink clients can be SQL Anywhere or UltraLite databases.

♦ **MobiLink - Server-Initiated Synchronization**   This manual describes MobiLink server-initiated synchronization, a feature of MobiLink that allows you to initiate synchronization or other remote actions from the consolidated database.

♦ **QAnywhere**   This manual describes QAnywhere, which is a messaging platform for mobile and wireless clients as well as traditional desktop and laptop clients.

♦ **SQL Remote**   This book describes the SQL Remote data replication system for mobile computing, which enables sharing of data between a SQL Anywhere consolidated database and many SQL Anywhere remote databases using an indirect link such as email or file transfer.

♦ **SQL Anywhere 10 - Context-Sensitive Help**   This manual contains the context-sensitive help for the Connect dialog, the Query Editor, the MobiLink Monitor, MobiLink Model mode, the SQL Anywhere Console utility, the Index Consultant, and Interactive SQL.

♦ **UltraLite - Database Management and Reference**   This manual introduces the UltraLite database system for small devices.

♦ **UltraLite - AppForge Programming**   This manual describes UltraLite for AppForge. With UltraLite for AppForge you can develop and deploy database applications to handheld, mobile, or embedded devices, running Palm OS, Symbian OS, or Windows CE.

♦ **UltraLite - .NET Programming**   This manual describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.

♦ **UltraLite - M-Business Anywhere Programming**   This manual describes UltraLite for M-Business Anywhere. With UltraLite for M-Business Anywhere you can develop and deploy web-based database applications to handheld, mobile, or embedded devices, running Palm OS, Windows CE, or Windows XP.

♦ **UltraLite - C and C++ Programming**   This manual describes UltraLite C and C++ programming interfaces. With UltraLite, you can develop and deploy database applications to handheld, mobile, or embedded devices.

Documentation formats

SQL Anywhere provides documentation in the following formats:

♦ **Online documentation**   The online documentation contains the complete SQL Anywhere documentation, including 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 10 ► 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 or on your installation CD.

♦ **PDF files**   The complete set of SQL Anywhere books is provided as a set of Adobe Portable Document Format (pdf) files, viewable with Adobe Reader.

On Windows, the PDF books are accessible from the online documentation via the PDF link at the top of each page, or from the Windows Start menu (Start ► Programs ► SQL Anywhere 10 ► Online Books - PDF Format).

On Unix, the PDF books are available on your installation CD.

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 uppercase, 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.

Operating system conventions

♦ **Windows**   The Microsoft Windows family of operating systems for desktop and laptop computers. The Windows family includes Windows Vista and Windows XP.

♦ **Windows CE**   Platforms built from the Microsoft Windows CE modular operating system, including the Windows Mobile and Windows Embedded CE platforms.

Windows Mobile is built on Windows CE. It provides a Windows user interface and additional functionality, such as small versions of applications like Word and Excel. Windows Mobile is most commonly seen on mobile devices.

Limitations or variations in SQL Anywhere are commonly based on the underlying operating system (Windows CE), and seldom on the particular variant used (Windows Mobile).

♦ **Unix**   Unless specified, Unix refers to both Linux and Unix platforms.

File name conventions

The documentation generally adopts Windows conventions when describing operating system dependent tasks and features such as paths and file names. In most cases, there is a simple transformation to the syntax used on other operating systems.

♦ **Directories and path names**   The documentation typically lists directory paths using Windows conventions, including colons for drives and backslashes as a directory separator. For example,

```
MobiLink\redirector
```

On Unix, Linux, and Mac OS X, you should use forward slashes instead. For example,

```
MobiLink/redirector
```

If SQL Anywhere is used in a multi-platform environment you must be aware of path name differences between platforms.

♦ **Executable files**   The documentation shows executable file names using Windows conventions, with the suffix *.exe*. On Unix, Linux, and Mac OS X, executable file names have no suffix. On NetWare, executable file names use the suffix *.nlm*.

For example, on Windows, the network database server is *dbsrv10.exe*. On Unix, Linux, and Mac OS X, it is *dbsrv10*. On NetWare, it is *dbsrv10.nlm*.

♦ **install-dir**   The installation process allows you to choose where to install SQL Anywhere, and the documentation refers to this location using the convention *install-dir*.

After installation is complete, the environment variable SQLANY10 specifies the location of the installation directory containing the SQL Anywhere components (*install-dir*). SQLANYSH10 specifies the location of the directory containing components shared by SQL Anywhere with other Sybase applications.

For more information on the default location of *install-dir*, by operating system, see "SQLANY10 environment variable" [*SQL Anywhere Server - Database Administration*].

♦ **samples-dir**   The installation process allows you to choose where to install the samples that are included with SQL Anywhere, and the documentation refers to this location using the convention *samples-dir*.

Copyright © 2007, iAnywhere Solutions, Inc.

After installation is complete, the environment variable SQLANYSAMP10 specifies the location of the directory containing the samples (*samples-dir*). From the Windows Start menu, choosing Programs ► SQL Anywhere 10 ► Sample Applications and Projects opens a Windows Explorer window in this directory.

For more information on the default location of *samples-dir*, by operating system, see "Samples directory" [*SQL Anywhere Server - Database Administration*].

♦ **Environment variables** The documentation refers to setting environment variables. On Windows, environment variables are referred to using the syntax *%envvar%*. On Unix, Linux, and Mac OS X, environment variables are referred to using the syntax *$envvar* or *${envvar}*.

Unix, Linux, and Mac OS X environment variables are stored in shell and login startup files, such as *.cshrc* or *.tcshrc*.

Graphic icons

The following icons are used in this documentation.

♦ A client application.
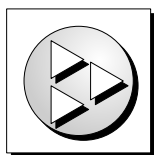
♦ A database server, such as SQL Anywhere.
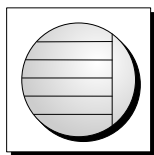
♦ An UltraLite application.
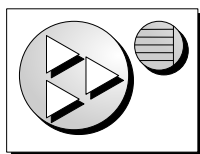
♦ A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.
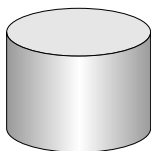
♦ Replication or synchronization middleware. These assist in sharing data among databases. Examples are the MobiLink server and the SQL Remote Message Agent.
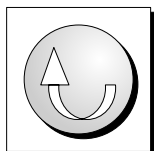
♦ A Sybase Replication Server

♦ A programming interface.

Interface

Copyright © 2007, iAnywhere Solutions, Inc.
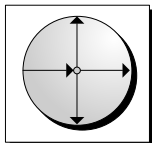
Finding out more and providing feedback

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 Sybase iAnywhere 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. You can find this information by entering **dbeng10 -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 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 email comments and suggestions to the SQL Anywhere documentation team at iasdoc@ianywhere.com. Although we do not reply to emails 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 for AppForge

Contents

UltraLite for AppForge features

UltraLite for AppForge 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 for AppForge, you require the following:

♦ Microsoft .NET (Visual Basic .NET or C#) or Visual Basic 6.

You must install a service pack that meets the requirements for the version of AppForge MobileVB or AppForge Crossfire that you are using. For more information, see the AppForge web site. If you are using Visual Basic 6, it is recommended that you install at least service pack 5.

> **AppForge Client**
> To deploy applications using UltraLite for AppForge you need the appropriate AppForge Client for the target device. For more information about AppForge Clients, see the AppForge web site.

♦ AppForge MobileVB, or AppForge Crossfire.

Target platforms

UltraLite for AppForge supports the following target platforms:

♦ Windows CE 3.0 and higher, with Pocket PC on the ARM processor, including Windows Mobile 5.0.

♦ Sony Ericsson UIQ 2.08 using ARMI (upward compatible with 2.1)

♦ Nokia Series 60 and Series 80 using ARMI

♦ Motorola 1000 using ARMI

♦ Palm OS version 4 and higher.

For more information about deployment, see the UltraLite Deployment Option for SQL Anywhere table in SQL Anywhere Supported Platforms and Engineering Support Status.

UltraLite for AppForge architecture

The UltraLite programming interface exposes a set of objects for data manipulation using an UltraLite database. The following figure describes the object hierarchy.
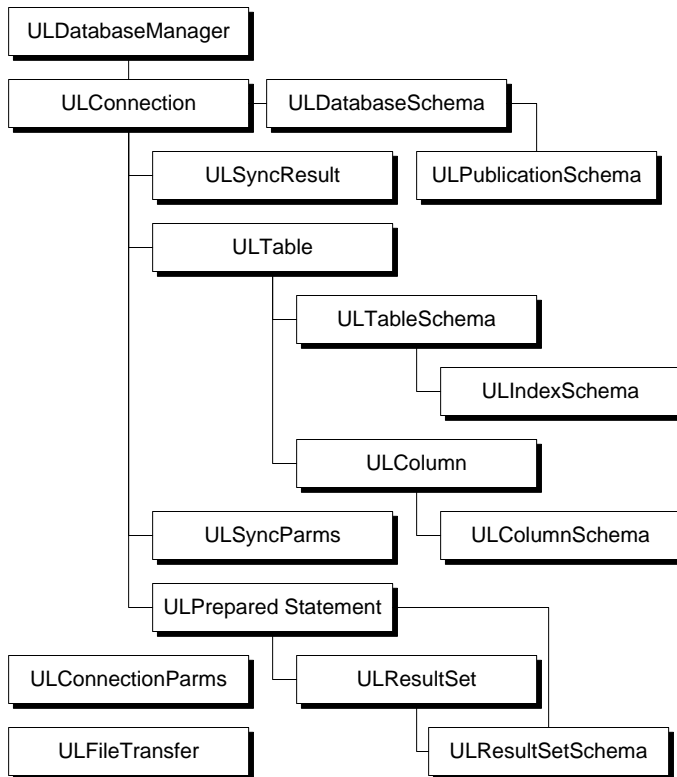
The following list describes some of the more commonly-used high-level objects.

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

See "ULDatabaseManager class" on page 90.

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

You can use a Connection Parameters control and specify connection parameters in a Visual Basic property sheet.

See "ULConnectionParms class" on page 88.

♦ **ULFileTransfer**   manages a file transfer with a MobiLink server.

See "ULFileTransfer class" on page 97.

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

See "ULConnection class" on page 79.

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

See:

- ♦ "ULPreparedStatement class" on page 103
- ♦ "ULResultSet class" on page 110
- ♦ "ULResultSetSchema class" on page 123

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

See:

- ♦ "ULTable class" on page 149
- ♦ "ULColumn class" on page 73

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

For more information about synchronization with MobiLink, see "UltraLite Clients" [*MobiLink - Client Administration*].

CHAPTER 2

Understanding UltraLite Development with AppForge

Contents

Preparing to use UltraLite for AppForge

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

Adding UltraLite to the MobileVB design environment

To access the UltraLite control from your MobileVB or Crossfire project, you must add UltraLite for MobileVB to the design environment.

♦ **To add the UltraLite connection parameters control**

1. From the Visual Basic menu, choose Project ► Components.

2. Click the Controls tab.

3. Scroll down the list to choose UltraLite Connection Parameters 10.0. Click OK.

   If this item does not appear in the list of available controls, complete the following steps:

   ♦ Close Visual Basic and save your project.

   ♦ Open a command prompt and run the following command:

   ```
   ulafreg -r
   ```

   See "UltraLite AppForge Registry utility (ulafreg)" [*UltraLite - Database Management and Reference*].

   ♦ Restart Visual Basic and open your project.

   ♦ Choose Project ► Components.

   ♦ Choose UltraLite Connection Parameters 10.0.

   A database icon is added to your toolbar. To add a ULConnectionParms object to your form you double-click this icon.

Adding a reference to UltraLite for MobileVB

Once SQL Anywhere is installed, UltraLite for MobileVB is automatically added to any new MobileVB project. It is therefore not usually necessary to manually add a reference to UltraLite for MobileVB to a project. The following procedure is provided for occasional situations where you may need to add a reference manually, such as if you install MobileVB after installing SQL Anywhere.

♦ **To add a reference to UltraLite for MobileVB**

1. From the Visual Basic menu, choose Project ► References.

2. If iAnywhere Solutions, UltraLite for MobileVB 10.0 is included in the list of available references, select it and click OK.

If iAnywhere Solutions, UltraLite for MobileVB 10.0 does not appear in the list of available references:

♦ Open a command prompt and run the following command:

```
ulafreg -r
```

See "UltraLite AppForge Registry utility (ulafreg)" [*UltraLite - Database Management and Reference*].

♦ Choose iAnywhere Solutions, UltraLite for MobileVB 10.0 and click OK.

Adding UltraLite to the Crossfire design environment

Although the SQL Anywhere installer automatically adds UltraLite to your Crossfire design environment, there are cases where you may have to add UltraLite to the environment manually. For example, if you install Crossfire after you install SQL Anywhere, you may need to carry out this procedure.

To find out if you need to add UltraLite to Crossfire, check that a new Crossfire project includes a reference to iAnywhere.UltraLiteForAppForge. If it does not, you need to add UltraLite to the environment. Also, check if the ULConnectionParms class appears in the AppForge panel of the toolbox. If it does not, you need to add UltraLite to the environment.

♦ **To add UltraLite references and controls to your Crossfire project**

1. Register UltraLite for MobileVB with Crossfire.

   a. Ensure that Crossfire is closed.

   b. Open a command prompt and run the following command:

   ```
   ulafreg -r
   ```

   See "UltraLite AppForge Registry utility (ulafreg)" [*UltraLite - Database Management and Reference*].

   c. If you have upgraded a MobileVB project remove the reference to UltraLiteAFLib from the Visual Basic.NET Solution Explorer.

   d. Add a reference to *iAnywhere.UltraLiteForAppForge.dll*

      i. From the Microsoft Development Environment menu, choose Project ► Add Reference and browse to the *install-dir\ultralite\UltraLiteForAppForge\win32* subdirectory of your SQL Anywhere installation.

      ii. Select *iAnywhere.UltraLiteForAppForge.dll* and click Open.

      iii. Click OK to add the reference.

2. Add the ULConnectionParms control to the AppForge toolbox.

   a. In the Microsoft Development Environment, right click the AppForge toolbox and choose Add/ Remove Items. A dialog appears.

b.   Click the COM Components tab.

c.   Scroll down to the entry named ULConnectionParms Class. Check the box beside this component and click OK.

d.   The ULConnectionParms control is added to the toolbox.

Creating UltraLite databases

You can create an UltraLite database using UltraLite in Sybase Central or the ulcreate utility:

♦ **UltraLite in Sybase Central**   Use the Create Database wizard to create an UltraLite database.

See "Creating an UltraLite database from Sybase Central" [*UltraLite - Database Management and Reference*].

♦ **The ulcreate utility**   You can use the ulcreate utility to create an empty UltraLite database.

See "UltraLite Create Database utility (ulcreate)" [*UltraLite - Database Management and Reference*].

Applications can create an UltraLite database dynamically by using the UltraLite CreateDatabase function. Since application deployment environments support deploying additional files with an application, most applications can be simplified by distributing an initial database along with the executable code. UltraLite databases are comprised of a single file.

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.

See "ULConnection class" on page 79.

♦ **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.

See "Commit method" on page 81.

♦ **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.

See "Authenticating users" on page 29.

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

See "Synchronizing data" on page 30.

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

See "GetTable method" on page 82.

Connecting to a database

You can connect to a database using either a ULConnectionParms object or a connection string. Use a ULConnectionParms object to manipulate multiple connection parameters for different target device platforms. Methods that use a connection string require you specify the different target platform strings in one large string.

For more information about connection parameters, see "UltraLite Connection String Parameters Reference" [*UltraLite - Database Management and Reference*]

The following procedure illustrates connecting to an UltraLite database:

♦ **Connect to UltraLite database**

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.

```
'MobileVB using VB6
Public DatabaseMgr As ULDatabaseManager
Set DatabaseMgr = New ULDatabaseManager

'Crossfire using vb.net
Public DatabaseMgr As New UltraLiteAFLib.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.

    ```
    'MobileVB using VB6
    Public Connection As New ULConnection

    'Crossfire using vb.net
    Public Connection As UltraLiteAFLib.ULDatabaseManager
    ```

3. Create a ULConnectionParms object:

    Double-click the ULConnectionParms object on the MobileVB tool palette. A ULConnectionParms object appears on your form.

4. Set the required properties of the ULConnectionParms object:

    In the ULConnectionParms properties window, specify properties such as the location of the database, and a user name and password for your database.

    Using the following properties, you must specify a database file for OpenConnection.

    For information about additional properties, see "Properties" on page 88.

Keyword	Description
DatabaseOnCE	The path and file name of the UltraLite database on Windows CE.
DatabaseOnDesktop	The path and file name of the UltraLite database on the desktop computer.

5. Open a connection to the database:

    OpenConnection returns an open connection as a ULConnection object. This 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 OpenConnection method returns an error.

    ```
    'MobileVB using VB6
    On Error Resume Next
    Set Connection = DatabaseMgr.OpenConnection(ULConnectionParms1.ToString
    ())

    'Crossfire using vb.net
    Try
      Connection = _
        DatabaseMgr.OpenConnection(ULConnectionParms1.ToString())
    Catch
      If Err.Number = _
        UltraLiteAFLib.ULSQLCode.ulSQLE_ULTRALITE_DATABASE_NOT_FOUND _
    ```

```
    Then
      ...
  End Try


// Crossfire using C#
using UltraLiteAFLib;
...
  public  UltraLiteAFLib.ULConnection Connection;
  public  UltraLiteAFLib.ULDatabaseManager DatabaseMgr;
  private UltraLiteAFLib._ULConnectionParms_ingotClass parms;
          // dropped onto design form
  parms.DatabaseOnCE = AppForge.System.AppPath + "\\mydb.udb";
  try {
        Connection = DatabaseMgr.OpenConnection( parms.ToString );
  } catch ( Exception ex ) {
    Debug.WriteLn("Connect failed: " + ex.Message );
  }
```

Encryption and obfuscation

UltraLite databases can be created with one of the following choices for data security: obfuscation or encryption. By default, UltraLite databases are created without any specific measures to obscure the data in the database. Utilities that examine the file which contains an UltraLite database and can display raw disk data could reveal character data stored in the database. The format of the actual database file is proprietary, but the contents are able to be viewed.

Obfuscation and encryption are creation-time configuration options. Although the actual encryption key can be changed, the choice to obfuscate or encrypt the data in the database cannot be changed without unloading the database, creating a new database, and reloading the data.

Encryption

To create a database with encryption, you must specify the encryption key when the database is created.

To open a connection to an encrypted database, you use the ULConnectionParms.EncryptionKey property to supply the encryption key string used when the database was created.

For more information about the EncryptionKey property, see "UltraLite DBKEY connection parameter" [*UltraLite - Database Management and Reference*].

You can change the encryption key by specifying a new encryption key on the Connection object. An application must first connect using the existing encryption key and then specify a new encryption key. In the following example, "apricot" is the new encryption key:

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

See "ChangeEncryptionKey method" on page 80.

After the database is encrypted, all connections to the database must specify the correct encryption key. Otherwise, the connection fails. If the encryption key is not known, *the data in the database cannot be retrieved*.

Obfuscation

To obfuscate the database, set the obfuscation option when you create the database. Obfuscation is a simple masking of the contents of the database that is meant to prevent utility programs from revealing the raw contents of the database file. Databases created with obfuscation operate transparently to the user and the application program; there are no additional programming considerations.

For more information about database encryption, see "UltraLite obfuscate property" [*UltraLite - Database Management and Reference*] and "UltraLite security considerations" [*UltraLite - Database Management and Reference*].

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 20.

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 SQL features, see "SQL Language Elements" [*SQL Anywhere Server - SQL Reference*].

The sequence of operations required is similar for any SQL operation. For an overview, see "SQL Statements" [*SQL Anywhere Server - SQL Reference*].

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.

See "ULPreparedStatement class" on page 103.

It is important for applications to free up resources after using prepared statements by calling the Close method.

> **Using parameters in your prepared statements**
> Placeholders for parameters are identified 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 parameter 1, and the second as parameter 2.

♦ To INSERT a row

1. Declare a ULPreparedStatement object.

```
'MobileVB using VB6
Dim PrepStmt As ULPreparedStatement

'Crossfire using vb.net
Dim PrepStmt As UltraLiteAFLib.ULPreparedStatement

// Crossfire using C#
 ULPreparedStatement PrepStmt = null;
```

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.

```
'MobileVB using VB6
Set PrepStmt = Connection.PrepareStatement( _
    "INSERT INTO TableName(ColumnName) VALUES ( ? )")

'Crossfire using vb.net
PrepStmt = Connection.PrepareStatement( _
    "INSERT INTO TableName(ColumnName) VALUES( ? )")

// CrossFire using C#
try {
        PrepStmt = Connection.PrepareStatement("INSERT INTO ...", null);
} catch ( Exception ){
}
if (PrepStmt == null) // failed
```

3. Assign parameter values for the statement.

```
PrepStmt.SetStringParameter (1, "Bob")
```

4. Execute the statement and free resources after the command is completed.

```
PrepStmt.ExecuteStatement
PrepStmt.Close()
```

♦ **To UPDATE a row**

1. Declare a ULPreparedStatement object.

```
Dim PrepStmt As ULPreparedStatement
```

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.

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

3. Assign parameter values for the statement.

```
PrepStmt.SetStringParameter (1, "newvalue")
PrepStmt.SetStringParameter (2, "oldvalue")
```

4. Execute the statement and free resources after the command is completed.

```
PrepStmt.ExecuteStatement
PrepStmt.Close()
```

♦ **To DELETE a row**

1. Declare a ULPreparedStatement object.

```
'MobileVB using VB6
Dim PrepStmt As ULPreparedStatement
```

```
'Crossfire using vb.net
Dim PrepStmt As UltraLiteAFLib.ULPreparedStatement
```

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

```
'MobileVB using VB6
Set PrepStmt = Connection.PrepareStatement( _
    "DELETE FROM customer WHERE ID = ?")

'Crossfire using vb.net
PrepStmt = Connection.PrepareStatement( _
    "DELETE FROM customer WHERE ID = ?")
```

3.   Assign parameter values for the statement.

```
PrepStmt.SetStringParameter (1, "oldvalue")
```

4.   Execute the statement and free resources after the command is completed.

```
PrepStmt.ExecuteStatement
PrepStmt.Close()
```

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 methods of the ULResultSet class.

See "ULResultSet class" on page 110.

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 18.

```
'MobileVB using VB6
Dim MyResultSet As ULResultSet
Dim PrepStmt As ULPreparedStatement
PrepStmt = Connection.PrepareStatement( _
    "SELECT ID, Name FROM customer")
MyResultSet = PrepStmt.ExecuteQuery
MyResultSet.MoveFirst

'Crossfire using vb.net
Dim MyResultSet As UltraLiteAFLib.ULResultSet
Dim PrepStmt As UltraLiteAFLib.ULPreparedStatement
PrepStmt = Connection.PrepareStatement( _
    "SELECT ID, Name FROM customer")
MyResultSet = PrepStmt.ExecuteQuery
MyResultSet.MoveFirst
```

UltraLite for AppForge provides you with methods to get data of particular types from the UltraLite database into a result set. MobileVB does not support the use of Variant data types and, because of this, UltraLite for

MobileVB includes functions to handle all types of data. Each of these methods is called using the following template, where Index is the ordinal position of the column name in your SELECT statement:

```
MyResultSetName.MethodName( Index )
```

Example

The following code demonstrates how to use the GetString method to obtain the column values for the current row.

The GetString method uses the following syntax, where Index is the ordinal position of the column name in your SELECT statement.

```
MyResultSetName.GetString(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.

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

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.

    ```
    'MobileVB using VB6
    Dim PrepStmt As ULPreparedStatement

    'Crossfire using vb.net
    Dim PrepStmt As UltraLiteAFLib.ULPreparedStatement
    ```

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

    ```
    Set PrepStmt = Connection.PrepareStatement( _
        "SELECT ColumnName FROM TableName")
    ```

3. Execute the query.

    In the code below, an AFListBox captures the result of the SELECT query.

    ```
    Dim MyResultSet As ULResultSet
    Set MyResultSet = PrepStmt.ExecuteQuery
    While MyResultSet.MoveNext
      aflistbox.AddItem MyResultSet.GetString(1)
    Wend
    ```

4. After processing the query, free resources by closing the result set.

```
        MyResultSet.Close()
```

Navigation with dynamic SQL

UltraLite for MobileVB provides you with a number of methods to navigate a result set 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.

```
'MobileVB using VB6
Set PrepStmt = Connection.PrepareStatement( _
    "SELECT ID, Name FROM customer")
Set MyResultSet = PrepStmt.ExecuteQuery
MyResultSet.MoveFirst

'Crossfire using vb.net
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 110.

ULResultSet schema property

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

Example

The following example shows how you can use ULResultSet.Schema to display schema information in a MobileVB grid. The example assumes you have a ULResultSet named MyResultSet and a MobileVB grid named grdSchema.

```
'MobileVB using VB6
Dim i As Integer
For i = 1 To MyResultSet.Schema.ColumnCount
  grdSchema.AddItem (MyResultSet.Schema.ColumnName(i) _
  & Chr(9) & CStr(MyResultSet.Schema.ColumnSQLType(i))), 0
Next i
grdSchema.AddItem _
  ("Column Name" & Chr(9) & "Column Type"), 0
```

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 14.

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 for MobileVB provides you with a number of methods to navigate a table 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 with the last name of each customer.

```
'MobileVB using VB6
Dim TCustomer as ULTable
Set TCustomer = Conn.GetTable("customer")
TCustomer.Open
While TCustomer.MoveNext
   MsgBox TCustomer.Column( "lname" ).StringValue
Wend

'Crossfire using vb.net
Dim TCustomer as UltraLiteAFLib.ULTable
```

```
Set TCustomer = Conn.GetTable("Customer")
TCustomer.Open
While TCustomer.MoveNext
    MsgBox TCustomer.Column("LName").StringValue
Wend
```

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.

```
'MobileVB using VB6
Set TCustomer = Conn.GetTable("customer")
TCustomer.Open "ix_name"
TCustomer.MoveFirst

'Crossfire using vb.net
TCustomer = Conn.GetTable("customer")
TCustomer.Open "ix_name"
TCustomer.MoveFirst
```

Accessing the values of the current row

At any time, a ULTable object is positioned at one of the following 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 Column method together with an appropriate 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.

```
Dim colID, colFirstName, colLastName As ULColumn
Set colID = tCustomer.Column("ID")
Set colFirstName = tCustomer.Column("fname")
Set colLastName = tCustomer.Column("lname")
txtID.Text = colID.IntegerValue
txtFirstName.Text = colFirstName.StringValue
txtLastName.Text = colLastName.StringValue
```

You can also use the properties of ULColumn to set values.

```
colLastName.StringValue = "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 code is incorrect
TCustomer.MoveBeforeFirst
id = TCustomer.Column( "ID" ).IntegerValue
```

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

See "GetByteChunk method" on page 74.

Casting values

The ULColumn property you choose must match the Visual Basic data type you want to assign. UltraLite automatically casts incompatible data types, so that you could use the StringValue method to fetch an integer value into a string variable, and so on. See "Converting data types explicitly" [*UltraLite - Database Management and Reference*].

For more information about accessing values of the current row, see "ULColumn class" on page 73.

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.

 See "FindBegin method" on page 151.

♦ **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.

 See "LookupBackward method" on page 154.

♦ **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.

```
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.

```
tCustomer.Column("lname").StringValue = "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.

```
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.

```
colID.IntegerValue = 3
```

Using UltraLite modes

The UltraLite mode determines the purpose for which the values in the buffer are 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 want to update.

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

2. Enter Update mode.

   For example, the following instruction enters Update mode on the table tCustomer.

```
tCustomer.UpdateBegin
```

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

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

    ```
    ColFirstName.StringValue = "Elizabeth"
    ```

4. Execute the Update.

    ```
    tCustomer.Update
    ```

After the update operation, the current row is the row that was just updated. If you changed the value of a column in the index specified when the ULTable object was opened, 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. See .

> **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 inserted according to 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.

    ```
    CustomerTable.InsertBegin
    ```

2. Set the values for the new row.

    If you do not set a value for a column, and that column has a default value defined, 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.

    ```
    CustomerTable.Column("FName").StringValue = fname
    CustomerTable.Column("LName").StringValue = 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.

```
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 want to delete.

2. Execute the deletion.

```
tCustomer.Delete
```

Working with BLOB data

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

See "GetByteChunk method" on page 74.

Example

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

```
'MobileVB using VB6
Dim table as ULTable
Dim col As ULColumn
Dim data(1 to 1024) As Byte
Dim data_fit As Boolean
Dim size As Long
Set table = Conn.GetTable("image")
table.Open
size = 1024
Set col = table.Column("img_data")
data_fit = col.GetByteChunk(VarPtr(data(1)), size)
If data_fit Then
  'No truncation
Else
  'data truncated at 1024
End if
table.Close

'Crossfire using vb.net
Dim table as ULTable
Dim col As ULColumn
Dim data(1 to 1024) As Byte
Dim data_fit As Boolean
Dim size As Long
Set table = Conn.GetTable("image")
table.Open
size = 1024
Set col = table.Column("img_data")
' The data argument must be a local variable
data_fit = col.GetByteChunk(data, size)
If data_fit Then
  'No truncation
```

```
      Else
        'data truncated at 1024
      End if
      table.Close
```

Managing transactions

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 AutoCommit is 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.

♦ **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 accessed using the ULTableSchema.GetIndex method.

♦ **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 accessible using the ULDatabaseSchema.GetPublicationSchema method.

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

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

Handling errors

In normal operation, UltraLite for AppForge 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 for AppForge, see "ULSQLCode enumeration" on page 124.

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

See "ULSQLCode enumeration" on page 124.

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

```
'MobileVB using VB6
Public Function GetError() As ULSQLCode
    GetError = Err.Number
End Function
```

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

Authenticating users

An UltraLite database may define up to four user IDs and associated passwords. UltraLite databases are created with an initial user ID of DBA with password sql; you must first connect as this initial user. While connected to the database, an application may grant connection authority to a new user ID, change the password for an existing user ID or revoke connection authority from an existing user ID.

Note that a user ID cannot be changed directly; however, you can add a new user ID and then delete the existing user ID.

UltraLite does not associate any specific rights with a user ID. All user IDs that are defined for the database can be used to connect to that UltraLite database. Code within an application can enforce different capabilities based on the user information supplied to the application.

For more information about granting or revoking connection authority, see "GrantConnectTo method" on page 82 and "RevokeConnectFrom method" on page 83.

♦ **To add a user or change the password for an existing user**

1. Connect to the database as an existing user.

2. Grant connection authority to a specific user with the desired password:

```
conn.GrantConnectTo("Robert", "newPassword")
```

♦ **To delete an existing user**

1. Connect to the database as an existing user.

2. Revoke a specified user's connection authority as follows:

```
conn.RevokeConnectFrom("Robert")
```

Synchronizing data

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

This section provides a brief introduction to synchronization and describes some features of particular interest to users of UltraLite for AppForge. For more information about synchronization, see "UltraLite Clients" [*MobiLink - Client Administration*].

You can also find a working example of synchronization in the CustDB sample application. See "Tutorial: A Sample Application for AppForge MobileVB" on page 57.

The UltraLite AppForge component does not support ECC or FIPS synchronization encryption for the Palm OS. For the Symbian OS, no synchronization encryption is available.

UltraLite for AppForge 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.

Separately licensed component required
ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.
See "Separately licensed components" [*SQL Anywhere 10 - Introduction*].

♦ **To synchronize over TCP/IP or HTTP**

1. Prepare the synchronization information.

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

    For more information about the properties and the values that you should set, see "UltraLite Clients" [*MobiLink - Client Administration*].

2. Synchronize.

    Call the ULConnection.Synchronize method.

Adding the synchronization template

UltraLite for MobileVB includes a template form that can be used to monitor the status of a synchronization session. A version of this form is included for both Palm OS and Windows CE. You can use these templates in your application, you can customize them, or you can simply examine them to learn how UltraLite synchronization events work.
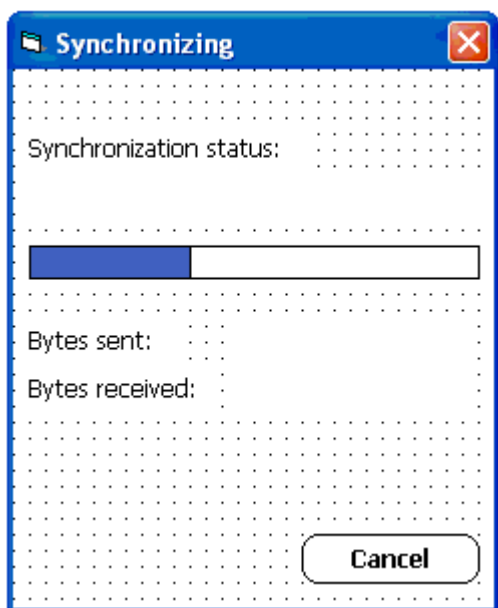
The way to add this template to your application depends on whether you are using MobileVB or Crossfire.

♦ To add a synchronization template to your application (MobileVB)

1. From the project menu, choose Add Form.

2. Select either UltraLite for MobileVB Sync Form (Windows CE) or UltraLite for MobileVB Sync Form (Palm).

3. Click Open. A copy of the form is added to your application.

♦ To add a synchronization template to your application (Crossfire)

1. From the project menu, choose Add New Item.

2. From Local Project Items ► Ultralite_Crossfire Forms, select UltraLite Crossfire 10 Sync Form for CE, Palm or PalmHires depending on your application.

3. Click Open. A copy of the form is added to your application.

Writing code to use the synchronization form

Call the InitSyncForm function, passing it your ULConnection object. This must be done before each synchronization.

Example

The following code uses a synchronization status form named Form_Sync and a ULConnection object named Connection.

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

Each time your application synchronizes, the synchronization status form appears. As synchronization progresses, your end user can observe the progress bar and byte count. When synchronization completes, the form closes. The Cancel button instructs UltraLite to cancel the current synchronization.

See "Tutorial: A Sample Application for AppForge MobileVB" on page 57.

Deploying UltraLite applications

When you have completed your application or when you want 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 for MobileVB applications to Windows CE

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

1. Deploy your application and UltraLite component.

   a. Configure the application settings.

      ♦ From the MobileVB menu, choose MobileVB Settings. A dialog appears.

      ♦ In the left pane, choose Dependencies and click the User Dependencies tab.

      ♦ Click the Add button and select the *c:\tutorial\mvb\tutCustomer.udb*. This indicates to MobileVB that the file should be included in the deployment.

      ♦ Choose the PocketPC Settings item in the left pane

      ♦ Enter *\tutorial\mvb* for the Device Installation Path.

      ♦ Click OK to close the dialog.

   b. From the MobileVB menu, choose Deploy to Device, and choose the PocketPC device. If a dialog appears asking if you want to save the project, click Yes.

2. Deploy an initial copy of the UltraLite database.

   You can then use synchronization to load an initial copy of the data. You can deploy the *.prc* file in the standard manner from the Install Tool included with your Palm Desktop Organizer Software.

   You must place the database file so that it can be located by the application. The DatabaseOnCE connection parameters define the location.

   See "UltraLite CE_FILE connection parameter" [*UltraLite - Database Management and Reference*].

3. Deploy the ActiveSync provider.

   This step is required only if the application is synchronizing using ActiveSync. See ActiveSync in the UltraLite table in SQL Anywhere 10.0.1 Components by Platform.

See also

   ♦ "Deploying ActiveSync and HotSync for UltraLite clients" [*MobiLink - Client Administration*]

Deploying UltraLite for MobileVB applications to Palm OS

You must carry out the following steps to deploy an UltraLite application to a Palm OS device:

1. Deploy your application and UltraLite component.

   a. Configure the application settings.

      ♦ From the MobileVB menu, choose MobileVB Settings. A dialog appears.

      ♦ In the left pane, choose Dependencies and click the User Dependencies tab.

      ♦ Click the Add button and select the *c:\tutorial\mvb\tutCustomer.pdb*. This indicates to MobileVB that the file should be included in the deployment.

      ♦ Choose the Palm OS Settings item in the left pane and enter the creator ID of your application. Choose a valid HotSync name. Click OK to close the dialog.

   b. From the MobileVB menu, choose Deploy to Device, and choose the Palm OS device. If a dialog appears asking if you want to save the project, click Yes.

2. Deploy an initial UltraLite database.

   In many situations it is sufficient to deploy an UltraLite database file. You can then use synchronization to load data.

   You can create *.pdb* files for deployment to Palm OS using UltraLite utilities sych as ulcreate, ulload, and ulinit.

   You must supply a database using the correct creator ID, so that it can be located by your application. The DatabaseOnPalm connection parameter uses the creator ID to find the database.

   See "UltraLite PALM_FILE connection parameter" [*UltraLite - Database Management and Reference*].

3. Deploy the MobiLink synchronization conduit for HotSync.

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

Maintaining state in UltraLite Palm applications

Palm OS devices run only one application at a time. However, when a user switches from your application to another application, and then returns to your application, it is common to make applications appear as they were simply suspended while the user was working with other applications. To maintain this illusion, the application must save its internal state when the user switches to another application. When the application is launched again, it must restore its internal state.

Saving and restoring state in a database application can be challenging, as the application must re-open result sets and re-position the application within those result sets. UltraLite provides a way for you to save and restore application state.

The state of cursors on result sets is maintained automatically. MobileVB applications that use the table-based API provide a value for the persistent name parameter in the Open method of the ULTable object.

Understanding how state is maintained

For each table whose state is being preserved, UltraLite stores a name for the table as well as enough information to restore the state of the table. The name associated with the table may be, but is not required to be, the name of the table. It is called the **persistent name**.

UltraLite applications can open more than one instance of the same table at the same time. In this case, the table name is not unique. For example, the following code (using MobileVB) opens the same table twice:

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

When opening a table, an application can optionally provide a persistent name as a parameter. If the state of the persistent name has been saved, the table is opened and positioned to the proper row. If not, the table is opened and positioned before the first row.

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

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

```
'MobileVB using VB6
Dim table As ULTable
Set table = Connection.GetTable( "ULCustomer" )
table.Open "", "customer"
```

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

In the Open call (the third line of code), the first parameter is the empty string, which indicates that the data is ordered by the primary key. The second parameter is the persistent name being assigned to the table. If

the application terminates while this table is still open, the state PDB associates *customer* with the ULCustomer table and save the current position.

Persistent name notes

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

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

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

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

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

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

Example: Using the persistent name to maintain state information

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

```
'MobileVB using VB6
CustomerTable.Open
AddRow "John", "Doe", "Atlanta"
AddRow "Mary", "Smith", "Toronto"
AddRow "Jane", "Anderson", "New York"
AddRow "Margaret", "Billington", "Vancouver"
AddRow "Fred", "Jones", "London"
AddRow "Jack", "Frost", "Dublin"
AddRow "David", "Reiser", "Berlin"
AddRow "Kathy", "Stevens", "Waterloo"
AddRow "Rebecca", "Gable", "Paris"
AddRow "George", "Jenkins", "Madrid"
CustomerTable.Close
```

This code adds ten rows to the ULCustomer table. It calls Open on the table, but in this case a persistent name is not supplied because there is no need to maintain the position in the table. Since the code only inserts data, the position in the table is not relevant.

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

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

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

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

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

Notes on AppForge for Symbian OS

This section provides some notes about UltraLite development on the Symbian OS using AppForge.

Supported platforms and devices

UltraLite for AppForge supports all Symbian OS 7 and Symbian OS 8 devices supported by AppForge Crossfire. These include UIQ devices such as the Sony Ericsson P800 and P900 series, Nokia S60 devices such as the Nokia N-90, and Nokia S-80 devices such as the Nokia 9300 Communicator.

The devices supported depend on the version of AppForge Crossfire that you are using. For example, in order to support the Nokia 9300 Communicator you must have at least Crossfire 5.6.1.

Sample application

The Crossfire CustDB application includes projects for both the Nokia 9300 Communicator and for Sony Ericsson P800/P900 devices. You can find the application in *samples-dir\UltraLiteForAppForge \CF_CustDB*.

♦ **To use the CustDB sample application**

1. From a command prompt, run the makedbs.bat file to create the UltraLite database.

2. Open the solution file in Visual Studio .NET 2003.

3. To synchronize, start the MobiLink server by choosing Programs ► SQL Anywhere 10 ► MobiLink ► MobiLink Server Sample.

Symbian-specific notes for AppForge developers

Symbian OS uses a Windows-like nomenclature for its file system. You can use the dbf connection parameter to specify the location of an UltraLite database.

As with other AppForge projects, you must add a reference to iAnywhere.UltraLiteForAppForge in your project.

♦ **To configure a project for your target device:**

1. Select the project to configure.

2. Choose AppForge ► Crossfire Settings

3. Choose Dependencies in the list, and add the UltraLite database file to the User Dependencies list.

4. Select the appropriate device type from the list and use the settings dialog to specify an application UID and other application properties.

For cross-platform development, you can use the AppForge.Platforms.DeviceType enumeration to identify the platform on which the application is running. The Symbian OS members of the enumeration are as follows:

Constantt	Target
SymbianOSCrystal	Nokia 9300/9500
SymbianOSPearl	Nokia S60
SymbianOSQuartz	Sony Ericsson P800/P900/P910 or Motorola A1000

Here is a simple example of platform-independent connection code in Visual Basic .Net:

```
deviceType = sysInfo.DeviceType
path = AppForge.MobileVB.Compatibility.Device.AppPath
If deviceType = AppForge.Platforms.DeviceType.SymbianOSCrystal Or _
 deviceType = AppForge.Platforms.DeviceType.SymbianOSPearl Or _
 deviceType = AppForge.Platforms.DeviceType.SymbianOSQuartz Then
    connString = "dbf=" & path & "\ul_custdb.udb;"
Else
    connString = "dbf=" & path & "\..\ul_custdb.udb;"
End If
connString += "con=custdbConn"
Try
    Connection = DBManager.OpenConnection(connString)
Catch ex As Exception
    MsgBox("Error when connecting to database: " & ex.Message)
    End
End Try
```

Deploying AppForge projects to devices

To deploy an AppForge project to a Symbian OS device, you must first connect your device to your computer using a cable or a Bluetooth connection. The required connectivity software, drivers, and instructions should be provided with your device.

Applications are deployed to Symbian OS devices as an .sis file. If your device is properly connected to your development computer, you can deploy an AppForge project to a Symbian OS device by choosing AppForge > Deploy Application To Device and selecting the device type from the list.

An alternative deployment method is to create the SIS file on your development computer and then to deploy that in a separate operation. You can do this by choosing AppForge > Build Installation File.

Synchronizing applications

When synchronizing applications using TCP/IP or HTTP-based protocols, it is recommended that you specify the host address using an IP address, rather than a host name in the network protocol options that you set in the SyncStream.StreamParms property.

CHAPTER 3

Tutorial: A Sample Application for AppForge Crossfire

Contents

Introduction to Crossfire development tutorial

This tutorial describes how to use AppForge Crossfire to build an UltraLite application for Windows CE or Palm OS. At the end of the tutorial you will have an application and small database on your Windows CE device that synchronizes with a central consolidated database.

For more information about the table API, see "UltraLite for AppForge API Reference" on page 71.

Timing

The tutorial takes about 30 minutes if you copy and paste the code. If you enter the code yourself, it takes significantly longer. If you chose to copy and paste the code from this help information, the special characters less-than, greater-than and ampersand may be pasted incorrectly in the code window and have to be manually corrected.

Prerequisites

This tutorial assumes that you have AppForge Crossfire installed on your computer. It also assumes a basic familiarity with Crossfire development.

The tutorial also assumes that you know how to create an UltraLite database using the command line utility ulcreate, or using UltraLite in Sybase Central. See "Creating an UltraLite database from Sybase Central" [*UltraLite - Database Management and Reference*].

Note

Crossfire users can perform most of this tutorial without SQL Anywhere. The synchronization sections of the tutorial require SQL Anywhere.

Lesson 1: Create a project architecture

The first procedure describes how to create an UltraLite database.

♦ **To create an UltraLite database**

1. Create a directory for this tutorial.

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

2. Create a database using UltraLite in Sybase Central.

   See "Creating an UltraLite database from Sybase Central" [*UltraLite - Database Management and Reference*].

   ♦ **Table name** Customer

   ♦ **Columns in Customer**

Column name	Datatype (size)	Column allows NULL values?	Default value
ID	integer	No	autoincrement
FName	varchar(15)	No	None
LName	varchar(20)	No	None
City	varchar(20)	Yes	None
Phone	varchar(12)	Yes	555-1234

   In an application with synchronization, it is usual to choose a global autoincrement or UUID data type for primary keys. An autoincrement method is chosen here to simplify the tutorial.

   ♦ **Primary key** Ascending ID

3. Save the database.

   If you are developing an application for Windows or Windows CE, choose File ► Save and choose *tutcustomer.udb* in your tutorial directory as the file name.

Create a Crossfire project

The following procedure creates an AppForge Crossfire project for your application and adds a reference to the UltraLite control.

AppForge tools appear in addition to the standard Visual Basic tools on the toolbar to the left of the development environment.

♦ To create a Crossfire project for UltraLite

1. Start Crossfire.

   a. Choose Start ► Programs ► AppForge ► Crossfire. The Crossfire Project Manager dialog appears.

   b. Choose New Project. The Microsoft Development Environment New Project dialog appears.

   c. In the Project Types window click to expand the Visual Basic Projects folder.

   d. In the Templates window, click Crossfire Application.

   e. Leave the project name as CrossfireApp1, and enter your tutorial directory (*c:\tutorial\crossfire*) as the location.

      Click OK.

   f. Choose your deployment platform and click Finish to create the project.

   You should now see a Crossfire form in the Microsoft Visual Basic .NET Development Environment.

2. If the Toolbox is not displayed, choose View ► Toolbox to open it. Open the AppForge tab.

3. Scroll down the list of AppForge controls to and double-click ULConnectionParms to add the database connection object to the form.

Troubleshooting

If your Crossfire project does not include a reference to iAnywhere.UltraLiteForAppForge.dll, and if the ULConnectionParms class does not appear in the list of AppForge controls, you need to register UltraLite with Crossfire. This may occur if, for example, you install Crossfire after installing SQL Anywhere.

See "Adding UltraLite to the Crossfire design environment" on page 7.

What's next

You now have an UltraLite database and a Crossfire project with an UltraLite control on a form. The next step is to create the application interface.

Lesson 2: Create the application interface

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. From the AppForge controls, add the following controls 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 Build ► Build Solution.

Your form should look as follows:

Copyright © 2007, iAnywhere Solutions, Inc.
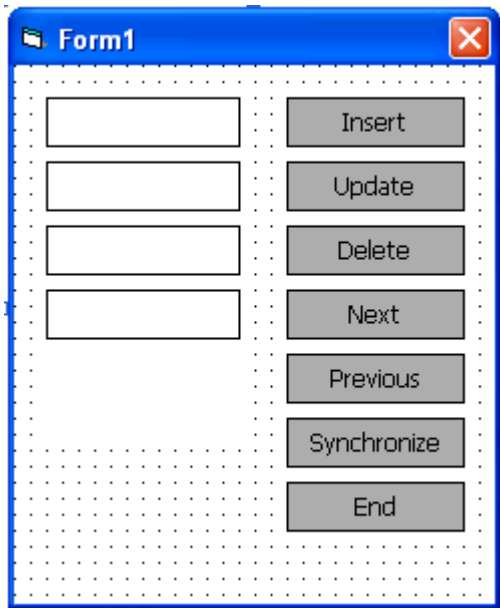
Lesson 3: Write the sample code

This lesson guides you through writing 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 a SQL Anywhere database. This portion of the lesson is optional, and requires SQL Anywhere.

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 your *tutcustomer* database. This method is recommended. Alternatively, the database connection can be established by providing connection parameters directly as follows:

```
Connection = DatabaseMgr.OpenConnection("DBF=c:\tutorial\crossfire
\tutCustomer.udb")
```

♦ Write code to connect to the UltraLite database

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

2. Declare the required UltraLite objects.

   Immediately after the line `Public NonInheritable Class CrossfireForm1 Inherits System.Windows.Forms.Form` enter the following code:

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

3. Specify the connection parameters.

   ♦ Select the ULConnectionParm1 control. In the Properties window, specify connection properties for this database.

   If you are deploying to a Windows CE device, specify the following properties:

Property	Value
DatabaseOnCE	\tutorial\crossfire\tutCustomer.udb
DatabaseOnDesktop	c:\tutorial\crossfire\tutCustomer.udb

   If you are deploying to a Palm device, specify the following properties:

Property	Value
DatabaseOnDesktop	c:\tutorial\crossfire\tutCustomer.pdb

Property	Value
DatabaseOnPalm	Syb3

4. In the Form Load event CrossfireForm1_Load, add code to connect to the database.

   The standard way of connecting is to try open a connection to the database specified by the connection string. If the database does not exist, generate an error message.

```
Try
    Connection = _
        DatabaseMgr.OpenConnection(ULConnectionParms1.ToString())
Catch
    MsgBox(Err.Number)
    MsgBox(Err.Description)
End Try
```

5. Add the following code to the click event of the End button (btnDone):

```
Connection.Close
End
```

6. Run the application.

   ♦ Choose Debug ► Start.

   ♦ After an initial message box, the form loads.

   ♦ To terminate the application, click End.

Troubleshooting

You now have the basic code in place to connect to your database.

If you see a data type conversion error on the attempt to connect, make sure you have used the ToString method on the ULConnectionParms1 object.

Write code for navigation and data manipulation

The following procedures implement data manipulation and navigation. The code uses the Table API, which provides methods for moving through and changing the rows of a table, one row at a time. For more complex applications, UltraLite provides an implementation of SQL.

♦ **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 Form1_Load event, just before the End Sub instruction.

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

```
      CustomerTable.Open()
   Catch
      If Err.Number <> UltraLiteAFLib.ULSQLCode.ulSQLE_NOERROR _
      Then
         MsgBox(Err.Description)
      End If
   End Try
```

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.Column("ID").StringValue
      txtFname.Text = _
           CustomerTable.Column("FName").StringValue
      txtLname.Text = _
           CustomerTable.Column("LName").StringValue
      If CustomerTable.Column ("City").IsNull Then
         txtCity.text =""
      Else
         txtCity.Text = _
         CustomerTable.Column("City").StringValue

      End If
      If CustomerTable.Column("Phone").IsNull Then
         txtphone.Text = ""
      Else
         txtphone.Text = _
         CustomerTable.Column("Phone").StringValue
      End If
   End If
End Sub
```

3. Call DisplayCurrentRow from the Form's Activated event. This call ensures that the fields get updated when the application starts.

```
DisplayCurrentRow
```

♦ **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 Click event of the Insert button (btnInsert).

```
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


Try
    CustomerTable.InsertBegin
    CustomerTable.Column("FName").StringValue = _
      fname
    CustomerTable.Column("LName").StringValue = _
      lname
    If Len(city) > 0 Then
        CustomerTable.Column("City").StringValue = _
      city

    End If
    If Len(phone) > 0 Then
        CustomerTable.Column("Phone").StringValue = _
      phone
    End If
    CustomerTable.Insert
    CustomerTable.MoveLast
    DisplayCurrentRow
    Exit Sub
Catch
    MsgBox "Error:  " & CStr(Err.Description)
End Try
```

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 End 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 code to the Click event of the Next button (btnNext).

    ```
    If Not CustomerTable.MoveNext Then
        CustomerTable.MoveLast
    End If
    DisplayCurrentRow
    ```

Add the following code to the Click event of the Previous button (btnPrevious).

```
If Not CustomerTable.MovePrevious Then
    CustomerTable.MoveFirst
End If
DisplayCurrentRow
```

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.

At this stage you can enter data and scroll 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 code to the Click event of the Update button (btnUpdate):

```
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

Try
    CustomerTable.UpdateBegin
    CustomerTable.Column("FName").StringValue = fname
    CustomerTable.Column("LName").StringValue = lname
    If Len(city) > 0 Then
        CustomerTable.Column("City").StringValue = city
    Else
        CustomerTable.Column("City").SetNull
    End If

    If Len(phone) > 0 Then
        CustomerTable.Column("Phone").StringValue = phone
    End If
    CustomerTable.Update
    DisplayCurrentRow
    Exit Sub
Catch
    MsgBox "Error:  " & CStr(Err.Description)
End Try
```

2. Write code to implement the Delete button.

In the code below, the call to Delete deletes the current row (the application displays the row data at the current position).

Add the following code to the Click event of the Delete button (btnDelete):

```
If CustomerTable.RowCount = 0 Then
    Exit Sub
End If
CustomerTable.Delete
CustomerTable.MoveRelative 0
DisplayCurrentRow
```

3. Run the application.

Write code to synchronize

The following procedure implements synchronization. Synchronization requires SQL Anywhere.

♦ **To write code for the synchronize button**

•   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.

Add the following code to the Click event of the Synchronize button (btnSync):

```
With Connection.SyncParms
    .UserName = "CrossfireSample"
    .Stream = UltraLiteAFLib.ULStreamType.ulTCPIP
    .Version = "ul_default"
End With
Connection.Synchronize
DisplayCurrentRow
```

Synchronize your application

The SQL Anywhere sample database has a Customers table with columns matching those in the **Customer** table in your UltraLite database. The following procedure synchronizes your database with the SQL Anywhere sample database.

♦ **To synchronize your application**

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

    ```
    mlsrv10 -c "dsn=SQL Anywhere 10 Demo" -v+ -zu+
    ```

    The -zu+ command line option permits automatic addition of users and generation of synchronization scripts. For more information about these options, see "MobiLink Server Options" [*MobiLink - Server Administration*].

    Verify that the MobiLink server starts and displays a server status window.

2. Start your UltraLite Crossfire application.

3. Click Delete repeatedly to delete all the rows in your table. Any rows left in the table would be uploaded to the Customers table in the SQL Anywhere sample database.

4. Synchronize your application.

    Click Synchronize.

    The MobiLink server window displays the synchronization progress.

5. When the synchronization is complete, click Next and Previous buttons to move through the rows of the table to view the data retrieved from the SQL Anywhere sample database.

Lesson 4: Deploy to a device

The following procedures deploy your application to either a Palm OS or Windows CE OS based device.

♦ **To deploy to a Windows CE OS based device**

1. Configure the application settings.

    ♦ From the AppForge menu, choose Crossfire Settings. A dialog appears.

    ♦ In the left pane, choose Dependencies and click the User Dependencies tab.

    ♦ Click the Add button and select the *c:\tutorial\crossfire\tutCustomer.udb*. This indicates to Crossfire that the database file must be included in the deployment.

    ♦ Choose the Windows Mobile Settings item in the left pane and in the right panel select the Packaging tab.

    ♦ Enter *%CE1%\tutorial\crossfire* for the Custom Device Installation Path.

    ♦ Click OK to close the dialog.

2. From the AppForge menu, choose Deploy Application to Device, and select Windows Mobile-based Pocket PC. If a dialog appears asking if you want to save the project, choose Yes.

3. On your device, click Start ► Programs.

4. Click UltraLiteTutorial to run your application.

♦ **To deploy to the Palm device**

1. Configure the application settings.

    ♦ From the AppForge menu, choose Crossfire Settings.

    ♦ In the dialog that appears, choose Dependencies in the left pane and click the User Dependencies tab.

    ♦ Click the Add button and select *c:\tutorial\mvb\tutCustomer.pdb*. This indicates to Crossfire that the file should be included in the deployment.

    ♦ Choose the Palm OS Settings item in the left pane and enter Syb3 for the creator ID. Select a valid HotSync name.

    ♦ Click OK to close the dialog.

2. From the AppForge menu, choose Deploy Application to Device, and select the Palm OS device. If a dialog appears asking if you want to save the project, choose Yes.

3. HotSync your device and ensure the application gets sent to the device. After the HotSync process is complete, your application files will be extracted on the device.

4. Click Home on the device and choose UltraLiteTutorial to run your application.

Summary

Learning accomplishments

During this tutorial, you:

♦ created a database file with one table defined

♦ created an UltraLite application for Crossfire

♦ synchronized an UltraLite remote database with a SQL Anywhere consolidated database

More samples

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

CHAPTER 4

Tutorial: A Sample Application for AppForge MobileVB

Contents

Introduction to MobileVB development tutorial

This tutorial guides you through the process of building an UltraLite application for MobileVB using the UltraLite 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. The target platform for the application is Windows CE or Palm OS.

For more information about the table API, see the "UltraLite for AppForge API Reference" on page 71.

Timing

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

If you copy the code literally from this help file, be aware that some characters may not be copied correctly. The ampersand, less-than, and greater-than symbols may be copied as html markup codes and have to be manually repaired in the Visual Basic code edit window.

Competencies and experience

This tutorial assumes:

♦ you have MobileVB and Microsoft Visual Basic 6 installed on your computer

♦ you can write, test, and troubleshoot a MobileVB application

♦ you know how to create an UltraLite database using the UltraLite plug-in for Sybase Central or using the ulcreate utility

♦ you have the Crossfire Client installed on the target device.

    Information about the Crossfire Client is available from the AppForge Web site.

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

Goals

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

Lesson 1: Create project architecture

The first lesson establishes the location for files in the project and the specifications for an UltraLite database for the project.

♦ **To create an UltraLite database**

1. Create a directory for this tutorial.

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

2. Create a database using UltraLite in Sybase Central.

   See "Creating an UltraLite database from Sybase Central" [*UltraLite - Database Management and Reference*].

   ♦ **Database file name**   *c:\Tutorial\mvb\tutcustomer.udb*

   ♦ **Table name**   ULCustomer

   ♦ **Columns in ULCustomer table**

Col- umn Name	Data Type (Size)	Column allows null values?	Column unique value?	Default value
cust_id	integer	No	n/a	autoincrement
cust_na me	varchar (30)	No	No	None

   ♦ **Primary key for ULCustomer table**   ascending cust_id

Create a MobileVB project

The following procedure creates a MobileVB project for your application and adds a reference to the UltraLite for MobileVB control.

MobileVB tools appear in addition to the standard Visual Basic tools on the Visual Basic toolbar to the left of the Visual Basic environment. If the UltraLite connection parameters control is not present, see "Adding UltraLite to the MobileVB design environment" on page 6

♦ **To create a reference to the UltraLite for MobileVB control**

1. Start MobileVB.

   ♦ Choose Start ► Programs ► AppForge MobileVB ► Start MobileVB.

   The MobileVB Project Manager appears.

2. Create a new project.

    Click New Project. When asked for the Design Target, choose the appropriate target. This tutorial provides instructions for a Windows CE OS based device.

3. Create a reference to UltraLite for MobileVB.

    ♦   Choose Project ► References

    ♦   Select iAnywhere Solutions, UltraLite Connection Parameters 10.0 and click OK.

4. Give the Project a name.

    ♦   Click Project ► MobileVBProject1 Properties

    ♦   Under Project Name, type **UltraLiteTutorial**. This is the name of the application as it will appear on your device.

    ♦   Click OK.

5. Save the Project:

    ♦   Choose File ► Save Project.

    ♦   Save the form file as *c:\tutorial\mvb\Tutorial.frm*.

    ♦   Save the project as *c:\tutorial\mvb\Tutorial.vbp*.

Lesson 2: Create a form

The project should display a form after completing the steps in "Lesson 1: Create project architecture" on page 59.

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
AFTextBox	txtname	
AFLabel	lblID	
AFButton	btnInsert	Insert
AFButton	btnUpdate	Update
AFButton	btnDelete	Delete
AFButton	btnNext	Next
AFButton	btnPrevious	Previous
AFButton	btnSync	Synchronize
AFButton	btnDone	End

2. Check the application:

♦ Choose MobileVB ► Compile and Validate.

Your form should look like the following screen capture:
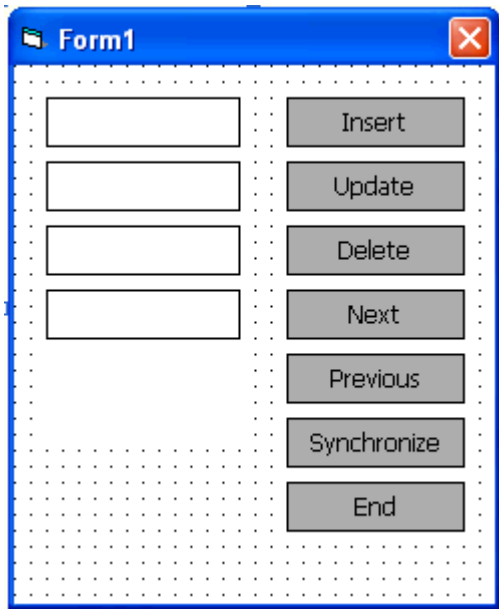
Lesson 3: Write the sample code

This lesson guides you through the process of writing 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 a SQL Anywhere database. The synchronization portion of the lesson is optional, and requires SQL Anywhere.

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. Alternatively, you can use a connection string in the application code.

See "ULConnectionParms class" on page 88.

♦ **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 declarations area of your form.

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

3. Specify the connection parameters:

   ♦ Add a ULConnectionParms object to your form named ULConnectionParms1. The ULConnectionParms control is a database icon on the toolbox.

   ♦ In the Properties window, specify the ULConnectionParms properties.

      If you are deploying to a Windows CE device, specify the following properties:

Property	Value
DatabaseOnCE	\tutorial\mvb\tutcustomer.udb
DatabaseOnDesktop	c:\tutorial\mvb\tutcustomer.udb

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

   The database manager opens a connection to the database specified by the ULConnectionParms1 object:

   ```
   Private Sub Form_Load()
   ' enable error handling
   On Error Resume Next
   ```

```
    Set Connection = DatabaseMgr.OpenConnection(ULConnectionParms1.ToString
())

    If Err.Number = ULSQLCode.ulSQLE_NOERROR Then
      MsgBox "Connected to an existing database"
    Else
      MsgBox Err.Description
      Exit Sub
    End If
  End Sub
```

Once the connection code is working, the line that issues a Msgbox to indicate a connection has been made can be removed.

If you prefer to use a connection string rather than the ULConnectionParms object, you can alter the code illustrated above to use this syntax instead:

```
  Set Connection = DatabaseMgr.OpenConnection _
      ("dbf=C:\tutorial\mvb\tutcustomer.udb;" & _
       "ce_file=\tutorial\mvb\tutcustomer.udb")
```

Note the inclusion of the database file name specification for the potential target platforms (dbf= for the desktop environment and ce_file= for the Windows CE device environment.

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

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

6. Run the application.

    ♦   Choose Run ► Execute.

    ♦   After an initial message box, the form loads.

    ♦   To terminate the application, click the End button.

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("ULCustomer")
```

```
CustomerTable.Open
If Err.Number <> ULSQLCode.ulSQLE_NOERROR Then
    MsgBox Err.Description
End If
CustomerTable.MoveFirst
```

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
    txtname.Text = ""
    lblID.Caption = ""
  Else
    txtname.Text = CustomerTable.Column("cust_name").StringValue
    lblID.Caption = CustomerTable.Column("cust_id").StringValue
  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()

    On Error GoTo InsertError
    CustomerTable.InsertBegin
    CustomerTable.Column("cust_name").StringValue = txtname.Text

    CustomerTable.Insert
    CustomerTable.MoveLast
    DisplayCurrentRow
    Exit Sub

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

    On Error GoTo UpdateError

    CustomerTable.UpdateBegin
    CustomerTable.Column("cust_name").StringValue = txtname.Text
    CustomerTable.Update
    DisplayCurrentRow
```

```
        Exit Sub

    UpdateError:
        MsgBox "Error:  " & CStr(Err.Description)
    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
        txtname.Text = ""
        lblID.Caption = ""
        DisplayCurrentRow
    End Sub
```

3. Run the application.

Write code to synchronize

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

♦ To write code for the synchronize button

•   Write code to implement the Synchronize button.

    In the code below, the ULSyncParms object contains synchronization parameters. For example, the UserName property specifies that when MobiLink is started, it uses the specified user name to determine the proper set of MobiLink scripts to employ for the synchronization process. The DownloadOnly property is set to true (in this program) to ensure that no data is uploaded from the UltraLite database since this is a simple demonstration application.

    For more information about synchronization parameters, see "UltraLite Synchronization Parameters and Network Protocol options" [*MobiLink - Client Administration*].

    Add the following procedure to the form:

```
    Private Sub btnSync_Click()
        With Connection.SyncParms
            .UserName = "50"
            .Stream = ULStreamType.ulTCPIP
            .Version = "custdb 10.0"
            .DownloadOnly = True
        End With
        Connection.Synchronize
        CustomerTable.MoveFirst
```

```
        DisplayCurrentRow
End Sub
```

Synchronize your application

The SQL Anywhere 10 CustDB database sample supplied with SQL Anywhere has a Customer table with columns matching those in the **ULCustomer** table in your UltraLite database. The following procedure synchronizes your database with the SQL Anywhere 10 CustDB database.

♦ To synchronize your application

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

    ```
    mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -v+ -zu+
    ```

    The −v+ option turns on verbose logging (+ means "show all"). Verbose logging is recommended during application debugging. The −zu+ option provides automatic addition of users. See "MobiLink Server Options" [*MobiLink - Server Administration*].

2. Start the UltraLite MobileVB application.

3. Synchronize your application.

    Click Synchronize.

    The MobiLink server window displays the synchronization progress.

4. When the synchronization is complete, click Next and Previous to move through the rows of the table to verify that new information has been downloaded from the SQL Anywhere 10 CustDB database.

Lesson 4: Deploy to a device

The following procedure describes how to deploy your application to a Windows CE OS based device.

♦ **To deploy to a Windows CE OS based device:**

1. Configure the application settings:

   ♦ From the AppForge menu, choose MobileVB Settings.

      A dialog appears.

   ♦ In the left pane, choose Dependencies and click the User Dependencies tab.

   ♦ Click the Add button and select *c:\tutorial\mvb\tutcustomer.udb*. This indicates to MobileVB that the file should be included in the deployment.

   ♦ Choose the Windows Mobile Settings item in the left pane

   ♦ Enter *\tutorial\mvb* for the Custom path in Device Installation Path.

   ♦ Click OK to close the dialog.

2. From the AppForge menu, choose Deploy Application to Device, and select the Windows Mobile-based Pocket PC device. If a dialog appears asking if you want to save the project, choose Yes.

3. On your device, click Start ► Programs.

4. Click ULTutorial to run your application.

Summary

Learning accomplishments

During this tutorial, you:

♦ created an UltraLite database

♦ created an UltraLite for MobileVB application

♦ synchronized an UltraLite remote database with a SQL Anywhere consolidated database

♦ gained competence with the process of developing an UltraLite for MobileVB application

More samples

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

UltraLite for AppForge API Reference

Contents

ULAuthStatusCode enumeration

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

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

ULColumn class

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

Note that get methods throw an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method before attempting a get.

> **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 more information about the table object, see "ULTable class" on page 149.

Properties

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

AppendByteChunk method

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

Syntax
AppendByteChunk(_
  data As Long**,** _
  data_len As Long _

)
Member of **UltraLiteAFLib.ULColumn**

Parameters

data   In MobileVB, a pointer to an array of Bytes. To get the pointer to the array of bytes, use the Visual Basic VarPtr() function. In Crossfire, a local variable that is an array of Bytes.

data_len   The number of bytes from the array to append.

Errors set

uISQLE_INVALID_PARAMETER   This error occurs if data length is less than 0.

uISQLE_CONVERSION_ERROR   This error occurs if the column data type is not LONG BINARY.

Example

The following examples append data to the edata column.

```
'MobileVB using VB6
Dim data (1 to 512) As Byte
' ...
table.Column("edata").AppendByteChunk( _
                VarPtr(data(1)), 232)

'Crossfire using vb.net
Dim data (1 to 512) As Byte
' ...
table.Column("edata").AppendByteChunk(data, 232)
```

AppendStringChunk method

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

Syntax

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

Parameters

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

Errors set

uISQLE_CONVERSION_ERROR   This error occurs if the column data type is not CHAR or LONG VARCHAR.

GetByteChunk method

Gets data from a TypeBinary or TypeLongBinary column.

Syntax

GetByteChunk (_
  offset As Long**, _**

```
      data As Long, _
      data_len As Long, _
      filled_len As Long _
   ) As Boolean
   Member of UltraLiteAFLib.ULColumn
```

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 is raised.

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

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

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

Returns

True if this column value contains more data.

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

Errors set

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

ulSQLE_INVALID_PARAMETER   This 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.

```
'MobileVB using VB6
Dim filled As Long
Dim more_data As Boolean
Dim data (1 to 512) As Byte
more_data = table.Column("edata").GetByteChunk(0, _
VarPtr(data(1)), 512, filled)

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

GetStringChunk method

Gets data from a TypeString or TypeLongString column.

Syntax

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

Parameters

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

data   The variable to receive the string data.

string_length   The length of the String you want returned.

filled_len   The length of the String fetched.

Returns

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

False if there is no more data.

Errors

ulSQLE_CONVERSION_ERROR   This error occurs if the column data type is not CHAR or LONG VARCHAR.

ulSQLE_INVALID_PARAMETER   This 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 0 or string length is less than 0.

SetByteChunk method

Sets data in a TypeBinary or TypeLongBinary column.

Syntax

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

Parameters

data   In MobileVB, a pointer to an array of Bytes. To get the pointer to the array of bytes, use the Visual Basic VarPtr() function. In Crossfire, a local variable that is an array of Bytes.

length   The length of the array.

Errors set

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

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

See also

♦ "AppendByteChunk method" on page 73

Example

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

```
'MobileVB using VB6
Dim data (1 to 512) As Byte
' ...
table.Column("edata").SetByteChunk( VarPtr(data(1)), 232)

'Crossfire using vb.net
Dim data (1 to 512) As Byte
' ...
table.Column("edata").SetByteChunk( data, 232)
```

SetNull method

Sets the column value to null.

Syntax

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

SetToDefault method

Sets the current column to its default value as defined by the database schema. For example, an autoincrement column is assigned the next available value.

Syntax

SetToDefault( )
Member of **UltraLiteAFLib.ULColumn**

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

Syntax	Description
AutoIncrement As Boolean (read-only)	Indicates whether this column defaults to an autoincrement value. True if AutoIncrement.
CurrentDate As Boolean (read-only)	Indicates whether this column defaults to the current date.
CurrentTime As Boolean (read-only)	Indicates whether this column defaults to the current time.
CurrentTimestamp As Boolean (read-only)	Indicates whether this column defaults to the current timestamp.
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.
GlobalAutoIncrementPartitionSize As Long (read-only)	Gets the partition size for a global autoincrement column.
ID As Integer (read- only)	Gets the ID of the column.
Name As String (read-only)	Gets the column name.
NewUUID As Boolean (read-only)	Indicates whether this column defaults to a new universally unique identifier.
Nullable As Boolean (read-only)	Indicates whether the column permits NULLs.
OptimalIndex As ULIndexSchema (read-only)	Gets the index with this column as its first column.
Precision As Integer (read-only)	Gets the precision value for the column if it is of type ulTypeNumeric.
Scale As Integer (read-only)	Gets the scale value for the column if it is of type ulTypeNumeric.
Size As Integer (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 to synchronize.

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

Example

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

```
Public WithEvents Connection As ULConnection
```

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

Properties

The following are properties of ULConnection:

Syntax	Description
AutoCommit As Boolean	Indicates the AutoCommit value. If true, all data changes are committed immediately after they are made. Otherwise, changes are not committed to the database until Commit is called. By default, this property is True.
DatabaseID As Long	Identification number of the database; -1 if not set.
GlobalAutoIncrementUsage As Integer (read-only)	Gets the percentage of available global autoincrement values that have been used.
LastIdentity As Long (read-only)	Gets the most recent value inserted into a column with a default of autoincrement or global autoincrement.
OpenParms As String (read-only)	Gets the string used to open the connection to the database.
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.

Syntax	Description
SyncResult As ULSyncResult (read-only)	Gets the results of the most recent synchronization.

CancelSynchronize method

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

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

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

ChangeEncryptionKey method

Encrypt the database with the specified key.

Syntax

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

Parameters
newkey   The new encryption key value for the database.

Example
When you call CreateDatabase 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 OpenConnection, must, after the database is encrypted, specify *apricot* as the EncryptionKey property too. Otherwise, the connection fails.

Close method

Closes the connection to the database.

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

Remarks

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 is implicitly closed when the application terminates.

Commit method

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

Syntax

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

See also

♦ "Properties" on page 79

CountUploadRows method

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

Syntax

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

Parameters

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

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

Returns

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

GetNewUUID method

Returns a new universally unique identifier.

Syntax

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

Remarks

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

Returns the **ULTable** object for the specified table.

Syntax

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

Parameters

name   The name of the table sought.

Remarks

You must then open the table before your application can read data from it.

Returns

Returns the ULTable object.

GrantConnectTo method

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

Syntax

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

Parameters

userid   The user ID being granted authority to connect.

password   The password the user ID must specify to connect.

LastDownloadTime method

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

Syntax

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

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

Prepares a SQL statement for execution.

Syntax

> **PrepareStatement(**
>   *sqlStatement* As String**,** _
>   *persistent_name* As String _
> **)** As ULPreparedStatement
> Member of **UltraLiteAFLib.ULConnection**

Parameters

> **sqlStatement**   The SQL statement to prepare.

> **persistent_name**   For Palm applications, the persistent name of the statement.

Returns

> Returns a ULPreparedStatement. If there was a problem preparing the statement, an error is raised. You can determine the offset of the statement where the error occurred with the SQLErrorOffset property.

ResetLastDownloadTime method

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

Syntax

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

Parameters

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

RevokeConnectFrom method

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

Syntax

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

Parameters

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

Rollback method

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

Syntax

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

RollbackPartialDownload method

Rolls back the changes from a failed synchronization.

Syntax

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

Remarks

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 required (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 - Server Administration*]
♦   "Keep Partial Download synchronization parameter" [*MobiLink - Client Administration*]
♦   "Partial Download Retained synchronization parameter" [*MobiLink - Client Administration*]
♦   "Resume Partial Download synchronization parameter" [*MobiLink - Client Administration*]

StartSynchronizationDelete method

Allows delete operations to be synchronized with the consolidated database.

Syntax

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

Description

Permits delete operations to be synchronized with the consolidated database. This method is used to resume synchronization of delete operations after they had been prevented by StopSynchronizationDelete.

See also

♦ "StopSynchronizationDelete method" on page 85

StopSynchronizationDelete method

Prevents delete operations from being synchronized to the consolidated database.

Syntax

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

Remarks

This method is useful for deleting old information from an UltraLite database to save space; however it does not delete information on the consolidated database.

See also

♦ "StartSynchronizationDelete method" on page 84

StringToUUID method

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

Syntax

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

Parameters

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

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

Remarks

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

The pointer to the buffer   The pointer to the buffer must be declared as at least16 bytes. Since Visual Basic does not provide bounds checking, memory could be overwritten if the buffer is too small. In MobileVB, use the VarPtr() function to get the pointer to the buffer. See also ULColumn.UUIDValue property.

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.

See also

♦ "UNIQUEIDENTIFIER data type" [*SQL Anywhere Server - SQL Reference*]

Example

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

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

Synchronize method

Synchronizes a consolidated database using MobiLink.

Syntax

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

Remarks

This method does not return until synchronization is complete.

You can be notified of events. Simply declare the connection WithEvents.

See also

♦ "ULConnection class" on page 79

UUIDToString method

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

Syntax

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

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*.

Remarks

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.

See also

♦ "UNIQUEIDENTIFIER data type" [*SQL Anywhere Server - SQL Reference*]

ULConnectionParms class

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

Properties

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

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

> **Note**
> Databases are created with a single authenticated user, DBA, whose initial password is sql. If a user ID and password are not supplied, connections are opened using the user ID DBA and password sql.

For more information about the meaning of these parameters, see "UltraLite Connection String Parameters Reference" [*UltraLite - Database Management and Reference*].

Syntax	Description
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. See "UltraLite CACHE_SIZE connection parameter" [*UltraLite - Database Management and Reference*].
ConnectionName As String (read-write)	A name for the connection. This is needed only if you create more than one connection to the database. See "UltraLite CON connection parameter" [*UltraLite - Database Management and Reference*].
DatabaseOnCE As String (read-write)	The file name of the database deployed to Windows CE. See "UltraLite CE_FILE connection parameter" [*UltraLite - Database Management and Reference*].
DatabaseOnDesktop As String (read-write)	The file name of the database during development. See "UltraLite DBF connection parameter" [*UltraLite - Database Management and Reference*].
DatabaseOnPalm As String (read-write)	The creator ID for the UltraLite database on the Palm device. See "UltraLite PALM_FILE connection parameter" [*UltraLite - Database Management and Reference*].

Syntax	Description
DatabaseOnSymbian As String (read-write)	The file name of the database on Symbian OS device. See "UltraLite SYMBIAN_FILE connection parameter" [*UltraLite - Database Management and Reference*].
EncryptionKey As String (read-write)	A key for encrypting the database. OpenConnection must use the same key as specified during database creation. Recommended guidelines for keys are: 1. Select an arbitrary, lengthy string 2. Select strings with a variety of numbers, letters and special characters, to decrease the chances of successfully guessing the key. See "UltraLite DBKEY connection parameter" [*UltraLite - Database Management and Reference*].
PalmCreatorID As String (read-write)	A registered four digit Palm creator ID.
Password As String (read-write)	The password for an authenticated user. Databases are initially created with one authenticated user (DBA) with password *sql*. Passwords are case sensitive. See "UltraLite PWD connection parameter" [*UltraLite - Database Management and Reference*].
ToString As String	The connection string, as built from the current property settings.
UserID As String (read-write)	The authenticated user for the database. Databases are initially created with one authenticated user DBA. The UserID is case insensitive if the database is case insensitive and case sensitive if the database is case sensitive. The default value is *DBA*. See "UltraLite UID connection parameter" [*UltraLite - Database Management and Reference*].

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 recommeneded that you use CreateDatabase, OpenConnection and DropDatabase, and include checks in your code to ensure that you are connected properly before attempting any DML with the database.

Properties

The following are properties of ULDatabaseManager:

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

CreateDatabase method

Create a new UltraLite database using specified creation parameters, collation sequence, and connection parameters.

Syntax

CreateDatabase(*connparms* As string **,**
collation As Long **,**
createparms As String
) As ULConnection
Member of **UltraLiteAFLib.ULDatabaseManager**

Parameters

connparms   Access parameters for the database. Similar to parameters supplied to OpenConnection.

collation   A pointer to an array of bytes representing a collation. The collation byte array can be initialized in Visual Basic .NET for AppForge by using predefined collations located in the folder *install-dir\src \ulcollations\af.vb.net*.

createparms   Database creation parameters—database characteristics that can only be specified at database creation; examples: obfuscation and page-size.

See "Choosing creation-time database properties for UltraLite" [*UltraLite - Database Management and Reference*].

Returns

Returns a connection if the database is successfully created.

DropDatabase method

Deletes the database file.

Syntax

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

Parameters

parms   The file name for the database.

Remarks

If you drop the databases, all information in the database file is permanently deleted.

This method fails if:

♦   The database you specify does not exist.

♦   There are open connections when DropDatabase is executed.

Example

The following example drops a database:

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

OpenConnection method

Opens a connection to a named database.

Syntax

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

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

See "Opening UltraLite connections with connection strings" [*UltraLite - Database Management and Reference*].

Remarks

Use this method to connect to a database that is already created.

This method fails if:

♦ The database does not exist.

♦ The connection parameters are invalid.

To determine why the call failed, use the error object.

Example

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

```
Set Connection = DatabaseMgr.OpenConnection("UID=JDoe;PWD=ULdb;
NT_FILE=c:\test\MyTestDB.udb;CE_FILE=\database
\MyCEDB.udb;PALM_FILE=MyPalmDB_MyCreatorID")
```

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:

Syntax	Description
CollationName As String (read-only)	Gets the database collation sequence name.
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 database.
DateOrder As String (read-only)	Indicates the interpretation of date formats; valid values are 'MDY', 'YMD', or 'DMY'.
IsCaseSensitive As Boolean (read-only)	Indicates whether the database is case sensitive.
NearestCentury As String (read-only)	Indicates the interpretation of two-digit years in string-to-date conversions. This is a numeric value that acts as a rollover point. Two digit years less than the value are converted to 20yy, while years greater than or equal to the value are converted to 19yy. The default is 50.
Precision As String (read-only)	Gets the maximum number of digits in the result of any decimal arithmetic.
PublicationCount As Integer (read-only)	Gets the number of publications in the connected database.
TableCount As Integer (read-only)	Gets the number of tables in the connected database.
TimeFormat As String (read-only)	Gets the format for times retrieved from the database.
TimestampFormat As String (read-only)	Gets the format for timestamps retrieved from the database.

GetDatabaseProperty method

Returns the value of the named database property.

Syntax

GetDatabaseProperty(*property_name* As String **)** As
String Member of **UltraLiteAFLib.ULDatabaseSchema**

Parameters

property_name   The *property_name* specifies the property name which is being queried.

Following is a list of property name strings. For more information about the meaning of the properties, see "UltraLite Database Settings Reference" [*UltraLite - Database Management and Reference*].

♦ CaseSensitive
♦ CharSet
♦ ChecksumLevel
♦ CollationName
♦ ConnCount
♦ date_format
♦ date_order
♦ Encryption
♦ File
♦ global_database_id
♦ MaxHashSize
♦ ml_remote_id
♦ Name
♦ nearest_century
♦ PageSize
♦ precision
♦ scale
♦ time_format
♦ timestamp_format
♦ timestamp_increment

Returns

Returns the named property value.

GetPublicationName method

Returns the name of the specified publication.

Syntax

GetPublicationName(*id* As Integer **)** As String
Member of **UltraLiteAFLib.ULDatabaseSchema**

Parameters

id   The *id* is the identifier of the publication whose name is returned. The publication *ID* can range from 1 to PublicationCount.

Returns

Returns the name of a publication in the connected database.

For more information about the ULPublicationSchema object, see "ULPublicationSchema class" on page 109.

GetPublicationSchema method

Uses the publication name to retrieve the ULPublicationSchema object.

Syntax

GetPublicationSchema(*Name* As String **)** As ULPublicationSchema
Member of **UltraLiteAFLib.ULDatabaseSchema**

Parameters

name   The *name* of the publication.

Returns

Returns the ULPublicationSchema object.

GetTableName method

Returns the name of the table in the connected database that corresponds to the *id* value you supply.

Syntax

GetTableName(*id* As Integer **)** As String
Member of **UltraLiteAFLib.ULDatabaseSchema**

Parameters

id   The *id* of the table.

Returns

Returns the name of the table for the specified *id*.

Remarks

The TableCount property returns the number of tables in the connected database. Each table has a unique number from 1 to the TableCount value, where 1 is the first table in the database, 2 is the second table in the database, and so on. The id for a table may change if a database's schema changes.

SetDatabaseOption method

Sets the value for a database option.

Syntax

SetDatabaseOption(
option_name As String
option_value As String
)

Parameters

option_name   The name of the database option to be set. The following database options may be set: global_database_id and ml_remote_id. See:

♦ "UltraLite global_database_id option" [*UltraLite - Database Management and Reference*]
♦ "UltraLite ml_remote_id option" [*UltraLite - Database Management and Reference*]

option_value   The new value for the option.

ULFileTransfer class

The interface to the MobiLink file transfer feature.

Properties

The following are properties of ULFileTransfer:

Syntax	Description
AuthStatus As ULAuthStatusCode	The authorization status code for the last file transfer.
AuthValue As Integer	The authorization value for the last file transfer.
DestinationFile As String	Name of downloaded file within DestinationPath.
DestinationPath As String	Client location of downloaded file.
DowloadedFile As Boolean	Set to true if file was downloaded.
FileAuthCode As Integer	The authorization code for the last file transfer.
FileName As String	Remote file name to download.
ForceDownload As Boolean	If true, download file unconditionally.
Password As String	Password for synchronization user name specified in User-Name property.
ResumePartialDownload As Boolean	If set to true, resume a previously kept partial download.
Stream As ULStreamType	Type of stream to use in synchronization.
StreamErrorCode As ULStreamErrorCode	The error code reported by the stream.
StreamParms As String	Additional parameters for the given stream type.
StreamErrorSystem As Long	The stream error system-specific code.
UserName As String	User name to submit to MobiLink during synchronization.
Version As String	The version of the synchronization scripts to run.

See also

♦ "MLFileTransfer function" [*UltraLite - C and C++ Programming*]
♦ "MobiLink file transfer utility [mlfiletransfer]" [*MobiLink - Client Administration*]

AddAuthenticationParm method

Adds an authentication parameter to the list of custom authentication parameters.

Syntax

AddAuthenticationParm(*newParm* As String **)**
As Boolean
　　Member of **UltraLiteAFLib.ULFileTransfer**

Parameters

newParm　The authentication parameter to add.

CancelTransfer method

Cancels an in-progress file transfer started by DownloadFile method.

Syntax

CancelTransfer() Member of **UltraLiteAFLib.ULFileTransfer**

ClearAuthenticationParms method

Clears the list of custom authentication parameters.

Syntax

ClearAuthenticationParms()
　　Member of **UltraLiteAFLib.ULFileTransfer**

DownloadFile method

Download a file using MobiLink file transfer, using current properties.

Syntax

DownloadFile()
　As Boolean
　　Member of **UltraLiteAFLib.ULFileTransfer**

Returns

Returns true if file downloaded successfully or already existed locally; false otherwise. The property ULFileTransfer.DownloadedFile can be checked to see if a file transfer actually occurred.

Errors set

ulSQLE_COMMMUNICATIONS_ERROR　This error occurs if no stream was specified.

See also

♦　"MLFileTransfer function" [*UltraLite - C and C++ Programming*]

　　　Copyright © 2007, iAnywhere Solutions, Inc.

♦ "MobiLink file transfer utility [mlfiletransfer]" [*MobiLink - Client Administration*]

ULFileTransferEvent class

The ULFileTransferEvent object provides a mechanism to signal events during MobiLink file transfer.

See "ULFileTransfer class" on page 97.

OnTransferProgressChanged method

Notifies your application that the current file transfer has transferred data.

Syntax

OnTransferProgressChanged(
file_size As Long **,**
bytes_received As Long **,**
resumed_at_size As Long
)
      Member of **UltraLiteAFLib.ULFileTransferEvent**

Parameters

file_size   File size in bytes.

bytes_received   Bytes transferred.

resumed_at_size   Byte count where resumed transfer occurred.

Returns

Return true if transfer continues.

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 is 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

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

GetColumnName method

Returns the names of the columns in the index.

Syntax

GetColumnName(*col_pos_in_index* As Integer **)** As String
Member of **UltraLiteAFLib.ULIndexSchema**

Parameters

    col_pos_in_index   The column position in the index. *col_pos_in_index* must be at least 1 and at most ColumnCount.

Returns

    Returns the name of a column in the index.

IsColumnDescending method

    Indicates whether the specified column in the index is in descending order.

Syntax

    IsColumnDescending(*col_name* As String **)** As Boolean
        Member of **UltraLiteAFLib.ULIndexSchema**

Parameters

    col_name   The index column name.

Returns

    True if the column is descending.

    False if the column is ascending.

ULPreparedStatement class

The ULPreparedStatement represents a pre-compiled SQL statement ready for execution. You can use a 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

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

AppendByteChunkParameter method

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

Syntax

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

Parameters

parameter_id   The 1-based parameter number to set.

data   The array of bytes to append.

data_len   The number of bytes from the array to append.

Errors set

ulSQLE_INVALID_PARAMETER   This error occurs if the data length is less than 0.

ulSQLE_CONVERSION_ERROR   This error occurs if the column data type is not LONG BINARY.

AppendStringChunkParameter method

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

Syntax

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

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   This error occurs if the column data type is not LONG VARCHAR.

Close method

Frees resources associated with the ULPreparedStatement.

Syntax

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

ExecuteQuery method

Executes the query and returns a result set.

Syntax

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

Returns

A ULResultSet object. The ULResultSet is the data you requested in your SELECT statement.

For more information about describing the product of your query, see "ULResultSetSchema class" on page 123.

ExecuteStatement method

Executes the statement.

Syntax

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

Returns

> The number of rows updated.

SetBooleanParameter method

Sets the parameter to the Boolean value passed in.

Syntax

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

Parameters

> **param_number**   The 1-based parameter number to set.
>
> **param_value**   The value the parameter should receive.

SetByteChunkParameter method

Sets data in a binary or long binary column.

Syntax

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

Parameters

> **param_number**   The 1-based parameter number to set.
>
> **data**   An array of bytes.
>
> **data_len**   The number of bytes from the array to set. SetByteChunk writes over the current content.
>
> For more information about appending to an existing value, see "AppendByteChunkParameter method" on page 103.

SetByteParameter method

Sets the parameter to the specified Byte value.

Syntax

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

Parameters

param_number   The 1-based parameter number to set.

param_value   The value the parameter should receive.

SetDatetimeParameter method

Sets the parameter to the specified Datetime value.

Syntax

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

Parameters

param_number   The 1-based parameter number to set.

param_value   The value the parameter should receive.

SetDoubleParameter method

Sets the parameter to the specified Double value.

Syntax

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

Parameters

param_number   The 1-based parameter number to set.

param_value   The value the parameter should receive.

SetIntegerParameter method

Sets the parameter to the specified Integer value.

Syntax

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

Parameters

param_number   The 1-based parameter number to set.

param_value   The value the parameter should receive.

SetLongParameter method

Sets the parameter to the specified Long value.

Syntax

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

Parameters

param_number   The 1-based parameter number to set.

param_value   The value the parameter should receive.

SetNullParameter method

Sets the parameter to NULL.

Syntax

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

Parameters

parameter_id   The 1-based parameter number to set.

SetRealParameter method

Sets the parameter to the specified Long value.

Syntax

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

Parameters

param_number   The 1-based parameter number to set.

param_value   The value the parameter should receive.

SetStringParameter method

Sets the parameter to the specified string.

Syntax

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

Parameters

param_number   The 1-based parameter number to set.

param_value   The value the parameter should receive.

ULPublicationSchema class

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

Properties

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

ContainsTable method

Indicates whether the specified table is part of this publication.

Syntax

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

Parameters

name   The target table name.

Returns

True if the table is in the publication.

False if the table is not in the publication.

ULResultSet class

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

ULResultSet columns are accessed using an ordinal number representing the 1-relative column number in the result set. This parameter is named *index* in the following descriptions.

> **Note**
> Get methods throw an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method before attempting a get.

Properties

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

AppendByteChunk method

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

Syntax

AppendByteChunk(
    index As Integer**,**
    data As Long**,**
    data_len As Long**)**
Member of **UltraLiteAFLib.ULResultSet**

Parameters

index   The 1-based parameter number to set.

data   The array of bytes to append.

data_len   The number of bytes from the array to append.

Errors set

    ulSQLE_INVALID_PARAMETER   This error occurs if the data length is less than 0.

    ulSQLE_CONVERSION_ERROR   This error occurs if the column data type is not LONG BINARY.

AppendStringChunk method

Appends the string to the row's column if the type is ulTypeLongString.

Syntax

    AppendStringChunk(
      index As Integer **,**
      data As String **)**
    Member of   **UltraLiteAFLib.ULResultSet**

Parameters

    index   The 1-based parameter number to set.

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

Errors set

    ulSQLE_CONVERSION_ERROR   This error occurs if the column data type is not LONG VARCHAR.

Close method

Frees all resources associated with this object.

Syntax

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

Delete method

Deletes the current row of the table.

Syntax

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

GetBoolean method

Gets column value as boolean.

Syntax

> **GetBoolean(***index* As Integer **) As Boolean
> Member of **UltraLiteAFLib.ULResultSet**

Parameters

> **index**   The 1-based ordinal in the result set.

Returns

> The value as a boolean.

Remarks

> This method throws an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method.

GetByte method

> Gets column value as byte.

Syntax

> **GetByte(** *index* As Integer **) As Byte
> Member of **UltraLiteAFLib.ULResultSet**

Parameters

> **index**   The 1-based ordinal in the result set.

Returns

> The value as a byte.

Remarks

> This method throws an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method.

GetByteChunk method

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

Syntax

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

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 is raised. A buffer bigger than 64K is also permissible.

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

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

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

Returns

    The number of bytes read.

Remarks

    This method throws an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method.

    This method is suitable for BLOBs.

Errors set

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

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

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

Example

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

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

GetDatetime method

    Gets the column value as a Date.

Syntax

    GetDatetime(*index* As Integer **)** As Date
    Member of **UltraLiteAFLib.ULResultSet**

Parameters

> **index**   The 1-based ordinal in the result set to get.

Returns

> The value as a Date.

Remarks

> This method throws an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method.

GetDouble method

> Gets the column value as a Double.

Syntax

> **GetDouble(** *index* As Integer **)** As Double
> Member of **UltraLiteAFLib.ULResultSet**

Parameters

> **index**   The 1-based ordinal in the result set to get.

Returns

> The value as a Double.

Remarks

> This method throws an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method.

GetInteger method

> Gets the column value as an Integer.

Syntax

> **GetInteger(** *index* As Integer **)** As Integer
> Member of **UltraLiteAFLib.ULResultSet**

Parameters

> **index**   The 1-based ordinal in the result set to get.

Returns

> The value as an Integer.

Remarks

> This method throws an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method.

GetLong method

Gets the column value as a Long.

Syntax

GetLong(*index* As Integer **)** As Long
Member of **UltraLiteAFLib.ULResultSet**

Parameters

index   The 1-based ordinal in the result set to get.

Returns

The value as a Long.

Remarks

This method throws an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method.

GetReal method

Gets the column value as a Real.

Syntax

GetReal(*index* As Integer **)** As Single
Member of **UltraLiteAFLib.ULResultSet**

Parameters

index   The 1-based ordinal in the result set to get.

Returns

The value as a Real.

Remarks

This method throws an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method.

GetString method

Gets the column value as a String.

Syntax

GetString(*index* As Integer **)** As String
Member of **UltraLiteAFLib.ULResultSet**

Parameters

index   The 1-based ordinal in the result set to get.

Returns

The value as a String.

Remarks

This method throws an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method.

GetStringChunk method

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

Syntax

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

Parameters

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

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

data   The data string.

string_len   The length of the string you want returned.

filled_len   The length of the string filled.

Returns

Gets BLOB data from a binary or long binary column.

Remarks

This method throws an error if the underlying column is NULL. Applications should first check for a NULL value in the property or method.

Errors set

ulSQLE_CONVERSION_ERROR   This error occurs if the column data type is not CHAR or LONG VARCHAR.

ulSQLE_INVALID_PARAMETER   This error occurs if the column data type is CHAR and the src_offset is greater than 64K. This error also occurs if offset is less than 0 or string length is less than 0.

IsNull method

Indicates whether this column contains a null value.

Syntax

IsNull(*index* As Integer **)** As Boolean
Member of **UltraLiteAFLib.ULResultSet**

Parameters

index   The column index value.

Returns

True if the value is NULL.

MoveAfterLast method

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

Syntax

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

MoveBeforeFirst method

Moves to a position before the first row.

Syntax

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

MoveFirst method

Moves to the first row.

Syntax

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

Returns

True if successful.

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

MoveLast method

Moves to the last row.

Syntax

MoveLast( ) As Boolean
Member of **UltraLiteAFLib.ULResultSet**

Returns

True if successful.

False if unsuccessful. For example, if there are no rows, the method fails.

MoveNext method

Moves to the next row.

Syntax

MoveNext( ) As Boolean
Member of **UltraLiteAFLib.ULResultSet**

Returns

True if successful.

False if unsuccessful. For example, if there are no rows, the method fails.

MovePrevious method

Moves to the previous row.

Syntax

MovePrevious( ) As Boolean
Member of **UltraLiteAFLib.ULResultSet**

Returns

True if successful.

False if unsuccessful. For example, if there are no rows, the method fails.

MoveRelative method

Moves a certain number of rows relative to the current row.

Syntax

MoveRelative(*index* As Long **)** As Boolean
Member of **UltraLiteAFLib.ULResultSet**

Parameters

> **index** The number of rows to move. The value can be positive, negative, or zero. 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.

Returns

> **True** if successful.

> **False** if unsuccessful. For example, if there are no rows, the method fails.

SetBoolean method

Sets the specified column to the Boolean value passed in.

Syntax

> **SetBoolean(**
>   *index* As Integer**, _**
>   *value* As Boolean
> **)**
> Member of **UltraLiteAFLib.ULResultSet**

Parameters

> **index** The 1-based ordinal of the column in the result set to be set.

> **value** The new boolean value.

SetByte method

Sets the specified column to the Byte value passed in.

Syntax

> **SetByte(**
> *index* As Integer **, _**
> *data* As Byte
> **)**
> Member of **UltraLiteAFLib.ULResultSet**

Parameters

> **index** The 1-based ordinal in the result set.

SetByteChunk method

Sets the specified column to the binary value passed in.

Syntax

    SetByteChunk(
      index As Integer**, _**
      data As Long **, _**
      data_len As Long
)
    Member of **UltraLiteAFLib.ULResultSet**

Parameters

    index   The 1-based ordinal of the column in the result set to be set.

    data   A pointer to the buffer containing the new data.

    data_len   The length of the data buffer.

SetDatetime method

Sets the specified column of the current row to the supplied Datetime value.

Syntax

    SetDatetime(
      index As Integer
      value As Date
)
    Member of **UltraLiteAFLib.ULResultSet**

Parameters

    index   The 1-based ordinal of the column in the current row to set.

    value   The value the column is to receive.

SetDouble method

Sets the parameter to the specified Double value.

Syntax

    SetDouble(
      index As Integer
      value As Double
)
    Member of **UltraLiteAFLib.ULResultSet**

Parameters

    index   The 1-based ordinal of the column in the current row to set.

    value   The value the parameter should receive.

SetInteger method

Sets the parameter to the specified Integer value.

Syntax

SetInteger(
  index As Integer
  value As Integer
)
Member of **UltraLiteAFLib.ULResultSet**

Parameters

index   The 1-based ordinal of the column in the current row to set.

value   The value the parameter should receive.

SetLong method

Sets the parameter to the specified Long value.

Syntax

SetLong(
  index As Integer
  value As Long
)
Member of **UltraLiteAFLib.ULResultSet**

Parameters

index   The 1-based ordinal of the column in the current row to set.

value   The value the parameter should receive.

SetNull method

Sets the specified column to NULL.

Syntax

SetNull(*index* As Integer **)**
Member of **UltraLiteAFLib.ULResultSet**

Parameters

index   The 1-based ordinal of the column in the current row to set.

Update method

Updates the current row of the table with the current data.

Syntax

> **Update()**
> Member of **UltraLiteAFLib.ULResultSet**

UpdateBegin method

Prepares a table for modification of the contents of the current row.

Syntax

> **UpdateBegin()**
> Member of **UltraLiteAFLib.ULResultSet**

ULResultSetSchema class

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

Properties

Syntax	Description
ColumnBaseName As String (read-only)	Gets the base column name of a given column in the result set (if available).
ColumnBaseTableName As String (read-only)	Gets the base table name of a named column in the result set (if available).
ColumnCount As Integer (read-only)	Gets the number of columns in the result set.
ColumnID As Integer (read-only)	Gets the column ID of a named column in the result set.
ColumnName As String (read-only)	Gets the name of the column in the result set.
ColumnPrecision As Integer (read-only)	Gets the precision of the datatype for the column if it is numeric.
ColumnScale As Integer (read-only)	Gets the scale of the datatype for the column if it is numeric.
ColumnSize As Integer (read-only)	Gets the size of the datatype for the column.
ColumnSQLType As ULSQLType (read-only)	Gets the ULSQLType of the column.

ULSQLCode enumeration

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

For a detailed description of the error codes, follow the link provided in the description to the *SQL Anywhere 10 - Error Messages* book.

members	Description
SQLE_AGGREGATES_NOT_ALLOWED	See "Invalid use of an aggregate function" [*SQL Anywhere 10 - Error Messages*].
SQLE_ALIAS_NOT_UNIQUE	See "Alias '%1' is not unique" [*SQL Anywhere 10 - Error Messages*].
SQLE_ALIAS_NOT_YET_DEFINED	See "Definition for alias '%1' must appear before its first reference" [*SQL Anywhere 10 - Error Messages*].
SQLE_AMBIGUOUS_INDEX_NAME	See "Index name '%1' is ambiguous" [*SQL Anywhere 10 - Error Messages*].
SQLE_BAD_ENCRYPTION_KEY	See "Incorrect or missing encryption key" [*SQL Anywhere 10 - Error Messages*].
SQLE_BAD_PARAM_INDEX	See "Input parameter index out of range" [*SQL Anywhere 10 - Error Messages*].
SQLE_CANNOT_ACCESS_FILESYSTEM	See "Unable to access the filesystem on the device" [*SQL Anywhere 10 - Error Messages*].
SQLE_CANNOT_CHANGE_USER_NAME	See "Cannot change synchronization user_name when status of the last upload is unknown" [*SQL Anywhere 10 - Error Messages*].
SQLE_CANNOT_CONVERT	See "Invalid data conversion" [*SQL Anywhere 10 - Error Messages*].
SQLE_CANNOT_EXECUTE_STMT	See "Statement cannot be executed" [*SQL Anywhere 10 - Error Messages*].
SQLE_CANNOT_MODIFY	See "Cannot modify column '%1' in table '%2'" [*SQL Anywhere 10 - Error Messages*].
SQLE_CLIENT_OUT_OF_MEMORY	See "Client out of memory" [*SQL Anywhere 10 - Error Messages*].
SQLE_COLUMN_AMBIGUOUS	See "Column '%1' found in more than one table -- need a correlation name" [*SQL Anywhere 10 - Error Messages*].
SQLE_COLUMN_CANNOT_BE_NULL	See "Column '%1' in table '%2' cannot be NULL" [*SQL Anywhere 10 - Error Messages*].
SQLE_COLUMN_IN_INDEX	See "Cannot alter a column in an index" [*SQL Anywhere 10 - Error Messages*].

members	Description
SQLE_COLUMN_NOT_FOUND	See "Column '%1' not found" [*SQL Anywhere 10 - Error Messages*].
SQLE_COLUMN_NOT_IN-DEXED	See "Column '%1' not part of any indexes in its containing ta-ble" [*SQL Anywhere 10 - Error Messages*].
SQLE_COLUMN_NOT_STREA MABLE	See "The operation failed because column '%1''s type does not sup-port streaming" [*SQL Anywhere 10 - Error Messages*].
SQLE_COMMUNICATIONS_E RROR	See "Communication error" [*SQL Anywhere 10 - Error Messages*].
SQLE_CONNECTION_ALREA DY_EXISTS	See "This connection already exists" [*SQL Anywhere 10 - Error Messages*].
SQLE_CONNECTION_NOT_F OUND	See "Connection not found" [*SQL Anywhere 10 - Error Messages*].
SQLE_CONNECTION_RE-STORED	See "UltraLite connection was restored" [*SQL Anywhere 10 - Error Messages*].
SQLE_CONSTRAINT_NOT_FO UND	See "Constraint '%1' not found" [*SQL Anywhere 10 - Error Messages*].
SQLE_CONVERSION_ERROR	See "Cannot convert %1 to a %2" [*SQL Anywhere 10 - Error Messages*].
SQLE_COULD_NOT_FIND_FU NCTION	See "Could not find '%1' in dynamic library '%2'" [*SQL Anywhere 10 - Error Messages*].
SQLE_COULD_NOT_LOAD_LI BRARY	See "Could not load dynamic library '%1'" [*SQL Anywhere 10 - Error Messages*].
SQLE_CURSOR_ALREADY_O PEN	See "Cursor already open" [*SQL Anywhere 10 - Error Messages*].
SQLE_CURSOR_NOT_DE-CLARED	See "Cursor has not been declared" [*SQL Anywhere 10 - Error Messages*].
SQLE_CURSOR_NOT_OPEN	See "Cursor not open" [*SQL Anywhere 10 - Error Messages*].
SQLE_CURSOR_RESTORED	See "UltraLite cursor (or result set or table) was restored" [*SQL Anywhere 10 - Error Messages*].
SQLE_CURSOROP_NOT_AL-LOWED	See "Illegal cursor operation attempt" [*SQL Anywhere 10 - Error Messages*].
SQLE_DATABASE_ERROR	See "Internal database error %1 -- transaction rolled back" [*SQL Anywhere 10 - Error Messages*].
SQLE_DATABASE_NAME_RE QUIRED	See "Database name required to start server" [*SQL Anywhere 10 - Error Messages*].

members	Description
SQLE_DATABASE_NOT_CRE-ATED	See "Database creation failed: %1" [*SQL Anywhere 10 - Error Messages*].
SQLE_DATATYPE_NOT_AL-LOWED	See "Expression has unsupported data type" [*SQL Anywhere 10 - Error Messages*].
SQLE_DB_INIT_NOT_CALLED	See "db_init has not been called or the call to db_init failed" [*SQL Anywhere 10 - Error Messages*]
SQLE_DBLIB_ENGINE_MIS-MATCH	See "Client/database server version mismatch" [*SQL Anywhere 10 - Error Messages*]
SQLE_DBSPACE_FULL	See "A dbspace has reached its maximum file size" [*SQL Anywhere 10 - Error Messages*].
SQLE_DESCRIBE_NONSE-LECT	See "Can only describe a SELECT statement" [*SQL Anywhere 10 - Error Messages*].
SQLE_DEVICE_IO_FAILED	See "File I/O failed for '%1'" [*SQL Anywhere 10 - Error Messages*].
SQLE_DIV_ZERO_ERROR	See "Division by zero" [*SQL Anywhere 10 - Error Messages*].
SQLE_DOUBLE_REQUEST	See "Attempted two active database requests" [*SQL Anywhere 10 - Error Messages*]
SQLE_DOWNLOAD_CON-FLICT	See "Download failed because of conflicts with existing rows" [*SQL Anywhere 10 - Error Messages*].
SQLE_DOWNLOAD_RESTART_FAILED	See "Unable to retry download because upload is not finished" [*SQL Anywhere 10 - Error Messages*].
SQLE_DROP_DATABASE_FAILED	See "An attempt to delete database '%1' failed" [*SQL Anywhere 10 - Error Messages*].
SQLE_DUPLICATE_CURSOR_NAME	See "The cursor name '%1' already exists" [*SQL Anywhere 10 - Error Messages*].
SQLE_DUPLICATE_FOREIGN_KEY	See "Foreign key '%1' for table '%2' duplicates an existing foreign key" [*SQL Anywhere 10 - Error Messages*].
SQLE_DUPLICATE_OPTION	See "Option '%1' specified more than once" [*SQL Anywhere 10 - Error Messages*].
SQLE_DYNAMIC_MEMORY_EXHAUSTED	See "Dynamic memory exhausted" [*SQL Anywhere 10 - Error Messages*].
SQLE_ENCRYPTION_INITIALIZATION_FAILED	See "Could not initialize the encryption DLL: '%1'" [*SQL Anywhere 10 - Error Messages*].
SQLE_ENGINE_ALREADY_RUNNING	See "Database server already running" [*SQL Anywhere 10 - Error Messages*].

members	Description
SQLE_ENGINE_NOT_RUN-NING	See "Database server not found" [*SQL Anywhere 10 - Error Messages*]
SQLE_ERROR	See "Run time SQL error -- %1" [*SQL Anywhere 10 - Error Messages*].
SQLE_ERROR_CALLING_FUNCTION	See "Could not allocate resources to call external function" [*SQL Anywhere 10 - Error Messages*].
SQLE_ERROR_IN_ASSIGN-MENT	See "Error in assignment" [*SQL Anywhere 10 - Error Messages*].
SQLE_EXPRESSION_ERROR	See "Invalid expression near '%1'" [*SQL Anywhere 10 - Error Messages*].
SQLE_FEATURE_NOT_EN-ABLED	See "The method you attempted to invoke was not enabled for your application" [*SQL Anywhere 10 - Error Messages*].
SQLE_FILE_BAD_DB	See "Unable to start specified database: '%1' is not a valid database file" [*SQL Anywhere 10 - Error Messages*].
SQLE_FILE_IN_USE	See "Specified database file already in use" [*SQL Anywhere 10 - Error Messages*].
SQLE_FILE_NOT_DB	See "Unable to start specified database: '%1' is not a database" [*SQL Anywhere 10 - Error Messages*].
SQLE_FILE_VOLUME_NOT_FOUND	See "Specified file system volume not found for database '%1'" [*SQL Anywhere 10 - Error Messages*].
SQLE_FILE_WRONG_VER-SION	See "Unable to start specified database: '%1' was created by a different version of the software" [*SQL Anywhere 10 - Error Messages*].
SQLE_FOREIGN_KEY_NAME_NOT_FOUND	See "Foreign key name '%1' not found" [*SQL Anywhere 10 - Error Messages*].
SQLE_IDENTIFIER_TOO_LONG	See "Identifier '%1' too long" [*SQL Anywhere 10 - Error Messages*].
SQLE_INCORRECT_VOLUME_ID	See "Incorrect volume ID for '%1'" [*SQL Anywhere 10 - Error Messages*].
SQLE_INDEX_NAME_NOT_UNIQUE	See "Index name '%1' not unique" [*SQL Anywhere 10 - Error Messages*].
SQLE_INDEX_NOT_FOUND	See "Cannot find index named '%1'" [*SQL Anywhere 10 - Error Messages*].
SQLE_INDEX_NOT_UNIQUE	See "Index '%1' for table '%2' would not be unique" [*SQL Anywhere 10 - Error Messages*].

members	Description
SQLE_INTERRUPTED	See "Statement interrupted by user" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_CONSTRAINT_REF	See "Invalid reference to or operation on constraint '%1'" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_DESCRIPTOR_INDEX	See "Invalid descriptor index" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_DESCRIPTOR_NAME	See "Invalid SQL descriptor name" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_DISTINCT_AGGREGATE	See "Grouped query contains more than one distinct aggregate function" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_FOREIGN_KEY	See "No primary key value for foreign key '%1' in table '%2'" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_FOREIGN_KEY_DEF	See "Column '%1' in foreign key has a different definition than primary key" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_GROUP_SELECT	See "Function or column reference to '%1' must also appear in a GROUP BY" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_INDEX_TYPE	See "Index type specification of '%1' is invalid" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_LOGON	See "Invalid user ID or password" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_OPTION_SETTING	See "Invalid setting for option '%1'" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_OPTION_VALUE	See "'%1' is an invalid value for '%2'" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_ORDER	See "Invalid ORDER BY specification" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_PARAMETER	See "Invalid parameter" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_PARSE_PARAMETER	See "Parse error: %1" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_PUBLICATION_MASK	See "The specified publication mask is invalid" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_SQL_IDENTIFIER	See "Invalid SQL identifier" [*SQL Anywhere 10 - Error Messages*].
SQLE_INVALID_STATEMENT	See "Invalid statement" [*SQL Anywhere 10 - Error Messages*].

members	Description
SQLE_INVALID_UNION	See "Select lists in UNION, INTERSECT, or EXCEPT do not match in length" [*SQL Anywhere 10 - Error Messages*].
SQLE_KEYLESS_ENCRYP-TION	See "Unable to perform requested operation since this database uses keyless encryption" [*SQL Anywhere 10 - Error Messages*].
SQLE_LOCKED	See "User '%1' has the row in '%2' locked" [*SQL Anywhere 10 - Error Messages*].
SQLE_MEMORY_ERROR	See "Memory error -- transaction rolled back" [*SQL Anywhere 10 - Error Messages*].
SQLE_METHOD_CANNOT_BE_CALLED	See "Method '%1' cannot be called at this time" [*SQL Anywhere 10 - Error Messages*].
SQLE_NAME_NOT_UNIQUE	See "Item '%1' already exists" [*SQL Anywhere 10 - Error Messages*].
SQLE_NO_COLUMN_NAME	See "Derived table '%1' has no name for column %2" [*SQL Anywhere 10 - Error Messages*].
SQLE_NO_CURRENT_ROW	See "No current row of cursor" [*SQL Anywhere 10 - Error Messages*].
SQLE_NO_INDICATOR	See "No indicator variable provided for NULL result" [*SQL Anywhere 10 - Error Messages*].
SQLE_NO_MATCHING_SELECT_ITEM	See "The select list for the derived table '%1' has no expression to match '%2'" [*SQL Anywhere 10 - Error Messages*].
SQLE_NO_PRIMARY_KEY	See "Table '%1' has no primary key" [*SQL Anywhere 10 - Error Messages*].
SQLE_NOERROR	SQLE_NOERROR(0) - This code indicates that there was no error or warning.
SQLE_NON_UPDATEABLE_COLUMN	See "Cannot update an expression" [*SQL Anywhere 10 - Error Messages*].
SQLE_NON_UPDATEABLE_CURSOR	See "FOR UPDATE has been incorrectly specified for a READ ONLY cursor" [*SQL Anywhere 10 - Error Messages*].
SQLE_NOT_IMPLEMENTED	See "Feature '%1' not implemented" [*SQL Anywhere 10 - Error Messages*].
SQLE_NOT_SUPPORTED_IN_ULTRALITE	See "Feature not available with UltraLite" [*SQL Anywhere 10 - Error Messages*].
SQLE_NOTFOUND	See "Row not found" [*SQL Anywhere 10 - Error Messages*].
SQLE_ONLY_ONE_TABLE	See "INSERT/DELETE on cursor can modify only one table" [*SQL Anywhere 10 - Error Messages*].

members	Description
SQLE_OVERFLOW_ERROR	See "Value %1 out of range for destination" [*SQL Anywhere 10 - Error Messages*].
SQLE_PAGE_SIZE_INVALID	See "Invalid database page size" [*SQL Anywhere 10 - Error Messages*].
SQLE_PARTIAL_DOWNLOAD _NOT_FOUND	See "No partial download was found" [*SQL Anywhere 10 - Error Messages*].
SQLE_PERMISSION_DENIED	See "Permission denied: %1" [*SQL Anywhere 10 - Error Messages*].
SQLE_PRIMARY_KEY_NOT_ UNIQUE	See "Primary key for table '%1' is not unique" [*SQL Anywhere 10 - Error Messages*].
SQLE_PRIMARY_KEY_TWIC E	See "Table cannot have two primary keys" [*SQL Anywhere 10 - Error Messages*].
SQLE_PRIMARY_KEY_VALU E_REF	See "Primary key for row in table '%1' is referenced by foreign key '%2' in table '%3'" [*SQL Anywhere 10 - Error Messages*].
SQLE_PUBLICATION_NOT_F OUND	See "Publication '%1' not found" [*SQL Anywhere 10 - Error Messages*].
SQLE_PUBLICATION_PREDIC ATE_IGNORED	See "Publication predicates were not evaluated" [*SQL Anywhere 10 - Error Messages*].
SQLE_RESOURCE_GOVERNO R_EXCEEDED	See "Resource governor for '%1' exceeded" [*SQL Anywhere 10 - Error Messages*].
SQLE_ROW_DELETED_TO_M AINTAIN_REFERENTIAL_IN- TEGRITY	See "Row was dropped from table %1 to maintain referential integrity" [*SQL Anywhere 10 - Error Messages*].
SQLE_SERVER_SYNCHRONI ZATION_ERROR	See "Synchronization failed due to an error on the server: %1" [*SQL Anywhere 10 - Error Messages*].
SQLE_START_STOP_DATAB ASE_DENIED	See "Request to start/stop database denied" [*SQL Anywhere 10 - Error Messages*].
SQLE_STATEMENT_ERROR	See "SQL statement error" [*SQL Anywhere 10 - Error Messages*].
SQLE_STRING_RIGHT_TRUN- CATION	See "Right truncation of string data" [*SQL Anywhere 10 - Error Messages*].
SQLE_SUBQUERY_SELECT_L IST	See "Subquery allowed only one select list item" [*SQL Anywhere 10 - Error Messages*].
SQLE_SYNC_INFO_INVALID	See "Information for synchronization is incomplete or invalid, check '%1'" [*SQL Anywhere 10 - Error Messages*].
SQLE_SYNC_INFO_RE- QUIRED	See "Information for synchronization was not provided" [*SQL Anywhere 10 - Error Messages*].

members	Description
SQLE_SYNC_NOT_REEN-TRANT	See "Synchronization process was unable to re-enter synchroniza-tion" [*SQL Anywhere 10 - Error Messages*].
SQLE_SYNC_STATUS_UN-KNOWN	See "The status of the last synchronization upload is un-known" [*SQL Anywhere 10 - Error Messages*].
SQLE_SYNTAX_ERROR	See "Syntax error near '%1' %2" [*SQL Anywhere 10 - Error Messages*].
SQLE_TABLE_ALREADY_IN-CLUDED	See "Table '%1' is already included" [*SQL Anywhere 10 - Error Messages*].
SQLE_TABLE_IN_USE	See "Table in use" [*SQL Anywhere 10 - Error Messages*].
SQLE_TABLE_NOT_FOUND	See "Table '%1' not found" [*SQL Anywhere 10 - Error Messages*].
SQLE_TOO_MANY_BLOB_REFS	See "Too many references to a BLOB" [*SQL Anywhere 10 - Error Messages*].
SQLE_TOO_MANY_CONNEC-TIONS	See "Database server connection limit exceeded" [*SQL Anywhere 10 - Error Messages*].
SQLE_TOO_MANY_PUBLICA-TIONS	See "Too many publications specified in publication mask" [*SQL Anywhere 10 - Error Messages*].
SQLE_TOO_MANY_TEMP_TABLES	See "Too many temporary tables in connection" [*SQL Anywhere 10 - Error Messages*].
SQLE_TOO_MANY_USERS	See "Too many users in database" [*SQL Anywhere 10 - Error Messages*].
SQLE_ULTRALITE_DATABASE_NOT_FOUND	See "The database '%1' was not found" [*SQL Anywhere 10 - Error Messages*].
SQLE_ULTRALITE_OBJ_CLOSED	See "Invalid operation on a closed object" [*SQL Anywhere 10 - Error Messages*].
SQLE_ULTRALITE_WRITE_ACCESS_DENIED	See "Write access was denied" [*SQL Anywhere 10 - Error Messages*].
SQLE_UNABLE_TO_CON-NECT	See "Database cannot be started -- %1" [*SQL Anywhere 10 - Error Messages*].
SQLE_UNABLE_TO_CONNECT_OR_START	See "Server not found and unable to autostart" [*SQL Anywhere 10 - Error Messages*]
SQLE_UNABLE_TO_START_DATABASE	See "Unable to start specified database: %1" [*SQL Anywhere 10 - Error Messages*].
SQLE_UNABLE_TO_START_DATABASE_VER_NEWER	See "Unable to start specified database: Server must be upgraded to start database %1" [*SQL Anywhere 10 - Error Messages*].

members	Description
SQLE_UNABLE_TO_START_ENGINE	See "Unable to start database server" [*SQL Anywhere 10 - Error Messages*]
SQLE_UNCOMMITTED_TRANSACTIONS	See "You cannot synchronize or upgrade with uncommitted transactions" [*SQL Anywhere 10 - Error Messages*].
SQLE_UNKNOWN_FUNC	See "Unknown function '%1'" [*SQL Anywhere 10 - Error Messages*].
SQLE_UNKNOWN_OPTION	See "'%1' is an unknown option" [*SQL Anywhere 10 - Error Messages*].
SQLE_UNKNOWN_USERID	See "User ID '%1' does not exist" [*SQL Anywhere 10 - Error Messages*].
SQLE_UNRECOGNIZED_OPTION	See "The option '%1' is not recognized" [*SQL Anywhere 10 - Error Messages*].
SQLE_UPLOAD_FAILED_AT_SERVER	See "Synchronization server failed to commit the upload" [*SQL Anywhere 10 - Error Messages*].
SQLE_VALUE_IS_NULL	See "Cannot return NULL result as requested data type" [*SQL Anywhere 10 - Error Messages*].
SQLE_VARIABLE_INVALID	See "Invalid host variable" [*SQL Anywhere 10 - Error Messages*].
SQLE_WRONG_NUM_OF_INSERT_COLS	See "Wrong number of values for INSERT" [*SQL Anywhere 10 - Error Messages*].
SQLE_WRONG_PARAMETER_COUNT	See "Wrong number of parameters to function '%1'" [*SQL Anywhere 10 - Error Messages*].

ULSQLType enumeration

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

Constant	UltraLite Database Type	Value
ulTypeLong	Integer	0
ulTypeShort	UnsignedInteger	1
ulTypeUnsignedLong	SmallInt	2
ulTypeUnsignedShort	UnsignedSmallInt	3
ulTypeBig	Big	4
ulTypeUnsignedBig	UnsignedBig	5
ulTypeByte	Byte	6
ulTypeBit	Bit	7
ulTypeDateTime	Time	8
ulTypeDate	Date	9
ulTypeTime	Timestamp	10
ulTypeDouble	Double	11
ulTypeReal	Real	12
ulTypeBinary	LongBinary	13
ulTypeLongBinary	Numeric	14
ulTypeString	(Var)Char	15
ulTypeLongString	LongVarchar	16
ulTypeNumeric	(Var)Binary	17
ulTypeUUID	UniqueIdentifier	18

ULStreamErrorCode enumeration

The ULStreamErrorCode constants identify communications errors during synchronization.

For more information about these errors, see "MobiLink Communication Error Messages" [*SQL Anywhere 10 - Error Messages*].

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

Constant	Value
ulStreamErrorCodeSecureCertificateExpired	50
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
ulStreamErrorIDPalmConduit	3
ulStreamErrorIDPalmSs	4
ulStreamErrorIDNettech	5
ulStreamErrorIDRimbb	6
ulStreamErrorIDHttp	7
ulStreamErrorIDHttps	8
ulStreamErrorIDDhCast	9
ulStreamErrorIDSecure	10
ulStreamErrorIDCerticom	11
ulStreamErrorIDJavaCerticom	12
ulStreamErrorIDCerticomSsl	13
ulStreamErrorIDCerticomTls	14
ulStreamErrorIDWirestrm	15
ulStreamErrorIDWireless	16
ulStreamErrorIDReplay	17
ulStreamErrorIDStrm	18
ulStreamErrorIDUdp	19
ulStreamErrorIDEmail	20
ulStreamErrorIDFile	21
ulStreamErrorIDActivesync	22
ulStreamErrorIDRsaTls	23

Constant	Value
ulStreamErrorIDJavaRsa	24
ulStreamErrorIDOpenSslRsa	25
ulStreamErrorIDPalmSsl	26

ULStreamType enumeration

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

Constant	Value	Description
ulUnknown	0	No stream type has been set. You must set a stream type before synchronization.
ulTCPIP	1	TCP/IP stream
ulHTTP	2	HTTP stream
ulHTTPS	3	HTTPS synchronization
ulPalmConduit	4	For HotSync synchronization
ulTLS	5	TLS (Transport Layer Security)

ULSyncEvent class

OnReceive event

Reports download information to the application from the consolidated database via MobiLink.

Syntax

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

Parameters

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

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

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

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

Remarks

This event can be called multiple times.

Example

For an example of this method, see the CustDB application. For more information about finding CustDB for AppForge, see "Finding CustDB sample files" [*UltraLite - Database Management and Reference*].

OnSend event

Reports upload information from the remote database via MobiLink to the consolidated database.

Syntax

OnSend(
  nBytes As Long**, _**
  nInserts As Long**, _**
  nUpdates As Long**, _**
  nDeletes As Long **_**
)
Member of **UltraLiteAFLib.ULSyncEvent**

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.

Remarks

This event can be called multiple times.

Example

For an example of this method, see the CustDB application. For information about finding CustDB for AppForge, see "Finding CustDB sample files" [*UltraLite - Database Management and Reference*].

OnStateChange event

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

Syntax

OnStateChange(
  newState As ULSyncState**, _**
  oldState As ULSyncState **_**
)
Member of **UltraLiteAFLib.ULSyncEvent**

Parameters

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

oldState   The state that the synchronization process just completed.

See also

♦   "ULSyncState enumeration" on page 148

Example

For an example of this method, see the CustDB application. For information about finding CustDB for AppForge, see "Finding CustDB sample files" [*UltraLite - Database Management and Reference*].

OnTableChange event

This event is called whenever the synchronization process begins synchronizing another table.

Syntax

OnTableChange(
  newTableIndex As Long**, _**
  numTables As Long **_**

)
Member of **UltraLiteAFLib.ULSyncEvent**

Parameters

newTableIndex   The index number of the table currently being synchronized. This number is not the same as the table ID, therefore, it cannot be used with the ULDatabaseSchema.GetTableName method.

numTables   The number of tables eligible to be synchronized.

Example

For an example of this method, see the CustDB application. For information about finding CustDB for AppForge, see "Finding CustDB sample files" [*UltraLite - Database Management and Reference*].

OnWaiting event

This event is called whenever synchronization is waiting for a MobiLink response.

Syntax

OnWaiting() Member of **UltraLiteAFLib.ULSyncEvent**

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:

Syntax	Description
CheckpointStore As Boolean	If true, adds checkpoints of the database during synchronization to limit database growth during the synchronization process. This is most useful for large downloads with many updates. See "Checkpoint Store synchronization parameter" [*MobiLink - Client Administration*].
DownloadOnly As Boolean	Indicates if a synchronization only downloads data. See "Download Only synchronization parameter" [*MobiLink - Client Administration*].
KeepPartialDownload As Boolean	If the synchronization fails during download because of a communications error, apply those changes that were successfully downloaded, rather than rolling back all the changes. See "Keep Partial Download synchronization parameter" [*MobiLink - Client Administration*].
NewPassword As String	Change a user password to this new password string on the next synchronization. See "New Password synchronization parameter" [*MobiLink - Client Administration*].
Password As String	The password corresponding to a given user name. See "Password synchronization parameter" [*MobiLink - Client Administration*].
PingOnly As Boolean	If true, check the server for liveness, but do not synchronize data. See "Ping synchronization parameter" [*MobiLink - Client Administration*].
PublicationMask As Long	Specify the publications to synchronize. The default is to synchronize all data. See "Publication synchronization parameter" [*MobiLink - Client Administration*].

Syntax	Description
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.
	See "Resume Partial Download synchronization parameter" [*MobiLink - Client Administration*].
SendColumnNames As Boolean	If SendColumnNames is true, column names are sent to the MobiLink server.
	See "Send Column Names synchronization parameter" [*MobiLink - Client Administration*].
SendDownloadAck As Boolean	If SendDownloadAck is true, a download acknowledgement is sent during synchronization.
	See "Send Download Acknowledgment synchronization parameter" [*MobiLink - Client Administration*].
Stream As ULStreamType constants	Set the type of stream to use during synchronization.
	See "Stream Type synchronization parameter" [*MobiLink - Client Administration*].
StreamParms As String	Set network protocol options for the given stream type.
	See "Stream Parameters synchronization parameter" [*MobiLink - Client Administration*] and "Network protocol options for UltraLite synchronization streams" [*MobiLink - Client Administration*].
TableOrder As String	Specify Table synchronization order.
	See "Table Order synchronization parameter" [*MobiLink - Client Administration*]
UploadOnly As Boolean	Indicates whether a synchronization only uploads data.
	See "Upload Only synchronization parameter" [*MobiLink - Client Administration*].
UserName As String	The MobiLink user name for synchronization.
	See "User Name synchronization parameter" [*MobiLink - Client Administration*].
Version As String	The synchronization script version to run.
	See "Version synchronization parameter" [*MobiLink - Client Administration*].

Example

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

```
With Connection.SyncParms
    .UserName = "afsample"
    .Stream = ULStreamType.ulTCPIP
    .Version = "ul_default"
End With
Connection.Synchronize
```

AddAuthenticationParm method

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

Syntax

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

Parameters

parm   The parameter being added.

See also

♦ "Authentication Parameters synchronization parameter" [*MobiLink - Client Administration*]
♦ "authenticate_parameters connection event" [*MobiLink - Server Administration*]

ClearAuthenticationParms method

Clears all parameters that were to be passed to the authenticate_parms MobiLink synchronization script.

Syntax

ClearAuthenticationParms( )
Member of **UltraLiteAFLib.ULSyncParms**

See also

♦ "Authentication Parameters synchronization parameter" [*MobiLink - Client Administration*]
♦ "authenticate_parameters connection event" [*MobiLink - Server Administration*]

ULSyncResult class

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

Properties

The following are properties of ULSyncResult:

Syntax	Description
AuthStatus As ULAuthStatusCode (read-only)	Gets the authorization status code for the last synchronization. See "Authentication Status synchronization parameter" [*MobiLink - Client Administration*].
AuthValue As Long (read-only)	Gets the MobiLink authentication value. See "Authentication Value synchronization parameter" [*MobiLink - Client Administration*].
PartialDownloadRetained As Boolean (read-only)	Indicates that the synchronization failed during download, and that a partial download was kept. See "Partial Download Retained synchronization parameter" [*MobiLink - Client Administration*].
IgnoredRows As Boolean (read-only)	Indicates whether rows were ignored during the last synchronization. See "Ignored Rows synchronization parameter" [*MobiLink - Client Administration*].
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.
Timestamp as Date (read-only)	Gets the timestamp of the last synchronization.
UploadOK As Boolean (read-only)	Indicates whether data was uploaded successfully in the last synchronization. See "Version synchronization parameter" [*MobiLink - Client Administration*].

ULSyncState enumeration

Constant	Value	Description
ulSyncStateStarting	0	No synchronization actions have been taken yet.
ulSyncStateConnecting	1	The synchronization stream has been built, but not yet opened.
ulSyncStateSendingHeader	2	The synchronization stream has been opened and the header is about to be sent.
ulSyncStateSendingTable	3	A table is being sent.
ulSyncStateSendingData	4	Data for the current table is being sent.
ulSyncStateFinishingUpload	5	The upload is completing. The final count of rows sent is included with this event.
ulSyncStateReceivingUploadAck	6	An acknowledgement that the upload is complete is being received.
ulSyncStateReceivingTable	7	A table is being received.
ulSyncStateReceivingData	8	Data for the current table is being received.
ulSyncStateCommittingDown-load	9	The download is being committed. The final count of rows received is included with this event.
ulSyncStateSendingDownload-Ack	10	An acknowledgement that the download is complete is being sent.
ulSyncStateDisconnecting	11	The synchronization stream is about to be closed.
ulSyncStateDone	12	Synchronization has successfully completed. The SyncResult object has been updated.
ulSyncStateError	13	Synchronization has completed but an error occurred. Check SyncResult and SQLCode for details.
ulSyncStateRollingBackDown-load	14	Synchronization is rolling back the download because an error was encountered during the download. The error is reported with a subsequent ulSyncStateError progress report.
ulSyncStateCancelled	99	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 the following table modes for table operations.

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

Properties

Syntax	Description
BOF As Boolean (read-only)	Indicates whether the current row position is before the first row. Returns True if the current row position is before the first row, otherwise false.
EOF As Boolean (read-only)	Indicates whether the current row position is after the last row. Returns True if the current row position is before the first row, otherwise false.
IsOpen As Boolean (read-only)	Indicates whether 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

Frees resources associated with the table.

Syntax

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

Remarks

This method should be called after all processing involving the table is complete.

For the Palm OS, if a table is not closed it can be reopened to its current position.

Column method

Returns the object for the specified column name.

Syntax

Column(*name* As String **)** As ULColumn
Member of **UltraLiteAFLib.ULTable**

Parameters

name   The name of the column to return.

Returns

Returns a Column's object.

See also

♦ "ULColumn class" on page 73

Delete method

Deletes the current row from the table.

Syntax

Delete( )
Member of **UltraLiteAFLib.ULTable**

DeleteAllRows method

Deletes all rows in the table.

Syntax

DeleteAllRows( )
Member of **UltraLiteAFLib.ULTable**

Remarks

In some applications, it can be useful to delete all rows from tables before downloading a new set of data into the table.

You can delete rows from the UltraLite database without having them deleted from the consolidated database as well:

♦ Use **ULConnection.StopSynchronizationDelete** .

♦ Call **Truncate** instead of this method.

See also

♦ "StopSynchronizationDelete method" on page 85
♦ "Truncate method" on page 158

FindBegin method

Prepares a table for a find.

Syntax

FindBegin( )
Member of **UltraLiteAFLib.ULTable**

FindFirst method

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

Syntax

FindFirst([*num_columns* As Long = 32767] **)** As Boolean
Member of **UltraLiteAFLib.ULTable**

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.

Remarks

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 for which to search, 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**
> This method requires that you first call FindBegin.

See also

♦   "FindBegin method" on page 151

FindLast method

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

Syntax

> **FindLast(** [*num_columns* As Long = 32767] **)** As Boolean
> Member of **UltraLiteAFLib.ULTable**

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.

Remarks

> 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.

> To specify the value for which to search, 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**
> This method requires that you first call FindBegin.
> ---

See also

> ♦ "Open method" on page 158
> ♦ "FindBegin method" on page 151

FindNext method

> Moves forward 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.

Syntax

> **FindNext(** [*num_columns* As Long = 32767] **)** As Boolean
> Member of **UltraLiteAFLib.ULTable**

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).

Remarks

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.

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**
> This method requires that you first call FindFirst or FindLast.

See also

FindPrevious method

Moves backward 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.

Syntax

FindPrevious([*num_columns* As Long = 32767] **)** As Boolean
Member of **UltraLiteAFLib.ULTable**

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).

Remarks

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.

On failure it is positioned before the first row (BOF).

See also

Insert method

Inserts a row in the table with values specified in previous **Set** methods.

Syntax

Insert( ) As Boolean
Member of **UltraLiteAFLib.ULTable**

Remarks

This method requires that you first call InsertBegin.

Returns

True if successful.

False if unsuccessful (BOF).

See also

♦ "InsertBegin method" on page 154

InsertBegin method

Prepares a table for inserting a new row, setting column values to their defaults.

Syntax

InsertBegin( )
Member of **UltraLiteAFLib.ULTable**

Examples

In this example, InsertBegin sets insert mode to allow you to begin assigning data values to CustomerTable columns.

```
CustomerTable.InsertBegin
CustomerTable.Column("Fname").StringValue = fname
CustomerTable.Column("Lname").StringValue = lname
CustomerTable.Insert
```

See also

♦ "UpdateBegin method" on page 159

LookupBackward method

Moves backward 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.

Syntax

LookupBackward([*num_columns* As Long = 32767] **)** As Boolean
Member of **UltraLiteAFLib.ULTable**

Parameters

>**num_columns** For composite indexes, the number of columns to use in the lookup.

Returns

>**True** if successful.

>**False** if unsuccessful.

Remarks

>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.

>To specify the value for which to search, 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 for which it is searching), the cursor position is before the first row (BOF).

See also

>♦ "Open method" on page 158

LookupBegin method

>Prepares a table for a lookup.

Syntax

>**LookupBegin( )**
>Member of **UltraLiteAFLib.ULTable**

LookupForward method

>Moves 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.

Syntax

>**LookupForward(** [*num_columns* As Long = 32767] **)** As Boolean
>Member of **UltraLiteAFLib.ULTable**

Remarks

>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.

>To specify the search value, 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 for which it is searching), 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.

See also

> ♦ "Open method" on page 158

MoveAfterLast method

> Moves to a position after the last row.

Syntax

> **MoveAfterLast( )** As Boolean
> Member of   **UltraLiteAFLib.ULTable**

Returns

> **True** if successful.

> **False** if the operation fails.

MoveBeforeFirst method

> Moves to a position before the first row.

Syntax

> **MoveBeforeFirst( )** As Boolean
> Member of **UltraLiteAFLib.ULTable**

Returns

> **True** if successful.

> **False** if the operation fails.

MoveFirst method

> Moves to the first row.

Syntax

> **MoveFirst( )** As Boolean
> Member of **UltraLiteAFLib.ULTable**

Returns

> **True** if successful.

> **False** if there is no data in the table.

MoveLast method

Moves to the last row.

Syntax

MoveLast( ) As Boolean
Member of **UltraLiteAFLib.ULTable**

Returns

True if successful.

False if there is no data in the table.

MoveNext method

Moves to the next row.

Syntax

MoveNext( ) As Boolean
Member of **UltraLiteAFLib.ULTable**

Returns

True if successful.

False if there is no more data in the table. For example, if there are no more rows, MoveNext fails.

MovePrevious method

Moves to the previous row.

Syntax

MovePrevious( ) As Boolean
Member of **UltraLiteAFLib.ULTable**

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

Moves a certain number of rows relative to the current row.

Syntax

MoveRelative(*index* As Long **)** As Boolean
Member of **UltraLiteAFLib.ULTable**

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. For example, if the cursor is positioned beyond the first or last row.

Open method

    Opens the table so it can be read or manipulated.

Syntax

    Open(
    [*index_name* As String]**,** _
    [*persistent_name* As String] _
)
    Member of **UltraLiteAFLib.ULTable**

Parameters

    index_name   An optional parameter referring to the name of the index.

    persistent_name   For Palm Computing Platform applications, an optional parameter referring to the stored name of the table.

Remarks

    By default, the rows are ordered by primary key. By supplying an index name, the rows can be ordered in other ways.

    The cursor is positioned before the first row in the table.

Truncate method

    Removes all data from this table.

Syntax

    Truncate( )
    Member of **UltraLiteAFLib.ULTable**

Remarks

    The changes are not synchronized. On synchronization, truncated data in the remote UltraLite database does get removed in the consolidated database.

See also

♦ "StartSynchronizationDelete method" on page 84

Update method

Updates a row in the table with values specified in **ULColumn** methods.

Syntax

Update( )
Member of **UltraLiteAFLib.ULTable**

Remarks

This method requires that you first call UpdateBegin.

See also

♦ "UpdateBegin method" on page 159

UpdateBegin method

Prepares a table for modifying the contents of the current row.

Syntax

UpdateBegin( )
Member of **UltraLiteAFLib.ULTable**

Example

```
CustomerTable.UpdateBegin
CustomerTable.Column("Fname").StringValue = fname
'...
CustomerTable.Update
```

ULTableSchema class

The ULTableSchema object allows you to obtain the attributes of a table.

Properties

The ULTableSchema represents metadata about the table. The following are properties of the ULTableSchema class:

Syntax	Description
ColumnCount As Integer (read-only)	The number of columns in this table.
IndexCount As Integer (read-only)	The number of indexes on this table.
Name As String (read-only)	This table's name.
NeverSynchronized As Boolean (read-only)	Indicates if the table is always excluded from synchronization.
PrimaryKey As ULIndexSchema (read-only)	The primary key for this table.
UploadUnchangedRows As Boolean (read-only)	Indicates if all rows in the table should be uploaded on synchronization, rather than just the rows changed since the last synchronization.

GetColumnName method

Returns the name of the column that corresponds to the *id* value you supply.

Syntax

GetColumnName(*id* As Integer **)** As String
Member of **UltraLiteAFLib.ULTableSchema**

Parameters

id   The id of the column.

Returns

The name of a column.

Remarks

The ColumnCount property returns the number of columns in the table. Each column has a unique number from 1 to the ColumnCount value, where 1 is the first column in the table, 2 is the second column in the table, and so on.

GetIndex method

Returns the ULIndexSchema object for the specified index.

Syntax

GetIndex(*name* As String **)** As ULIndexSchema
Member of **UltraLiteAFLib.ULTableSchema**

Parameters

name   The name of the index.

Returns

Returns a schema object for a given index on the table.

See also

♦   "ULIndexSchema class" on page 101

GetIndexName method

Returns the name of the index in the table that corresponds to the *id* value you supply.

Syntax

GetIndexName(*id* As Integer **)** As String
Member of **UltraLiteAFLib.ULTableSchema**

Parameters

id   The id of the index.

Returns

Returns the name of the index.

Remarks

The IndexCount property returns the number of indexes in the table. Each index has a unique number from 1 to the IndexCount value, where 1 is the first index in the table, 2 is the second index in the table, and so on.

GetPublicationPredicate method

Get publication predicate (if any) for specified publication name.

Syntax

GetPublicationPredicate(
 pub_name As String
) As String
Member of **UltraLiteAFLib.ULTableSchema**

Parameters

    pub_name   Publication name.

Returns

    Returns the publication predicate for the named publication or an empty string.

InPublication method

    Indicates whether this table is part of the specified publication.

Syntax

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

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

Copyright © 2007, iAnywhere Solutions, Inc.

V

W