SYBASE iAnywhere

UltraLite® M-Business Anywhere Programming (deprecated)

Contents

About this book

This book 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 Mobile, or Windows. UltraLite support for M-Business Anywhere is deprecated in UltraLite 12.

M-Business Anywhere is the iAnywhere platform for developing and deploying mobile web-based applications. The previous name for the product was AvantGo M-Business Server.

About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in four formats:

- **DocCommentXchange** DocCommentXchange is a community for accessing and discussing SQL Anywhere documentation on the web.

  To access the documentation, go to http://dcx.sybase.com.

- **HTML Help** On Windows platforms, the HTML Help contains the complete SQL Anywhere documentation, including the books and the context-sensitive help for SQL Anywhere tools.

  To access the documentation, choose **Start** » **Programs** » **SQL Anywhere 12** » **Documentation** » **HTML Help (English)**.

- **Eclipse** On Unix platforms, the complete Help is provided in Eclipse format. To access the documentation, run *sadoc* from the *bin32* or *bin64* directory of your SQL Anywhere installation.

- **PDF** The complete set of SQL Anywhere books is provided as a set of Portable Document Format (PDF) files. You must have a PDF reader to view information.

  To access the PDF documentation on Windows operating systems, choose **Start** » **Programs** » **SQL Anywhere 12** » **Documentation** » **PDF (English)**.

  To access the PDF documentation on Unix operating systems, use a web browser to open */documentation/en/pdf/index.html* under the SQL Anywhere installation directory.

Documentation conventions

This section lists the conventions used in this documentation.

Operating systems

SQL Anywhere runs on a variety of platforms. Typically, the behavior of the software is the same on all platforms, but there are variations or limitations. These are commonly based on the underlying operating system (Windows, Unix), and seldom on the particular variant (IBM AIX, Windows Mobile) or version.

To simplify references to operating systems, the documentation groups the supported operating systems as follows:

- **Windows** The Microsoft Windows family includes platforms that are used primarily on server, desktop, and laptop computers, as well as platforms used on mobile devices. Unless otherwise specified, when the documentation refers to Windows, it refers to all supported Windows-based platforms, including Windows Mobile.

  Windows Mobile is based on the Windows CE operating system, which is also used to build a variety of platforms other than Windows Mobile. Unless otherwise specified, when the documentation refers to Windows Mobile, it refers to all supported platforms built using Windows CE.

- **Unix** Unless otherwise specified, when the documentation refers to Unix, it refers to all supported Unix-based platforms, including Linux and Mac OS X.

For the complete list of platforms supported by SQL Anywhere, see "Supported platforms" [*SQL Anywhere 12 - Introduction*].

Directory and file names

Usually references to directory and file names are similar on all supported platforms, with simple transformations between the various forms. In these cases, Windows conventions are used. Where the details are more complex, the documentation shows all relevant forms.

These are the conventions used to simplify the documentation of directory and file names:

- **Uppercase and lowercase directory names** On Windows and Unix, directory and file names may contain uppercase and lowercase letters. When directories and files are created, the file system preserves letter case.

  On Windows, references to directories and files are *not* case sensitive. Mixed case directory and file names are common, but it is common to refer to them using all lowercase letters. The SQL Anywhere installation contains directories such as *Bin32* and *Documentation*.

  On Unix, references to directories and files *are* case sensitive. Mixed case directory and file names are not common. Most use all lowercase letters. The SQL Anywhere installation contains directories such as *bin32* and *documentation*.

  The documentation uses the Windows forms of directory names. You can usually convert a mixed case directory name to lowercase for the equivalent directory name on Unix.

- **Slashes separating directory and file names** The documentation uses backslashes as the directory separator. For example, the PDF form of the documentation is found in *install-dir\Documentation\en\PDF* (Windows form).

  On Unix, replace the backslash with the forward slash. The PDF documentation is found in *install-dir/documentation/en/pdf*.

- **Executable files** The documentation shows executable file names using Windows conventions, with a suffix such as *.exe* or *.bat*. On Unix, executable file names have no suffix.

For example, on Windows, the network database server is *dbsrv12.exe*. On Unix, it is *dbsrv12*.

- ***install-dir*** During the installation process, you choose where to install SQL Anywhere. The environment variable SQLANY12 is created and refers to this location. The documentation refers to this location as *install-dir*.

  For example, the documentation may refer to the file *install-dir/readme.txt*. On Windows, this is equivalent to *%SQLANY12%\readme.txt*. On Unix, this is equivalent to *$SQLANY12/readme.txt* or *${SQLANY12}/readme.txt*.

  For more information about the default location of *install-dir*, see "SQLANY12 environment variable" [*SQL Anywhere Server - Database Administration*].

- ***samples-dir*** During the installation process, you choose where to install the samples included with SQL Anywhere. The environment variable SQLANYSAMP12 is created and refers to this location. The documentation refers to this location as *samples-dir*.

  To open a Windows Explorer window in *samples-dir*, choose **Start** » **Programs** » **SQL Anywhere 12** » **Sample Applications And Projects**.

  For more information about the default location of *samples-dir*, see "SQLANYSAMP12 environment variable" [*SQL Anywhere Server - Database Administration*].

Command prompts and command shell syntax

Most operating systems provide one or more methods of entering commands and parameters using a command shell or command prompt. Windows command prompts include Command Prompt (DOS prompt) and 4NT. Unix command shells include Korn shell and bash. Each shell has features that extend its capabilities beyond simple commands. These features are driven by special characters. The special characters and features vary from one shell to another. Incorrect use of these special characters often results in syntax errors or unexpected behavior.

The documentation provides command line examples in a generic form. If these examples contain characters that the shell considers special, the command may require modification for the specific shell. The modifications are beyond the scope of this documentation, but generally, use quotes around the parameters containing those characters or use an escape character before the special characters.

These are some examples of command line syntax that may vary between platforms:

- **Parentheses and curly braces** Some command line options require a parameter that accepts detailed value specifications in a list. The list is usually enclosed with parentheses or curly braces. The documentation uses parentheses. For example:

  ```
  -x tcpip(host=127.0.0.1)
  ```

  Where parentheses cause syntax problems, substitute curly braces:

  ```
  -x tcpip{host=127.0.0.1}
  ```

  If both forms result in syntax problems, the entire parameter should be enclosed in quotes as required by the shell:

```
-x "tcpip(host=127.0.0.1)"
```

- **Semicolons** On Unix, semicolons should be enclosed in quotes.
- **Quotes** If you must specify quotes in a parameter value, the quotes may conflict with the traditional use of quotes to enclose the parameter. For example, to specify an encryption key whose value contains double-quotes, you might have to enclose the key in quotes and then escape the embedded quote:

  ```
  -ek "my \"secret\" key"
  ```

  In many shells, the value of the key would be `my "secret" key`.
- **Environment variables** The documentation refers to setting environment variables. In Windows shells, environment variables are specified using the syntax *%ENVVAR%*. In Unix shells, environment variables are specified using the syntax *$ENVVAR* or *${ENVVAR}*.

Contacting the documentation team

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

You can leave comments directly on help topics using DocCommentXchange. DocCommentXchange (DCX) is a community for accessing and discussing SQL Anywhere documentation. Use DocCommentXchange to:

- View documentation
- Check for clarifications users have made to sections of documentation
- Provide suggestions and corrections to improve documentation for all users in future releases

Go to http://dcx.sybase.com.

Finding out more and requesting technical support

Newsgroups

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 details about your problem, including the build number of your version of SQL Anywhere. You can find this information by running the following command: **dbeng12 -v**.

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

For web development issues, see http://groups.google.com/group/sql-anywhere-web-development.

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, and other staff, assist on the newsgroup service when they have time. They offer their help on a volunteer basis and may not be available regularly to provide solutions and information. Their ability to help is based on their workload.

Developer Centers

The **SQL Anywhere Tech Corner** gives developers easy access to product technical documentation. You can browse technical white papers, FAQs, tech notes, downloads, techcasts and more to find answers to your questions as well as solutions to many common issues. See http://www.sybase.com/developer/library/sql-anywhere-techcorner.

The following table contains a list of the developer centers available for use on the SQL Anywhere Tech Corner:

Name	URL	Description
SQL Anywhere .NET Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/microsoft-net	Get started and get answers to specific questions regarding SQL Anywhere and .NET development.
PHP Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/php	An introduction to using the PHP (PHP Hypertext Preprocessor) scripting language to query your SQL Anywhere database.

Name	URL	Description
SQL Anywhere Windows Mobile Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/windows-mobile	Get started and get answers to specific questions regarding SQL Anywhere and Windows Mobile development.

Introduction to UltraLite for M-Business Anywhere

UltraLite for M-Business Anywhere features

UltraLite for M-Business Anywhere 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 M-Business Anywhere, you require the following:

- M-Business Anywhere is the new name for AvantGo M-Business Server. This software requires M-Business Server 5.3 or later, and the corresponding M-Business Anywhere client.

Target platforms

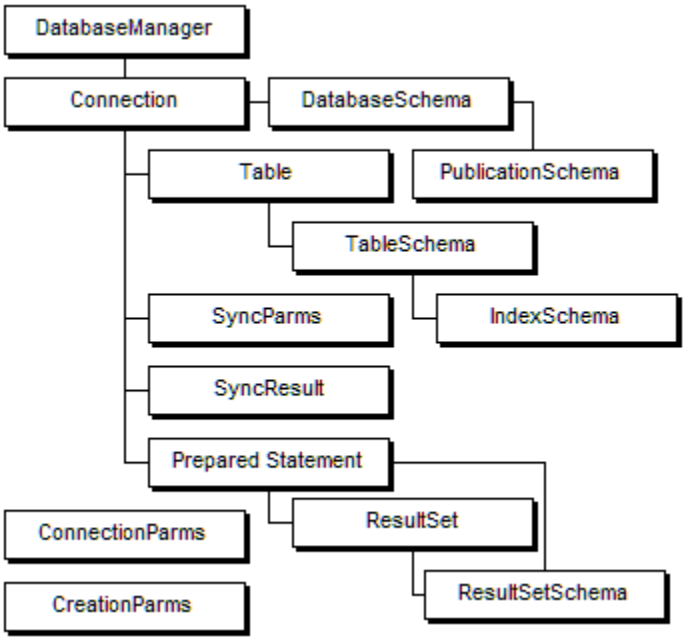
UltraLite for M-Business Anywhere supports the following target platforms:

- Windows Mobile 3.0 and later, with Pocket PC on the ARM processor, including Windows Mobile 5.0.
- Windows, starting with M-Business Anywhere 5.5.

For more information about deployment, see http://www.sybase.com/detail?id=1002288.

UltraLite for M-Business Anywhere 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.

- **DatabaseManager** manages connections to UltraLite databases. See "DatabaseManager class" on page 66.
- **ConnectionParms** holds a set of connection parameters. See "ConnectionParms class" on page 58.
- **CreationParms** holds a set of database creation parameters. See "CreationParms class" on page 61.
- **Connection** represents a database connection, and governs transactions. See "Connection class" on page 41.
- **PreparedStatement, ResultSet, and ResultSetSchema** manage database requests and their results using SQL. See:
  - "PreparedStatement class" on page 81
  - "ResultSet class" on page 91
  - "ResultSetSchema class" on page 110
- **Table** manages data using a table-based API. See "ULTable class" on page 154.
- **SyncParms and SyncResult** manage synchronization through the MobiLink server.

  For more information about synchronization with MobiLink, see "UltraLite clients" [*UltraLite - Database Management and Reference*].

Understanding UltraLite for M-Business Anywhere development

UltraLite for M-Business Anywhere quick start

The following procedures describe how to run the supplied CustDB and Simple sample applications.

Before you start, ensure that you have M-Business Anywhere 6.0 or later installed and running, and that you have administrative privileges on the server. You must also have a supported handheld device.

To install and run M-Business Anywhere samples

1. Copy the UltraLite for M-Business Anywhere sample files to your installation directory for deployment.
   a. Open a command prompt and change the directory to the *samples-dir* *\UltraLiteForMBusinessAnywhere\CustDB* subdirectory of your SQL Anywhere installation.
   b. Run the following command:

      ```
      build.bat deploy-dir
      ```

      where *deploy-dir* is the directory where the CustDB and UltraLite files are to be deployed. For example, you may choose *C:\tutorial\mba*.

      The batch file copies the required files to the location you specify. To see what files are being copied, examine the file *samples-dir\UltraLiteForMBusinessAnywhere\CustDB\build.bat* using a text editor.

2. Create a virtual directory in your web server that points to the directory *deploy-dir* specified in step 1. The following instructions are for Microsoft IIS:
   a. Open the IIS management tool.
   b. Right-click your web site and choose **New** » **Virtual Directory**. Name this virtual directory **CustDB** and specify your deployment directory *deploy-dir* as the content directory. Leave the other settings at their default values.
   c. Right-click the new virtual directory and choose **Properties**. In the **HTTP Headers** tab, click **File Types** and register the following file extensions as the type application/octet-stream:
      - For Windows and Windows Mobile: cab, dll
      - udb
   d. Make a note of the URL that accesses the file *main.htm* in this virtual directory. In a default installation this would be *http://localhost/CustDB/main.htm*.

3. Add users to M-Business Anywhere.

   There are three ways to add new users to M-Business Anywhere: by creating new user profiles, by allowing users to self-register, and by importing a CSV file. These instructions describe how to create a new user profile. For more information, see the M-Business Anywhere documentation.

a. Log in to M-Business Anywhere as an administrator.

   The default administrator account settings are a user ID of **Admin** and an empty password.

b. In the left panel, click **Users**.

c. Click **Create User**.

d. In the **User Name** field, type a unique user name.

e. In the **Password** and **Confirm Password** fields, type the password.

f. Click **Save**.

4. Deploy the M-Business Anywhere Client to a handheld device or PC.

   a. Click the **Download Client Software Only** link on the M-Business Anywhere login page. Run the installation to install the client.

   b. On the handheld device or PC, configure M-Business Connect to synchronize with the M-Business Anywhere server.

      Enter the user ID and password of the new M-Business user account you created.

   c. Synchronize to the M-Business Anywhere server.

      If you have connection problems at this stage, specify the host using an IP number rather than a host name (to avoid name resolution issues with some versions of ActiveSync).

   For more information, see your M-Business Anywhere documentation.

5. Add a group to M-Business Anywhere.

   The group will be used to test UltraLite for M-Business Anywhere.

   a. From a web browser, connect to M-Business Anywhere.

      The default URL is *http://localhost* or *http://localhost:8091*.

   b. Log in using the administrator account.

   c. In the left navigation panel, click **Groups** and then click **Create Group**.

   d. Name your group **UltraLite Samples**.

6. Configure the M-Business Anywhere channel settings.

   a. In the left panel under **Edit Group**, add the user you created in step 3 to the group UltraLite Samples using the Users option.

   b. Use the group's "Channels" option in the left navigation panel to create the following channel:

Setting	Value
Channel Name	CustDB
Location	*http://localhost/CustDB/main.htm* or the URL from step 2.
Size	1000

Setting	Value
Link depth	3
Allow binary distribution	Yes (checked)
Hidden	No (unchecked)

After setting the Location value, click **View** to confirm that you have entered the value correctly.

7. Synchronize your client.

   The initial synchronization downloads the UltraLite for M-Business Anywhere files onto your handheld device.

To verify your setup

1. Check that the required files are present.
   - On Windows Mobile, after you have synchronized the device, check that the following files are in the *\Program Files\AvantGo\Pods* folder:
     - *ulpod12.dll*
     - *custdb.udb*

     If any of these files is missing, you may have to manually copy them over to the device.
   - On Windows desktops, after you have synchronized your device, check that the following files are in the *AvantGo\Pods* subdirectory of your *AvantGo Connect* folder:
     - *ulpod12.dll*
     - *custdb.udb*

     If any of these files are missing, you may have to manually copy them over to the device.

2. Launch M-Business Client.

   On your handheld device or PC, check that the About screen displays the UltraLite for M-Business Anywhere version number. This confirms that UltraLite for M-Business Anywhere is successfully installed.

3. Run the CustDB sample application.
   a. Start the MobiLink server on your desktop.

      From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **MobiLink** » **Synchronization Server Sample**.

   b. Start the CustDB application on your M-Business Client.

      The CustDB application is a link on your M-Business home page.

   c. Enter your user ID.

      The default value is **50**.

d. Synchronize.

   At the prompt "Do you have a network connection now?" answer Yes, or in the CustDB application, click **Synchronize**. This synchronizes data with MobiLink, and is a separate operation from synchronizing with M-Business Anywhere.

   You should now see data in the **CustDB** fields. You can now explore the CustDB application.

See "Exploring the CustDB sample for MobiLink" [*MobiLink - Getting Started*].

Connecting to an UltraLite database

UltraLite applications must connect to a database before carrying out operations on the data in it.

Here is the simplest way to establish a connection. Extensions to this technique are given in the following sections.

```
var DatabaseMgr;
var Connection;
DatabaseMgr = CreateObject("iAnywhere.UltraLite.DatabaseManager.CustDB");
Connection = DatabaseMgr.openConnection("dbf=" + DatabaseMgr.directory + "\
\mydb.udb");
```

Using the Connection object

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

For more information about the Connection object, see "Connection class" on page 41.

- **Commit behavior** By default, UltraLite applications are in autoCommit mode. Each insert, update, or delete statement is committed to the database immediately. Set Connection.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 45.
- **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 22.
- **Synchronization** A set of objects governing synchronization are accessed from the Connection object. See "Synchronizing data" on page 22.
- **Tables** UltraLite tables are accessed using the Connection.getTable method. See "getTable method" on page 50.

Maintaining connections and application state across pages

The scope of a JavaScript variable is limited to one web page. Most web applications require multiple pages, and so a mechanism is needed for making some objects persistent across the pages of an application.

UltraLite for M-Business Anywhere provides persistence for the ULTable, ResultSet, and PreparedStatement objects. To make one of these objects persist across pages, supply a **persistent name** as a parameter when creating the object. You can use the persistent name on subsequent pages.

To carry a connection object from page to page, you reopen the connection on each page. One way to do this is to use the reOpen method. Another is to supply an open method on each page, perhaps by including a JavaScript file on each web page to initialize the settings. For examples of how to do this, see the sample files *samples-dir\UltraLiteForMBusinessAnywhere\CustDB\main.htm* and *samples-dir\UltraLiteForMBusinessAnywhere\Simple\main_page.htm*.

The requirement to reopen connections across pages provides a security feature for UltraLite applications. You can use it to require that the user confirm some information, perhaps the password, on moving from page to page.

If an UltraLite object is not needed in another web page, the application should issue a close method on the object to save memory.

See also

- "reOpenConnection method" on page 69
- "PreparedStatement class" on page 81
- "ResultSet class" on page 91
- "ULTable class" on page 154

Persistent names in M-Business Anywhere applications

In HTML, when control transfers to a new page, all handles to allocated JavaScript objects from the old page are lost. For example, in main.html, you have a M-Business Anywhere database connection object:

```
conn = dbMgr.openConnection("...");
```

If you click a link in *main.html* and it takes you to a different page (for example: *insert.html*), you cannot find the object named "conn" in *insert.html*. To get the connection object back, you may have to call dbMgr.openConnection("...") again. However, you do not have to do this since the connection object is still in memory, you have only lost the JavaScript handle to it.

This is why there is a persistName argument in all the M-Business Anywhere API calls to DataManager, Connection, ULTable, PreparedStatement, or ResultSet. For example, when the M-Business Anywhere runtime receives a call from JavaScript for an UltraLite connection object, M-Business Anywhere first checks to see if a connection object exists in memory that has the same persistName. If the runtime can find a matching object, it will return that connection object. Otherwise, M-Business Anywhere goes through the normal procedure to make a new UltraLite database connection and return it.

Using persistent names

There are two types of hierarchy among M-Business Anywhere objects. They both start with DatabaseManager and Connection:

- DatabaseManager -> Connection -> Table (for table API)
- DatabaseManager -> Connection -> PreparedStatement -> ResultSet (for Dynamic SQL API)

To retrieve any of these M-Business Anywhere objects with a persistent name, you have to retrieve the top-level object with a persistent name, and then all the upper level M-Business Anywhere objects along the hierarchy tree until the one you want.

For example, if you want to retrieve an existing ULTable object from insert.html, you need to give a persistent name to dbMgr, conn, and table objects in *main.html*, and then use persistent names in *insert.html* to get all of them back:

Code segment for *main.html*:

```
var dbMgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.simple" );
// "simple" is the persistent name here. A real database manager object is
allocated

var conn = dbMgr.openConnection( "CON=simple_con;..." );
// "simple_con" is the persistent name here. A real database connection is
made.

var custTable = conn.getTable( "ULCustomer", "simpleCustTable" );
// a real table is allocated
```

Code segment for *insert.html*:

```
var dbMgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.simple" );
// "simple" is the persistent name here.
// The allocated database manager object from main.html is returned

var conn = dbMgr.openConnection( "CON=simple_con;..." );
// "simple_con" is the persistent name here.
// The existing connection object from memory is returned.

var custTable = conn.getTable( "ULCustomer", "simpleCustTable" );
// the existing table object is returned.

var newTable = conn.getTable( "ULOrder", "simpleOrderTable" );
// since there is no order table from main.html,
// it does not exist in memory. A real order table object is allocated.
```

Using persistent names correctly

Put the commonly used code in a JavaScript file. Since most HTML pages of an M-Business Anywhere application need to refer to DatabaseManager, Connection, and some major ULTable objects, it is convenient to put the code that creates them (or retrieves them with a persistent name) in a common JavaScript file, and include this file at the top of HTML pages that use them. Both M-Business Anywhere "simple" and "CustDB" sample programs demonstrate how to do this.

Close the object if you do not plan to use the object from another page. If the M-Business Anywhere application only has one HTML page, then there is no need to have persistent names. The persistent name argument can be set to NULL. On the other hand, if each HTML page has many opened

PreparedStatement and ResultSet objects, then the developer needs to balance between the convenience of having them in memory to retrieve them easily with a persistent name from another html page, and wasted memory usage because these objects are always around. For example, suppose you have 5 PreparedStatement objects and 10 ResultSet objects created in *main.html*. They are occupying a significant amount of memory. When the application jumps to *insert.html*, if you only need to refer to some of these objects with a persistent name, then the objects that are not needed anymore are wasting memory. If you try to create new PreparedStatement and ResultSet objects in *insert.html*, you may run out of memory. The solution is to explicitly close those PreparedStatement object or ResultSet object at the end of *main.html* if you are sure you do not need them from *insert.html*.

The state of each M-Business Anywhere object is preserved when it is retrieved with a persistent name. If you have a persistent ULTable object from page 1, when you call openTable method from page 2 using the same persistent name, you get the exact ULtable object back with the same state as the one from page 1. If the cursor is on the nth row of the table when you leave page 1, the cursor will be still on the nth row when you get it back in page 2. It will not be "before first row".

Be careful using the persistent name on ResultSet. When there are place holders on the PreparedStatement, you need to be very careful about whether you want to give a persistent name to the ResultSet. For example, in *main.html* you have the following code:

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" );

OrderStmt.setInt(1, 5000);

var OrderResultSet = OrderStmt.executeQuery( "order_query_result" );
```

Then from *insert.html*, you want the same ResultSet object, you must do the following:

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" );

//OrderStmt.setInt(1, 5000); // no need to do this since both the OrderStmt
and
OrderResultSet are retrieve from "cache" without any SQL statement being
actually executed

var OrderResultSet = OrderStmt.executeQuery( "order_query_result" );
```

This OrderResultSet object contains the same result as the "order_id" set to 5000.

However, consider a different situation. You want the same PreparedStatement because you want to do the same query on the Order Table. But you want to query with an order ID other than 5000. In this case, you can assign a persistent name to the PreparedStatement, but you don't need a persistent name on the ResultSet. Since the order id is different this time, the result set will be different from the previous one. In *main.html*, you still do the following:

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" ); // with persistent name

OrderStmt.setInt(1, 5000);
```

```
var OrderResultSet = OrderStmt.executeQuery( null ); // notice here, no
persistent name
```

In *insert.html*, you do the following to get a new ResultSet:

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" ); // get the prepared statement from memory with
persistent name

OrderStmt.setInt(1, 6000); // set a different place holder value

var OrderResultSet = OrderStmt.executeQuery( null ); // a real query is
executed
here!
```

In the example above, since the place holder value is different, or some other operation is performed on the Order table that you expect the returned result set will be different, you do not use persistent name on the ResultSet when calling executeQuery.

Database encryption and obfuscation

You can encrypt or obfuscate your UltraLite database using UltraLite for M-Business Anywhere.

For more information about database encryption, see "Securing UltraLite databases" [*UltraLite - Database Management and Reference*].

Encryption

UltraLite databases may be unencrypted or may employ either encryption or obfuscation. If you want the database to be encrypted or obfuscated that choice must be made when the database is created.

Encryption of an UltraLite database uses extremely strong industry-standard techniques to encrypt the data in the database. The encryption is based on a key phrase that is specified when the database is created. This key phrase must also be supplied when a connection is made to the database.

If an UltraLite database is encrypted, all connections to that database must specify the correct encryption key or the connection attempt fails.

For more information about the EncryptionKey property, see "ConnectionParms class" on page 58 and "changeEncryptionKey method" on page 45.

Obfuscation

Obfuscation is a very weak form of encryption that simply masks the data in the database to discourage casual observation of the contents of the database by file or disk viewer programs. To obfuscate the database, set the creationParms.obfuscate boolean value to true. For example:

```
var create_parms = dbMgr.createCreationParms();
create_parms.obfuscate = true;
```


Example

You can change the encryption key by specifying a new encryption key on the Connection object. Before calling the changeEncryptionKey method, the application must make a connection to the encrypted database using the existing encryption key. In the following example code, "apricot" is the new encryption key.

```
conn.changeEncryptionKey("apricot")
```

Working with data using SQL

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

For information about the Table API, see “Working with data using the Table API” on page 15.

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

- Inserting, deleting, and updating rows.
- Executing a query.
- Scrolling through the rows of a result set.

This section does not describe the SQL language itself. For more information about SQL features, see “SQL Anywhere Server - SQL Reference”.

Data manipulation: INSERT, UPDATE, and DELETE

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

For more information the PreparedStatement class, see “PreparedStatement class” on page 81.

Parameter markers in prepared statements
UltraLite handles variable values using the ? parameter marker. For any INSERT, UPDATE or DELETE, each ? is referenced according to its ordinal position in the prepared statement. For example, the first ? is referred to as 1, and the second as 2.

To insert a row

1. Declare a PreparedStatement object.

   ```
   var PrepStmt;
   ```

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

```
PrepStmt = conn.prepareStatement(
    "INSERT into TableName(ColumnName) values (?)", null );
```

The null parameter indicates that the statement has no persistent name.

3. Assign parameter values to the statement.

```
var NewValue;
NewValue = "Bob";
PrepStmt.setStringParameter(1, NewValue);
```

4. Execute the statement.

```
PrepStmt.executeStatement( null );
```

To update a row

1. Declare a PreparedStatement object.

```
var PrepStmt;
```

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

```
PrepStmt = conn.prepareStatement(
  "UPDATE TableName SET ColumnName = ? WHERE ID = ?", null);
```

The null parameter indicates that the statement has no persistent name.

3. Assign parameter values to the statement using methods appropriate for the data type.

```
var NewValue;
NewValue = "Bob";
PrepStmt.setStringParameter(1, NewValue);
PrepStmt.setIntParameter(2, 6);
```

4. Execute the statement.

```
PrepStmt.executeStatement( );
```

To delete a row

1. Declare a PreparedStatement object.

```
var PrepStmt;
```

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

```
PrepStmt = conn.prepareStatement(
    "DELETE FROM customer WHERE ID = ?", null );
```

The null parameter indicates that the statement has no persistent name.

3. Assign parameter values for the statement.


```
var IDValue;
IDValue = 6;
PrepStmt.setIntParameter( 1, IDValue );
```

4. Execute the statement.

```
PrepStmt.executeStatement( );
```

Data retrieval: SELECT

When you execute a SELECT statement, the PreparedStatement.executeQuery method returns a ResultSet object. The ResultSet class contains methods for navigating within a result set and methods to update data using the ResultSet.

For more information about ResultSet objects, see "ResultSet class" on page 91.

Example

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

```
var MyResultSet;
var PrepStmt;
PrepStmt = conn.prepareStatement("SELECT ID, Name FROM customer", null );
MyResultSet = PrepStmt.executeQuery( null );
MyResultSet.moveFirst();
```

Example

The following code demonstrates how to obtain column values for the current row. The example uses character data; similar methods are available for other data types.

The getString method uses the following syntax: MyResultSetName.getString(*Index*) where *Index* is the ordinal position of the column name in your SELECT statement.

```
if ( MyResultSet.getRowCount() == 0 ) {
} else {
  alert( MyResultSet.getString(1) );
  alert( MyResultSet.getString(2) );
  MyResultSet.moveRelative(0);
}
```

For more information about navigating a result set, see "Navigation with SQL" on page 14.

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

To perform a select statement

1. Declare a PreparedStatement object.

```
var OrderStmt;
```

2. Assign a prepared statement to your PreparedStatement object.

```
OrderStmt = Connection.prepareStatement(
   "SELECT order_id, disc, quant, notes, status, c.cust_id,
   cust_name, p.prod_id, prod_name, price
  FROM ULOrder o, ULCustomer c, ULProduct p
  WHERE o.cust_id = c.cust_id
  AND o.prod_id = p.prod_id
  ORDER BY der_id", "order_query_stmt" );
```

   The second parameter is a persistent name that provides cross-page JavaScript object persistence.

3. Execute the query.

```
OrderResultSet = OrderStmt.executeQuery( "order_query" );
```

For more information about how to use queries, see the CustDB sample code in *samples-dir\UltraLiteForMBusinessAnywhere\CustDB\custdb.js*.

Navigation with SQL

UltraLite for M-Business Anywhere provides you with many methods to navigate a result set to perform a wide range of navigation tasks.

The following methods of the ResultSet 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 fragment demonstrates how to use the moveFirst method to navigate within a result set.

```
PrepStmt = conn.prepareStatement(
    "SELECT ID, Name FROM customer", null );
MyResultSet = PrepStmt.executeQuery( null );
MyResultSet.moveFirst();
```

The same technique is used for all the move methods.

For more information about these navigational methods, see “ResultSet class” on page 91.

The ResultSetSchema object

The ResultSet.schema property allows you to retrieve information about the columns in the query.

The following example demonstrates how to use ResultSetSchema to capture schema information.

```
var I;
var MySchema = rs.schema ;
for ( I = 1; I <= MySchema.columnCount; I++) {
  colname = MySchema.getColumnName(I);
  coltype = MySchema.getColumnSQLType(colname).toString();
  alert ( colname + " " + coltype );
}
```

Working with data using the Table API

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

For information about SQL, see "Working with data using SQL" on page 11.

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 M-Business Anywhere provides you with many 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 an alert with the last name of each customer.

```
var tCustomer;
tCustomer = conn.getTable( "customer", null );
tCustomer.open();
tCustomer.moveBeforeFirst();
While (tCustomer.moveNext()) {
  alert( tCustomer.getString(3) );
}
```

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.

```
tCustomer = conn.getTable("customer", null );
tCustomer.openWithIndex("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 one of the ULTable get methods to get the value of each column for the current row.

Example

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

```
var colID, colFirstName, colLastName;
colID = tCustomer.schema.getColumnID( "id" );
colFirstName = tCustomer.schema.getColumnID( "fname" );
```



```
colLastName = tCustomer.schema.getColumnID( "lname" );
alert( tCustomer.getInt( colID ) );
alert( tCustomer.getString( colFirstName ) );
alert( tCustomer.getString( colLastName ) );
```

You can also use methods of ULTable to set values.

```
tCustomer.setString( colLastName, "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 fragment generates an error.

```
tCustomer.moveBeforeFirst();
id = tCustomer.getInt( colID );
```

Searching rows with find and lookup

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

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

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

  For more information about find methods, see "ULTable class" on page 154.

- **Lookup methods** move to the first row that matches or is greater than a specified search value, under the sort order specified when the ULTable object was opened.

  For more information about lookup methods, see "ULTable class" 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 fragment 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 fragment sets the last name column in the buffer to Kaminski.

```
tCustomer.setString(3, "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 first column in the buffer to 3.

```
tCustomer.setInt( 1 , 3 );
```

Using UltraLite modes

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

- **Insert mode** The data in the buffer is added to the table as a new row when the ULTable.insert method is called.
- **Update mode** The data in the buffer replaces the current row when the ULTable.update method is called.
- **Find mode** Used to locate a row whose value exactly matches the data in the buffer when one of the ULTable.find methods is called.
- **Lookup mode** Used to locate a row whose value matches or is greater than the data in the buffer when one of the ULTable.lookup methods is called.

To update a row

1. Move to the row you 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.

   ```
   tCustomer.setString( 2, "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. For more information about autoCommit mode, see "Managing transactions" on page 20.

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

Inserting rows

The steps to insert a row are similar to those for updating rows, except that there is no need to locate any particular row in the table before carrying out the insert operation. Rows are automatically sorted by the index specified when opening the table.

To insert a row

1. Enter Insert mode.

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

   ```
   tCustomer.insertBegin();
   ```

2. Set the values for the new row.

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

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

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

   ```
   colID = tCustomer.schema.getColumnID( "id" );
   colFirstName = tCustomer.schema.getColumnID( "fname" );
   colLastName = tCustomer.schema.getColumnID( "lname" );
   tCustomer.setInt( colID, 42 );
   ```

```
tCustomer.setString( colFirstName, "Mitch" );
tCustomer.setString( colLastName, "McLeod" );
```

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.

   ```
   tCustomer.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 delete:

   ```
   tCustomer.deleteRow();
   ```

Working with BLOB data

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

See “getBytes method” on page 164 and “getBytesSection method” on page 165.

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 for M-Business Anywhere 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 Connection.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 Connection.commit() statement to complete a transaction and make changes to your database permanent, or you may execute a Connection.rollback() statement to cancel all the operations of a transaction. Turning autoCommit off improves performance.

Note
Synchronization causes a commit even if you have autoCommit set to false.

Accessing schema information

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

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

  To obtain a DatabaseSchema object, access the Connection.databaseSchema property.

- **TableSchema** The number and names of columns in the table, and the Indexes collections for the table.

  To obtain a TableSchema object, access the ULTable.schema property.

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

  The IndexSchema objects are accessed using the TableSchema.getIndex method.

- **PublicationSchema** The numbers and names of tables and columns contained in a publication. Publications include the PublicationSchema object, but not the Publication object.

  The PublicationSchema objects are accessible using the DatabaseSchema.getPublicationSchema method.

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

  The ResultSetSchema objects are accessible using the PreparedStatement.getResultSetSchema method or the ResultSet.schema property.

Handling errors

In normal operation, UltraLite for M-Business Anywhere can throw errors that are intended to be caught and handled in the script environment. See “Error Messages”.

Errors are expressed as SQLCODE values, negative numbers indicating the particular kind of error.

UltraLite for M-Business Anywhere throws errors only from the DatabaseManager and Connection objects. The following methods of DatabaseManager can throw errors:

- createDatabase
- dropDatabase
- openConnection

All other errors and exceptions within UltraLite for M-Business Anywhere are routed through the Connection object.

You can access error numbers from DatabaseManager and Connection objects. See:

- "Connection class" on page 41
- "DatabaseManager class" on page 66

Authenticating users

New users have to be added from an existing connection. As all UltraLite databases are created with a default user ID and password of **DBA** and **sql**, respectively, you must first connect as this initial user.

You cannot change a user ID: you add a user and delete an existing user. A maximum of four user IDs are permitted for each UltraLite database.

For more information about granting or revoking connection authority, see "grantConnectTo method" on page 51 and "revokeConnectFrom method" on page 53.

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

1. Connect to the database as an existing user.
2. Grant the user connection authority with the desired password.

```
conn.grantConnectTo("Robert", "newPassword");
```

To delete an existing user

1. Connect to the database as an existing user.
2. Revoke the user's connection authority as follows.

```
conn.revokeConnectFrom("Robert");
```

Synchronizing data

UltraLite for M-Business Anywhere applications typically involve two kinds of synchronization:

- **Web content synchronization** Web content, including HTML pages that define the application itself, is synchronized through M-Business Anywhere.
- **Data synchronization** The UltraLite database is synchronized with a MobiLink server.

Synchronizing data

Synchronization requires the MobiLink server and appropriate licensing. You can find a working example of synchronization in the CustDB sample application.

UltraLite for M-Business Anywhere supports TCP/IP, HTTP, and HTTPS synchronization. Synchronization is initiated by the UltraLite application. Always use the methods and properties of the Connection 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 12 - Introduction*].

To synchronize over TCP/IP or HTTP

1. Prepare the synchronization information.

   Assign values to the required properties of the Connection.syncParms object.

   For more information about the properties and the values that you should set, see "UltraLite clients" [*UltraLite - Database Management and Reference*].

2. Synchronize.

   Call the Connection.synchronize method.

Synchronizing data via M-Business Anywhere

To synchronize data via M-Business Anywhere

1. In your client, set synchronization parameters so that UltraLite synchronization is directed to the host and port number of M-Business Anywhere. You can use the SyncParms.setMBAServer method to carry out this task. See "setMBAServer method" on page 130.

2. From a client application, initiate synchronization using separate data synchronization. See "Synchronizing data" on page 23.

Deploying UltraLite for M-Business Anywhere 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 applications to Windows Mobile and Windows desktops

You must carry out the following steps to deploy an UltraLite application to a Windows Mobile device:

- Deploy your application and UltraLite component. See "UltraLite for M-Business Anywhere quick start" on page 3.
- Deploy an initial copy of the UltraLite database. See "UltraLite for M-Business Anywhere quick start" on page 3.

  In many situations it is enough to deploy an UltraLite database. You can use synchronization to load an initial copy of the data.

  You must place the database so that it can be located by the application. The Database On CE connection parameter defines the location for Windows Mobile. The Database on Desktop connection parameter defines the location for Windows. See:

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

Deploying applications requiring synchronization

Synchronization requires a set of files, including: *ulconnect.exe*, *ulconnect.udb*, *ulpod12.dll*, and *ulrt12.dll*. For Windows Mobile, these files are located in file *ulpod.cab* in the directory *install-dir\ultralite\UltraLiteForMBusinessAnywhere\ce\arm*. When you deploy the cab file to a Windows Mobile device, it installs its contents in the proper locations automatically. For Windows, the required files must be deployed manually from the directory *install-dir\ultralite\UltraLiteForMBusinessAnywhere\win32\386*.

Tutorial: A sample application for M-Business Anywhere

Introduction to the M-Business Anywhere development tutorial

This tutorial describes how to build a cross-platform UltraLite for M-Business Anywhere application. The application will also be able to synchronize with a consolidated database.

Timing

The tutorial takes about 60 minutes to complete assuming you copy and paste the code. Complete code samples for *main.htm* and *tutorial.js* are provided in "Listings for main.htm and tutorial.js" on page 34 at the end of this tutorial.

Prerequisites

The tutorial assumes a basic familiarity with JavaScript programming, developing mobile applications in the M-Business Anywhere environment, and managing databases using Sybase Central.

See also

- "Create a database with the Create Database Wizard" [*UltraLite - Database Management and Reference*]
- "Creating and configuring UltraLite databases" [*UltraLite - Database Management and Reference*]

Lesson 1: Set up databases

This lesson describes how to create a remote database for the tutorial and how to deploy a synchronization model to a consolidated database.

To set up a database

1. Create a directory for this tutorial. This tutorial assumes the directory is *c:\tutorial*. If you create a directory with a different name, use that directory throughout the tutorial.
2. Create an UltraLite database using Sybase Central with the following information. (For more information about creating a remote database, see "Create a database with the Create Database Wizard" [*UltraLite - Database Management and Reference*].)
   - **Table name** Customer
   - **Columns**

Column Name	Data Type (Size)	Column allows NULL values?	Default value
ID	integer	No	autoincrement
GivenName	char(15)	No	None
Surname	char(20)	No	None
City	char(20)	Yes	None
Phone	char(12)	Yes	None
Street	char(50)	No	None

3. Save the remote database file for the following platforms:
   - **Windows** *c:\tutorial\WIN32_OS\tutorial.udb*
   - **Windows Mobile** *c:\tutorial\WIN32_CE\tutorial.udb*

To deploy a synchronization model

1. Create a synchronization model from Sybase Central. For more information about creating a synchronization model, see "Creating synchronization models" [*MobiLink - Getting Started*].

   Use *c:\tutorial* as the work folder for the synchronization model file. For the **Consolidated Database Schema** page and the **Remote Database Schema** page, use the following settings:

   a. For the **Consolidated Database Schema**, use the SQL Anywhere sample database, *demo.db*, to obtain the schema.

   b. For the **Remote Database Schema**, use the UltraLite sample database, *tutorial.udb* (or *.pdb*), to obtain the schema.

   c. Use the defaults for all other pages of the wizard.

2. Deploy a synchronization model from Sybase Central. For more information about deploying synchronization models, see "Deploying synchronization models" [*MobiLink - Getting Started*].

   a. For the **Consolidated Database Deployment Destination** page, use the SQL Anywhere sample database as the consolidated database.

   b. For the **Remote Database Deployment** page choose the **Existing SQL Anywhere or UltraLite database** option to deploy the synchronization model to the UltraLite database, *tutorial.udb* (or *.pdb*).

   c. For the **Existing Remote Database** page, uncheck the **connect to the remote database directly to apply the changes** option.

   d. For the **MobiLink User** page, specify the following settings for connecting to the MobiLink server:
      - **user name** tutorial
      - **password** tutorial

e. Use the defaults for all other pages of the wizard.

A command file, *tutorial_mlsrv.bat*, should be generated when you complete the **Deploy Synchronization Model Wizard**. This command file will be used later in the tutorial.

Lesson 2: Create the application files

This lesson describes how to set up the application files.

Create the application files

1. Create the file *c:\tutorial\main.htm*.

   Later in the tutorial, you will add more logic to *main.htm*. For now, you will simply set it up to include a platform-specific file, *ul_deps.html*.

   Add the following content to *main.htm*:

   ```
   <html>
   <body>
   <a href="AG_DEVICEOS/ul_deps.html"></a>
   </body>
   </html>
   ```

2. Create the platform-specific file, *ul_deps.html*.

   This file references specific binaries for different operating systems, as follows:

   - **Windows** *c:\tutorial\WIN32_OS\ul-deps.htm*

     ```
     <!-- WIN32_OS\ul_deps.html -->
     <html>
       <a href="ulpod12.dll"></a>
       <a href="tutorial.udb"></a>
     </html>
     ```

   - **Windows Mobile** *c:\tutorial\WIN32_CE\ul-deps.htm*

     ```
     <!-- WIN32_CE\ul_deps.html -->
     <html>
       <a href="AG_DEVICEPROCESSOR/ulpod12.dll"></a>
       <a href="tutorial.udb"></a>
     </html>
     ```

3. Copy the UltraLite Pod file, *ulpod12.dll* for Windows and Windows Mobile to the *tutorial* directory.

   - For Windows desktops, copy *ulpod12.dll* from *install-dir\UltraLite\UltraLiteForMBusinessAnywhere\win32\386* to *c:\tutorial\WIN32_OS*.
   - For Windows Mobile, copy *ulpod12.dll* from *install-dir\UltraLite\UltraLiteForMBusinessAnywhere\CE\Arm* to *c:\tutorial\WIN32_CE\arm*.

   All application files are now in place.

Lesson 3: Configure M-Business Anywhere

This lesson describes how to configure M-Business Anywhere and synchronize the UltraLite tutorial channel.

Configure M-Business Anywhere

1. Open the M-Business Anywhere administration console and log in as **Admin** (without a password).
2. Create a new user named *tutorial*.
3. Create a channel for the user:
   a. Use the following settings for the channel. Make sure to substitute the correct URL for your web server:
      - **Channel Title** UltraLite Tutorial.
      - **Channel URL** *http://<yourwebserver>/tutorial/main.htm*.

        The location is the URL of the tutorial *main.htm* page, as served by your web server.
      - **Channel Size** 1000 KB.
      - **Channel Link Depth** 3.
      - **Allow Binary Distribution** true (checked).
      - **Hide From Users** false (unchecked).

The user and channel are now set up on M-Business Anywhere. The next step is to synchronize the content of this channel to an M-Business client. You can do this from whichever platform you want to use.

The next procedure assumes that you have an M-Business client installed. It is recommended that you click **Tools** » **Options** and set the client options to **Show JavaScript Errors**. This setting provides for easier debugging of any errors in your application.

Synchronize the channel for your device

- Synchronize the UltraLite channel to your device.

  At this stage there is no content for your application, so the page appears blank.

For details on the M-Business Anywhere environment, refer to your *M-Business Anywhere Application Developer's Guide*.

Lesson 4: Add startup code to your application

This lesson provides the startup code to your application to connect to an UltraLite database.

Add content to your application

1. Add the following content to *main.htm*, immediately before the <a> tag:

```
<form name="form">
<br><td> ID: </td>
    <td> <input type="text" name="ID" size="10"> </td>
<br><td> Given Name: </td>
    <td> <input type="text" name="GivenName" size="15">
    </td>
<br><td> Surname: </td>
    <td> <input type="text" name="Surname" size="50"> </td>
<br><td> Street: </td>
    <td> <input type="text" name="Street" size="20"> </td>
<br><td> City: </td>
    <td> <input type="text" name="City" size="20"> </td>
<br><td> Phone: </td>
    <td> <input type="text" name="Phone" size="12"> </td>
<br>
<br>

<table>
<tr>
    <td> <input type="button" value="Insert"
         onclick="ClickInsert();"> </td>
    <td> <input type="button" value="Next"
         onclick="ClickNext();"> </td>
    <td> <input type="button" value="Prev"
         onclick="ClickPrev();"> </td>
</tr>
<tr>
    <td colspan=3>
    <input type="button" value="Synchronize"
    onclick="ClickSync();">
    </td>
</tr>
</table>
</form>
```

2. Create a JavaScript file *c:\tutorial\tutorial.js* to provide application logic.

3. Add the following variables declaration to *tutorial.js* for the UltraLite Pod object :

```
var DB_mgr;
var Connection;
var Table;
```

4. Add the following function to *tutorial.js* to connect to the tutorial database:

```
function Connect()
{
    var     dir;
    var     open_parms;
    var     browser = navigator.platform;

    DB_mgr =
CreateObject( "iAnywhere.UltraLite.DatabaseManager.Tutorial" );
    if( DB_mgr == null ) {
     alert( "Error: cannot create database manager: " + DB_mgr.sqlCode );
 return;
    }
```

```
    dir = DB_mgr.directory;
    open_parms = "con=tutorial;" + "file_name=" + dir + "\\tutorial.udb";
    }
    try {
 Connection = DB_mgr.reOpenConnection( "tutorial" );
     if( Connection == null ) {
     Connection = DB_mgr.openConnection( open_parms );
 }
    } catch( ex ) {
     if( DB_mgr.sqlCode !=
         DB_mgr.SQLError.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
         alert( "Error: cannot connect to database: " + ex.getMessage() );
     return;
     }
}
```

5. Use the onload event handler to connect to the database when the application is started. Modify *main.htm* as follows:

   a. Load *tutorial.js* by adding the following line immediately before the <body> tag:

   ```
   <script src="tutorial.js"></script>
   ```

   b. Modify the <body> tag:

   ```
   <body onload="Connect();">
   ```

6. Test your application.

   Synchronize the UltraLite tutorial channel. The synchronization application should be connecting to the tutorial database.

Lesson 5: Add data manipulation and navigation

This lesson describes how to fill out your application with data manipulation and navigation logic.

Initialize the table

1. Initialize the CustomerTable representing the **Customer** table in your database by adding the following code to the end of the **Connect** function in *tutorial.js*:

   ```
   try {
     CustomerTable = Connection.getTable( "customer", null );
     CustomerTable.open();
   } catch( ex3 ) {
     alert("Error: " + ex3.getMessage() );
   }
   ```

2. Add variables to move data between the database and the web form.

   Add the following variable declarations to the top of *tutorial.js* for the customer data:

   ```
   var GivenName = "";
   var Surname = "";
   var Street = "";
   var City = "";
   ```



```
var Phone = "";
var ID = "";
```

3. Create functions to fetch and display customer data.

   Add the following function to *tutorial.js*. This function fetches the current row of the customer data and also ensures that NULL columns are displayed as empty strings:

```
function Fetch()
{
    if( CustomerTable.getRowCount() == 0 ) {
 GivenName = "";
 Surname = "";
 Street = "";
 City = "";
 Phone = "";
 ID = "";
 return;
    }
    ID =
CustomerTable.getString( CustomerTable.schema.getColumnID( "ID" ) );
    GivenName =
CustomerTable.getString( CustomerTable.schema.getColumnID( "GivenName" ) )
;
    Surname =
CustomerTable.getString( CustomerTable.schema.getColumnID( "Surname" ) );
    Street =
CustomerTable.getString( CustomerTable.schema.getColumnID( "Street" ) );

if( CustomerTable.isNull( CustomerTable.schema.getColumnID( "City" ) ) ) {
 City = "";
    } else {
 City =
CustomerTable.getString( CustomerTable.schema.getColumnID( "City" ) );
    }

if( CustomerTable.isNull( CustomerTable.schema.getColumnID( "Phone" ) ) )
{
 Phone = "";
    } else {
 Phone =
CustomerTable.getString( CustomerTable.schema.getColumnID( "Phone" ) );
    }
}
```

   Add the following to *main.htm* immediately after the <script> tag. **DisplayRow** takes the values from the database and displays them in the web form. **FetchForm** takes the current values in the web form and makes them available to the database code.

```
<script>
function DisplayRow() {
    Fetch();
    document.form.ID.value = ID;
    document.form.GivenName.value = GivenName;
    document.form.Surname.value = Surname;
    document.form.Street.value = Street;
    document.form.City.value = City;
    document.form.Phone.value = Phone;
}

function FetchForm() {
    GivenName = document.form.GivenName.value;
```

```
    Surname = document.form.Surname.value;
    Street = document.form.Street.value;
    City = document.form.City.value;
    Phone = document.form.Phone.value;
}
</script>
```

4. Call **DisplayRow** to display the current row when the application is loaded. Modify the body tag at the top of *main.htm* as follows:

```
<body onload="Connect(); DisplayRow();">
```

Although there is no data in the tutorial database up to this point, this is a good place to synchronize the channel to ensure that the application is running properly.

Add code to insert rows

- Create functions to insert customer data.

  In the following procedure, the call to **InsertBegin** puts the application into insert mode and sets all values in the current row to their defaults. For example, the ID column receives the next autoincrement value. The column values are set and the new row is inserted.

  Add the following function to *tutorial.js*:

```
function Insert()
{
    try {
 CustomerTable.insertBegin();
 CustomerTable.setString( CustomerTable.schema.getColumnID( "GivenName" ),
GivenName );
 CustomerTable.setString( CustomerTable.schema.getColumnID( "Surname" ),
Surname );
 CustomerTable.setString( CustomerTable.schema.getColumnID( "Street" ),
Street );
 if( City.length > 0 ) {
     CustomerTable.setString( CustomerTable.schema.getColumnID( "City" ),
City );
 }
 if( Phone.length > 0 ) {
     CustomerTable.setString( CustomerTable.schema.getColumnID( "Phone" ),
Phone );
 }
 CustomerTable.insert();
 CustomerTable.moveLast();
    } catch( ex ) {
 alert( "Error: cannot insert row: " + ex.getMessage() );
    }
}
```

  Add the following function to *main.htm*:

```
function ClickInsert()
{
    FetchForm();
    Insert();
    DisplayRow();
}
```


Lesson 6: Add navigation to your application

This lesson describes code for moving forward and backward through the rows of the **Customer** table.

Add navigation code to your application

1. Add the **Next** function to *tutorial.js*:

```
function Next()
{
    if( ! CustomerTable.moveNext() ) {
 CustomerTable.moveLast();
    }
}
```

2. Add the **Prev** function to *tutorial.js*:

```
function Prev()
{
    if( ! CustomerTable.movePrevious() ) {
 CustomerTable.moveFirst();
    }
}
```

3. Add the following functions to *main.htm*:

```
function ClickNext()
{
    Next();
    DisplayRow();
}

function ClickPrev()
{
    Prev();
    DisplayRow();
}
```

4. When the form is first displayed, the controls are empty because 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.

Lesson 7: Add synchronization to your application

The following procedure implements synchronization.

Add a synchronization function to your application

1. Add the **Synchronize** function to *tutorial.js*.

```
function Synchronize()
{
    var sync_parms;

    sync_parms = Connection.syncParms;
    sync_parms.setUserName( "tutorial" );
    sync_parms.setPassword( "tutorial" );
    sync_parms.setVersion( "tutorial" );
```



```
        sync_parms.setStream( sync_parms.STREAM_TYPE_TCPIP );
        try {
    Connection.synchronize();
        } catch( ex ) {
         alert( "Error: cannot synchronize: " + ex.getMessage() );
        }
    }
```

The synchronization parameters are stored in the SyncParms object. For example, the SyncParms.userName property specifies the user name for which MobiLink searches. The SyncParms.sendColumnNames property specifies that the column names are sent to MobiLink so that it can generate upload and download scripts.

This function uses a TCP/IP synchronization stream and the default network communication options (stream parameters). These default options assume that you are synchronizing from either a Windows Mobile client connected to the computer running the MobiLink server via ActiveSync, or from a 32-bit Windows desktop client running on the same computer as MobiLink. If this is not the case, change the synchronization stream type and set the network communication options to the appropriate values.

See also:

- "setStream method" on page 133
- "setStreamParms method" on page 133

2. Add the following function to *main.htm*:

```
function ClickSync()
{
    Synchronize();
    DisplayRow();
}
```

3. Start the MobiLink server using the *tutorial_mlsrv.bat* file.

The data in the **Customer** table should now be downloaded. You should be able to move through the rows in the **Customer** table of the remote database.

This completes the tutorial. See "Listings for main.htm and tutorial.js" on page 34 for complete code samples for *main.htm* and *tutorial.js*.

Listings for main.htm and tutorial.js

Following is a complete listing of *main.htm* for your use:

```
<html>
<script src="tutorial.js"></script>

<script>
function DisplayRow() {
    Fetch();
    document.form.ID.value = ID;
    document.form.GivenName.value = GivenName;
    document.form.Surname.value = Surname;
    document.form.Street.value = Street;
    document.form.City.value = City;
```



```
    document.form.Phone.value = Phone;
}

function FetchForm() {
    GivenName = document.form.GivenName.value;
    Surname = document.form.Surname.value;
    Street = document.form.Street.value;
    City = document.form.City.value;
    Phone = document.form.Phone.value;
}

function ClickInsert()
{
    FetchForm();
    Insert();
    DisplayRow();
}

function ClickNext()
{
    Next();
    DisplayRow();
}

function ClickPrev()
{
    Prev();
    DisplayRow();
}

function ClickSync()
{
    Synchronize();
    DisplayRow();
}
</script>

<body onload="Connect(); DisplayRow();" >

<form name="form">
<br><td> ID: </td>
    <td> <input type="text" name="ID" size="10"> </td>
<br><td> Given Name: </td>
    <td> <input type="text" name="GivenName" size="15"> </td>
<br><td> Surname: </td>
    <td> <input type="text" name="Surname" size="50"> </td>
<br><td> Street: </td>
    <td> <input type="text" name="Street" size="20"> </td>
<br><td> City: </td>
    <td> <input type="text" name="City" size="20"> </td>
<br><td> Phone: </td>
    <td> <input type="text" name="Phone" size="12"> </td>
<br>
<br>

<table>
<tr>
    <td> <input type="button" value="Insert" onclick="ClickInsert();"> </td>
    <td> <input type="button" value="Next" onclick="ClickNext();"> </td>
    <td> <input type="button" value="Prev" onclick="ClickPrev();"> </td>
</tr>
<tr>
    <td colspan=3>
    <input type="button" value="Synchronize" onclick="ClickSync();">
```

```
    </td>
</tr>
</table>
</form>

<a href="AG_DEVICEOS/ul_deps.htm"></a>
</body>
</html>
```

Following is a complete listing of *tutorial.js* for your reference and use:

```
// UltraLite Tutorial

var DB_mgr;
var Connection;
var CustomerTable;

var GivenName = "";
var Surname = "";
var Street = "";
var City = "";
var Phone = "";
var ID = "";

function Connect()
{
    var     dir;
    var     open_parms;
    var     browser = navigator.platform;

    DB_mgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.Tutorial" );
    if( DB_mgr == null ) {
     alert( "Error: cannot create database manager: " + DB_mgr.sqlCode );
 return;
    }
    dir = DB_mgr.directory;
    open_parms = "con=tutorial;" + "file_name=" + dir + "\\tutorial.udb";
    }
    try {
 Connection = DB_mgr.reOpenConnection( "tutorial" );
     if( Connection == null ) {
     Connection = DB_mgr.openConnection( open_parms );
 }
    } catch( ex ) {
     if( DB_mgr.sqlCode !=
DB_mgr.SQLError.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
         alert( "Error: cannot connect to database: " + ex.getMessage() );
     return;
     }
    }
    try {
 CustomerTable = Connection.getTable( "Customer", null );
 if( CustomerTable != null ) {
     CustomerTable.open();
 }
    } catch( ex ) {
 alert( "Error: cannot open table: " + ex.getMessage() );
    }
}

function Fetch()
{
    if( CustomerTable.getRowCount() == 0 ) {
```



```
 GivenName = "";
 Surname = "";
 Street = "";
 City = "";
 Phone = "";
 ID = "";
 return;
    }
    ID = CustomerTable.getString( CustomerTable.schema.getColumnID( "ID" ) );
    GivenName =
CustomerTable.getString( CustomerTable.schema.getColumnID( "GivenName" ) );
    Surname =
CustomerTable.getString( CustomerTable.schema.getColumnID( "Surname" ) );
    Street =
CustomerTable.getString( CustomerTable.schema.getColumnID( "Street" ) );
    if( CustomerTable.isNull( CustomerTable.schema.getColumnID( "City" ) ) )
{
 City = "";
    } else {
 City =
CustomerTable.getString( CustomerTable.schema.getColumnID( "City" ) );
    }
    if( CustomerTable.isNull( CustomerTable.schema.getColumnID( "Phone" ) ) )
{
 Phone = "";
    } else {
 Phone =
CustomerTable.getString( CustomerTable.schema.getColumnID( "Phone" ) );
    }
}

function Insert()
{
    try {
 CustomerTable.insertBegin();
 CustomerTable.setString( CustomerTable.schema.getColumnID( "GivenName" ),
GivenName );
 CustomerTable.setString( CustomerTable.schema.getColumnID( "Surname" ),
Surname );
 CustomerTable.setString( CustomerTable.schema.getColumnID( "Street" ),
Street );
 if( City.length > 0 ) {
     CustomerTable.setString( CustomerTable.schema.getColumnID( "City" ),
City );
 }
 if( Phone.length > 0 ) {
     CustomerTable.setString( CustomerTable.schema.getColumnID( "Phone" ),
Phone );
 }
 CustomerTable.insert();
 CustomerTable.moveLast();
    } catch( ex ) {
 alert( "Error: cannot insert row: " + ex.getMessage() );
    }
}

function Next()
{
    if( ! CustomerTable.moveNext() ) {
 CustomerTable.moveLast();
    }
}

function Prev()
```

```
{
    if( ! CustomerTable.movePrevious() ) {
 CustomerTable.moveFirst();
    }
}

function Synchronize()
{
    var sync_parms;

    sync_parms = Connection.syncParms;
    sync_parms.setUserName( "tutorial" );
    sync_parms.setPassword( "tutorial" );
    sync_parms.setVersion( "tutorial" );
    sync_parms.setStream( sync_parms.STREAM_TYPE_TCPIP );
    try {
 Connection.synchronize();
    } catch( ex ) {
     alert( "Error: cannot synchronize: " + ex.getMessage() );
    }
}
```


UltraLite for M-Business Anywhere API reference

Data types in UltraLite for M-Business Anywhere

JavaScript has only one numeric data type and only one Date data type.

The prototypes in this API Reference include a variety of other data types in the method and property descriptions. These types are internal M-Business Anywhere data types. Distinct numeric data types such as UInt32 (unsigned 32-bit integer) are reported here to give an idea of the size and precision of data that may be supplied. Distinct data types related to date and time (Date, Time, Timestamp) are reported so that you can write code to extract the required information from the supplied data if necessary.

AuthStatusCode class

Enumerates the status codes that may be reported during MobiLink user authentication.

Syntax

```
public class AuthStatusCode
```

Members

All members of AuthStatusCode class, including all inherited members.

Name	Description
“toString method”	Generates the string name of the authorization status code constant.
“EXPIRED variable”	User ID or password has expired; authorization failed.
“IN_USE variable”	User ID is already in use; authorization failed.
“INVALID variable”	Bad user ID or password; authorization failed.
“UNKNOWN variable”	Authorization status is unknown, possibly because the connection has not yet performed a synchronization.
“VALID variable”	User ID and password were valid at time of synchronization.
“VALID_BUT_EXPIRES_SOON variable”	User ID and password are valid at the time of synchronization but expire soon.

Remarks

This object can be obtained from DatabaseManager object as follows:

```
var authStatus = dbMgr.AuthStatusCode;
```

toString method

Generates the string name of the authorization status code constant.

Syntax

```
public String toString(UInt16 code)
```

Parameters

- **code** The authorization code returned by SyncResult.

Returns

The name of the code or *Given Code Unknown* if not a recognized code.

EXPIRED variable

User ID or password has expired; authorization failed.

Syntax

```
public const UInt16 EXPIRED;
```

Remarks

EXPIRED = UL_AUTH_STATUS_EXPIRED.

IN_USE variable

User ID is already in use; authorization failed.

Syntax

```
public const UInt16 IN_USE;
```

Remarks

IN_USE = UL_AUTH_STATUS_IN_USE.

INVALID variable

Bad user ID or password; authorization failed.

Syntax

```
public const UInt16 INVALID;
```

Remarks

INVALID = UL_AUTH_STATUS_INVALID.

UNKNOWN variable

Authorization status is unknown, possibly because the connection has not yet performed a synchronization.

Syntax

```
public const UInt16 UNKNOWN;
```

Remarks

UNKNOWN = UL_AUTH_STATUS_UNKNOWN.

VALID variable

User ID and password were valid at time of synchronization.

Syntax

```
public const UInt16 VALID;
```

Remarks

VALID = UL_AUTH_STATUS_VALID.

VALID_BUT_EXPIRES_SOON variable

User ID and password are valid at the time of synchronization but expire soon.

Syntax

```
public const UInt16 VALID_BUT_EXPIRES_SOON;
```

Remarks

VALID_BUT_EXPIRES_SOON = UL_AUTH_STATUS_VALID_BUT_EXPIRES_SOON.

Connection class

Represents a connection to an UltraLite database.

Syntax

```
public class Connection
```

Members

All members of Connection class, including all inherited members.

Name	Description
"cancelGetNotification method"	Cancels any pending get-notification calls on all queues matching the given name.
"changeEncryptionKey method"	Changes the database encryption key to the new specified key.
"close method"	Closes this UltraLite database connection.
"commit method"	Commits outstanding changes to the database.
"countUploadRow method"	Returns the number of rows that need to be uploaded.
"createNotificationQueue method"	Creates an event notification queue for this connection.
"declareEvent method"	Declares an event which can then be registered for and triggered.
"destroyNotificationQueue method"	Destroys the given event notification queue.
"getDatabaseID method"	Returns the database ID value used for global autoincrement columns.
"getGlobalAutoIncrementUsage method"	Returns the percentage of available global autoincrement values that have been used.
"getLastDownloadTime method"	Returns the time of the last download.
"getLastIdentity method"	Returns the @identity value.
"getNotification method"	Reads an event notification.
"getNotificationParameter method"	Gets a parameter for the event notification just read by GetNotification().
"getTable method"	Creates and returns a reference to the requested table in the database.
"getTableAGDBSet method"	Binds to an AGDBSet object using the specified table name.
"grantConnectTo method"	Creates a new user or changes the password of an existing one.

Name	Description
“isOpen method”	Checks whether this connection is currently open.
“prepareStatement method”	Prepares a SQL statement.
“registerForEvent method”	Registers a queue to receive notifications of an event.
“resetLastDownloadTime method”	Resets the time of the last download of the named publication.
“revokeConnectFrom method”	Deletes an existing user.
“rollback method”	Rolls back outstanding changes to the database.
“rollbackPartialDownload method”	Rolls back a partial download.
“sendNotification method”	Send a notification to all queues matching the given name.
“setDatabaseID method”	Sets the database ID value to be used for global autoincrement columns.
“startSynchronizationDelete method”	Marks for synchronization all subsequent deletes made by this connection.
“stopSynchronizationDelete method”	Prevents delete operations from being synchronized.
“synchronize method”	Synchronizes the database.
“synchronizeWithParm method”	Synchronizes the database using the specified SyncParms object.
“triggerEvent method”	Triggers a user-defined event and sends a notification to all registered queues.
“validateDatabase method”	Validates the database on this connection.
“autoCommit variable”	Controls whether a commit is performed after each statement (insert, update or delete).
“databaseSchema variable”	Readonly property; Holds the database schema.
“openParms variable”	Readonly property; Returns the parameter string used to open this connection.
“skipMBASync variable”	Controls whether the data synchronization with MobiLink is integrated into the M-Busniess Anywhere synchronization process.

Name	Description
"sqlCode variable"	Readonly property; Provides error checking capabilities by returning the SQL code value for the success or failure of a database operation.
"syncParms variable"	Readonly property; Holds the synchronization parameter object of the current connection.
"syncResult variable"	Readonly property; Holds the synchronization result object of the current connection.

Remarks

Connections are instantiated and opened using DatabaseManager.openConnection.

You must open a connection before carrying out any other operation, and you must close the connection after you have finished all operations on the connection and before your application terminates. You must close all tables opened on a connection before closing the connection.

When a JavaScript Error is thrown because of a failed UltraLite database operation, the SQL error code is set on the sqlCode field of the Connection object.

cancelGetNotification method

Cancels any pending get-notification calls on all queues matching the given name.

Syntax

```
public UInt32 cancelGetNotification(String queueName)
```

Parameters

- **queueName** The name of an event notification queue.

Returns

The number of affected queues.

Remarks

For more information, see "Working with event notifications" [*UltraLite - Database Management and Reference*].

See also

- “createNotificationQueue method” on page 46
- “declareEvent method” on page 47
- “destroyNotificationQueue method” on page 47
- “getNotification method” on page 49
- “registerForEvent method” on page 52
- “sendNotification method” on page 54
- “triggerEvent method” on page 56

changeEncryptionKey method

Changes the database encryption key to the new specified key.

Syntax

```
public void changeEncryptionKey(String newKey)
```

Parameters

- **newKey** The new encryption key for the database.

Remarks

It is not possible to open the database if the encryption key is lost.

close method

Closes this UltraLite database connection.

Syntax

```
public void close()
```

Remarks

Once a connection is closed, it can not be reopened. To reopen a connection, a new connection object must be created and opened.

It is an error to use any object, such as a table or a schema, associated with a closed connection.

The closed connection object is not set to NULL automatically after it is closed. It is recommended that you explicitly set the connection object to NULL after closing the connection.

commit method

Commits outstanding changes to the database.

Syntax

```
public void commit()
```

countUploadRow method

Returns the number of rows that need to be uploaded.

Syntax

```
public UInt32 countUploadRow(String pub_list, UInt32 threshold)
```

Parameters

- **pub_list** A comma-separated list of publications to consider.
- **threshold** The limit on the number of rows to count.

Returns

The number of rows that need to be uploaded.

createNotificationQueue method

Creates an event notification queue for this connection.

Syntax

```
public void createNotificationQueue(String queueName, String parms)
```

Parameters

- **queueName** The name of an event notification queue.
- **parms** The creation parameter. Currently unused and must be set to NULL.

Remarks

For more information, see "Working with event notifications" [*UltraLite - Database Management and Reference*].

See also

- "cancelGetNotification method" on page 44
- "declareEvent method" on page 47
- "destroyNotificationQueue method" on page 47
- "getNotification method" on page 49
- "registerForEvent method" on page 52
- "sendNotification method" on page 54
- "triggerEvent method" on page 56

declareEvent method

Declares an event which can then be registered for and triggered.

Syntax

```
public void declareEvent(String eventName)
```

Parameters

- **eventName** The name of an event.

See also

- “createNotificationQueue method” on page 46
- “cancelGetNotification method” on page 44
- “destroyNotificationQueue method” on page 47
- “getNotification method” on page 49
- “registerForEvent method” on page 52
- “sendNotification method” on page 54
- “triggerEvent method” on page 56

destroyNotificationQueue method

Destroys the given event notification queue.

Syntax

```
public void destroyNotificationQueue(String queueName)
```

Parameters

- **queueName** The name of an existing event notification queue.

Remarks

For more information, see “Working with event notifications” [*UltraLite - Database Management and Reference*].

See also

- “createNotificationQueue method” on page 46
- “cancelGetNotification method” on page 44
- “declareEvent method” on page 47
- “getNotification method” on page 49
- “registerForEvent method” on page 52
- “sendNotification method” on page 54
- “triggerEvent method” on page 56

getDatabaseID method

Returns the database ID value used for global autoincrement columns.

Syntax

```
public UInt32 getDatabaseID()
```

Returns

The Database ID value specified by setDatabaseID. INVALID_DATABASE_ID is returned if a Database ID is not set.

See also

- "setDatabaseID method" on page 54

getGlobalAutoIncrementUsage method

Returns the percentage of available global autoincrement values that have been used.

Syntax

```
public UInt16 getGlobalAutoIncrementUsage()
```

Returns

The percentage of available global increment values used so far.

Remarks

Your application should set a new value for the global database ID using the setDatabaseID method if the percentage approaches 100.

See also

- "setDatabaseID method" on page 54

getLastDownloadTime method

Returns the time of the last download.

Syntax

```
public Date getLastDownloadTime(String pub_list)
```

Parameters

- **pub_list** A comma-separated list of publication names to check.

Returns

The timestamp of the last download.

See also

- "PublicationSchema class" on page 91

getLastIdentity method

Returns the @identity value.

Syntax

```
public UInt64 getLastIdentity()
```

Returns

The @identity value.

getNotification method

Reads an event notification.

Syntax

```
public String getNotification(String queueName, UInt32 wait_ms)
```

Parameters

- **queueName** The queue to read or NULL for default connection queue.
- **wait_ms** The time to wait before returning. Measured in milliseconds. Pass UL_READ_WAIT_INFINITE for wait_ms to wait indefinitely.

Returns

The name of the event.

Remarks

For more information, see "Working with event notifications" [*UltraLite - Database Management and Reference*].

See also

- “createNotificationQueue method” on page 46
- “cancelGetNotification method” on page 44
- “declareEvent method” on page 47
- “destroyNotificationQueue method” on page 47
- “registerForEvent method” on page 52
- “sendNotification method” on page 54
- “triggerEvent method” on page 56

getNotificationParameter method

Gets a parameter for the event notification just read by GetNotification().

Syntax

```
public String getNotificationParameter(
    String queueName,
    String parameterName
)
```

Parameters

- **queueName** The queue name matching the GetNotification() call.
- **parameterName** The name of the parameter to read (or "*").

Returns

The parameter value.

Remarks

For more information, see “Working with event notifications” [*UltraLite - Database Management and Reference*].

See also

- “createNotificationQueue method” on page 46
- “cancelGetNotification method” on page 44
- “declareEvent method” on page 47
- “destroyNotificationQueue method” on page 47
- “getNotification method” on page 49
- “registerForEvent method” on page 52
- “sendNotification method” on page 54
- “triggerEvent method” on page 56

getTable method

Creates and returns a reference to the requested table in the database.

Syntax

```
public Table getTable(String tableName, String persistName)
```

Parameters

- **tableName** The name of the table to fetch.
- **persistName** The name for cross-page JavaScript object persistence. Set to NULL if no persistence is required (for example, if the application has only a single HTML page).

Returns

An instance of the ULPod Table object.

getTableAGDBSet method

Binds to an AGDBSet object using the specified table name.

Syntax

```
public TableAGDBSet getTableAGDBSet(String schemaName)
```

Parameters

- **schemaName** The table to be binded to the AGDBSet.

Returns

An instance of the TableAGDBSet object.

grantConnectTo method

Creates a new user or changes the password of an existing one.

Syntax

```
public void grantConnectTo(String uid, String pwd)
```

Parameters

- **uid** The name of the user to create or change the password for.
- **pwd** The password for this user.

isOpen method

Checks whether this connection is currently open.

Syntax

```
public Boolean isOpen()
```

Returns

True if the connection is open, false otherwise.

prepareStatement method

Prepares a SQL statement.

Syntax

```
public PreparedStatement prepareStatement(
    String sql,
    String persistName
)
```

Parameters

- **sql** The SQL statement to prepare.
- **persistName** The persistent name for the returned PreparedStatement object.

Returns

The prepared statement object on success. Otherwise, NULL.

registerForEvent method

Registers a queue to receive notifications of an event.

Syntax

```
public void registerForEvent(
    String eventName,
    String objectName,
    String queueName,
    Boolean register_not_unreg
)
```

Parameters

- **eventName** The name of an event.
- **objectName** The object to which event applies, such as the table name.
- **queueName** The name of an event notification queue.
- **register_not_unreg** Set to true to register, or false to unregister.

Remarks

For more information, see "Working with event notifications" [*UltraLite - Database Management and Reference*].

See also

- "createNotificationQueue method" on page 46
- "cancelGetNotification method" on page 44
- "declareEvent method" on page 47
- "destroyNotificationQueue method" on page 47
- "getNotification method" on page 49
- "sendNotification method" on page 54
- "triggerEvent method" on page 56

resetLastDownloadTime method

Resets the time of the last download of the named publication.

Syntax

```
public void resetLastDownloadTime(String pub_list)
```

Parameters

- **pub_list** A comma-separated list of publications to reset.

revokeConnectFrom method

Deletes an existing user.

Syntax

```
public void revokeConnectFrom(String uid)
```

Parameters

- **uid** The name of the user to delete.

rollback method

Rolls back outstanding changes to the database.

Syntax

```
public void rollback()
```

rollbackPartialDownload method

Rolls back a partial download.

Syntax

```
public void rollbackPartialDownload()
```

sendNotification method

Send a notification to all queues matching the given name.

Syntax

```
public UInt32 sendNotification(
    String queueName,
    String eventName,
    String parms
)
```

Parameters

- **queueName** The target queue name (or "*").
- **eventName** The identity for notification.
- **parms** Optional parameters option list. This argument specifies a semicolon delimited name=value pairs option list.

Returns

The number of notifications sent (the number of matching queues).

Remarks

For more information, see "Working with event notifications" [*UltraLite - Database Management and Reference*].

See also

- "createNotificationQueue method" on page 46
- "cancelGetNotification method" on page 44
- "declareEvent method" on page 47
- "destroyNotificationQueue method" on page 47
- "getNotification method" on page 49
- "registerForEvent method" on page 52
- "triggerEvent method" on page 56

setDatabaseID method

Sets the database ID value to be used for global autoincrement columns.

Syntax

```
public void setDatabaseID(UInt32 value)
```

Parameters

- **value** The database ID value. This value must be in the range of [0,0x0fffffff].

startSynchronizationDelete method

Marks for synchronization all subsequent deletes made by this connection.

Syntax

```
public void startSynchronizationDelete()
```

Remarks

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

stopSynchronizationDelete method

Prevents delete operations from being synchronized.

Syntax

```
public void stopSynchronizationDelete()
```

Remarks

This method is useful for deleting old information from an UltraLite database to save space, while not deleting this information on the consolidated database.

synchronize method

Synchronizes the database.

Syntax

```
public void synchronize()
```

synchronizeWithParm method

Synchronizes the database using the specified SyncParms object.

Syntax

```
public void synchronizeWithParm(SyncParms parms)
```

Parameters

- **parms** The provided synchronization parameter object.

triggerEvent method

Triggers a user-defined event and sends a notification to all registered queues.

Syntax

```
public UInt32 triggerEvent(String eventName, String parms)
```

Parameters

- **eventName** The name of the system or user-defined event to trigger.
- **parms** Optional parameters option list. This argument specifies a semicolon delimited name=value pairs option list.

Returns

The number of event notifications sent.

Remarks

For more information, see "Working with event notifications" [*UltraLite - Database Management and Reference*].

See also

- "createNotificationQueue method" on page 46
- "cancelGetNotification method" on page 44
- "declareEvent method" on page 47
- "destroyNotificationQueue method" on page 47
- "registerForEvent method" on page 52
- "sendNotification method" on page 54
- "getNotification method" on page 49

validateDatabase method

Validates the database on this connection.

Syntax

```
public void validateDatabase(UInt16 type, String tableName)
```


Parameters

- **type** The type of validation to perform.
- **tableName** A specific table name to validate. Specify null to validate the entire database.

autoCommit variable

Controls whether a commit is performed after each statement (insert, update or delete).

Syntax

```
public Boolean autoCommit;
```

Remarks

If autoCommit is false, a commit or rollback is performed only when the user invokes the commit() or rollback() method.

By default, a database commit is performed after each successful statement. If the commit fails, you have the option to execute additional SQL statements and perform the commit again, or execute a rollback statement.

databaseSchema variable

Readonly property; Holds the database schema.

Syntax

```
public DatabaseSchema databaseSchema;
```

Remarks

This property is only valid while its connection is open.

openParms variable

Readonly property; Returns the parameter string used to open this connection.

Syntax

```
public String openParms;
```

Remarks

The open parameters string is a semicolon-separated list of name=value pairs.

skipMBASync variable

Controls whether the data synchronization with MobiLink is integrated into the M-Busniess Anywhere synchronization process.

Syntax

```
public Boolean skipMBASync;
```

Remarks

If skipMBASync is false, user has to manually initiate the data synchronization with MobiLink by calling Connection.Synchronize().

By default, UltraLite database is synchronized with MobiLink as an integrated part of M-Business Anywhere synchronization on Pocket PC.

sqlCode variable

Readonly property; Provides error checking capabilities by returning the SQL code value for the success or failure of a database operation.

Syntax

```
public Int32 sqlCode;
```

syncParms variable

Readonly property; Holds the synchronization parameter object of the current connection.

Syntax

```
public SyncParms syncParms;
```

syncResult variable

Readonly property; Holds the synchronization result object of the current connection.

Syntax

```
public SyncResult syncResult;
```

ConnectionParms class

Specifies parameters for opening a connection to an UltraLite database.

Syntax

```
public class ConnectionParms
```

Members

All members of ConnectionParms class, including all inherited members.

Name	Description
“toString method”	Returns the constructed connection string.
“additionalParms variable”	Specifies additional parameters as a semicolon-separated list of name=value pairs.
“cacheSize variable”	Specifies the size of the cache.
“connectionName variable”	Specifies a name for the connection.
“databaseOnCE variable”	Specifies the path and filename of the UltraLite database on Windows Mobile.
“databaseOnDesktop variable”	Specifies the path and filename of the UltraLite database on Windows desktop platforms.
“encryptionKey variable”	Specifies a key for encrypting the database.
“password variable”	Specifies the password for the user.
“userID variable”	Specifies an authenticated user for the database.

Remarks

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

Use Connection.revokeConnectFrom to disable the default user. Use Connection.grantConnectTo to add a user or change a user's password,

Currently, only one connection can be opened at any time.

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

toString method

Returns the constructed connection string.

Syntax

```
public String toString()
```

additionalParms variable

Specifies additional parameters as a semicolon-separated list of name=value pairs.

Syntax

```
public String additionalParms;
```

cacheSize variable

Specifies the size of the cache.

Syntax

```
public String cacheSize;
```

Remarks

The values for the cache size are specified in units of bytes. Use the suffix k or K to indicate units of kilobytes and use the suffix m or M to indicate megabytes.

The default cache size is used if the cache size is unspecified or improperly specified.

connectionName variable

Specifies a name for the connection.

Syntax

```
public String connectionName;
```

Remarks

This is required if you have multiple HTML pages in an application. It is used to persist connection objects across different HTML pages.

databaseOnCE variable

Specifies the path and filename of the UltraLite database on Windows Mobile.

Syntax

```
public String databaseOnCE;
```


Remarks

You must escape any backslash characters in paths.

databaseOnDesktop variable

Specifies the path and filename of the UltraLite database on Windows desktop platforms.

Syntax

```
public String databaseOnDesktop;
```

Remarks

You must escape any backslash characters in paths.

encryptionKey variable

Specifies a key for encrypting the database.

Syntax

```
public String encryptionKey;
```

password variable

Specifies the password for the user.

Syntax

```
public String password;
```

userID variable

Specifies an authenticated user for the database.

Syntax

```
public String userID;
```

CreationParms class

Defines parameters that may be specified when creating an UltraLite database.

Syntax

```
public class CreationParms
```

Members

All members of CreationParms class, including all inherited members.

Name	Description
“toString method”	Returns the constructed creation string.
“caseSensitive variable”	Sets the case-sensitivity of string comparisons in the UltraLite database.
“checksumLevel variable”	Sets the level of checksum validation in the database; the default value is 0.
“dateFormat variable”	Sets the default string format in which dates are retrieved from the database.
“dateOrder variable”	Controls the interpretation of date ordering of months, days, and years.
“fips variable”	Specifies whether the database is encrypted using FIPS strong encryption.
“maxHashSize variable”	Sets the maximum number of bytes that are used to hash the UltraLite indexes; the default value is 0.
“nearestCentury variable”	Controls the interpretation of two-digit years in String-to-Date conversions.
“obfuscate variable”	Controls whether data in the database is obfuscated.
“pageSize variable”	Defines the database page size; the default value is 4096.
“precision variable”	Specifies the maximum number of digits in the result of any decimal arithmetic.
“scale variable”	Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.
“timeFormat variable”	Sets the format for times retrieved from the database.
“timestampFormat variable”	Determines how the timestamp is formatted in UltraLite.
“timestampIncrement variable”	Determines how the timestamp is truncated in UltraLite.

Name	Description
"timestampWithTimeZoneFormat variable"	Sets the format for the timestamp with the time zone retrieved from the database.
"utf8Encoding variable"	Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode.

Remarks

Some UltraLite database options must be set at the time the database is created.

See also

- "createDatabase method" on page 68

toString method

Returns the constructed creation string.

Syntax

```
public String toString()
```

caseSensitive variable

Sets the case-sensitivity of string comparisons in the UltraLite database.

Syntax

```
public Boolean caseSensitive;
```

checksumLevel variable

Sets the level of checksum validation in the database; the default value is 0.

Syntax

```
public UInt32 checksumLevel;
```

dateFormat variable

Sets the default string format in which dates are retrieved from the database.

Syntax

```
public String dateFormat;
```

dateOrder variable

Controls the interpretation of date ordering of months, days, and years.

Syntax

```
public String dateOrder;
```

fips variable

Specifies whether the database is encrypted using FIPS strong encryption.

Syntax

```
public Boolean fips;
```

maxHashSize variable

Sets the maximum number of bytes that are used to hash the UltraLite indexes; the default value is 0.

Syntax

```
public UInt32 maxHashSize;
```

nearestCentury variable

Controls the interpretation of two-digit years in String-to-Date conversions.

Syntax

```
public UInt32 nearestCentury;
```

obfuscate variable

Controls whether data in the database is obfuscated.

Syntax

```
public Boolean obfuscate;
```


pageSize variable

Defines the database page size; the default value is 4096.

Syntax

```
public UInt32 pageSize;
```

Remarks

Supported values are 1024, 2048, 4096, 8192, and 16384.

precision variable

Specifies the maximum number of digits in the result of any decimal arithmetic.

Syntax

```
public UInt32 precision;
```

scale variable

Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

Syntax

```
public UInt32 scale;
```

timeFormat variable

Sets the format for times retrieved from the database.

Syntax

```
public String timeFormat;
```

timestampFormat variable

Determines how the timestamp is formatted in UltraLite.

Syntax

```
public String timestampFormat;
```

timestampIncrement variable

Determines how the timestamp is truncated in UltraLite.

Syntax

```
public UInt32 timestampIncrement;
```

timestampWithTimeZoneFormat variable

Sets the format for the timestamp with the time zone retrieved from the database.

Syntax

```
public String timestampWithTimeZoneFormat;
```

utf8Encoding variable

Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode.

Syntax

```
public Boolean utf8Encoding;
```

DatabaseManager class

Manages connections and UltraLite databases.

Syntax

```
public class DatabaseManager
```

Members

All members of DatabaseManager class, including all inherited members.

Name	Description
“createConnectionParms method”	Creates and returns an instance of the ConnectionParms object.
“createCreationParms method”	Creates and returns an instance of the CreationParms object.
“createDatabase method”	Creates a new database.
“dropDatabase method”	Erases an existing database that is not currently running.

Name	Description
“openConnection method”	Opens a new connection to an existing database.
“reOpenConnection method”	Returns the persistent Connection object previously created or opened.
“validateDatabase method”	Performs low level and index validataion on a database.
“AuthStatusCode variable”	Readonly property; Holds the AuthStatusCode object.
“directory variable”	Readonly property; Returns the application directory for AvantGo.
“runtimeType variable”	Readonly property; Returns the runtime type: standalone or engine/client.
“sqlCode variable”	Readonly property; Provides error checking capabilities by returning the SQL code value for the success or failure of a database operation.
“SQLError variable”	Readonly property; Holds the SQLError object.
“SQLType variable”	Readonly property; Holds the SQLType object.
“UL_ENGINE_CLIENT variable”	Engine version of UltraLite runtime.
“UL_STANDALONE variable”	Standalone version of UltraLite runtime.
“VALIDATE_EXPRESS variable”	Validation input type: do a faster, though less thorough, validation.
“VALIDATE_FULL variable”	Validation input type: validate tables, indexes, and database.

Remarks

You must open a connection before carrying out any other operation, and you must close the connection after you have finished all operations on the connection and before your application terminates. You must close all tables opened on a connection before closing the connection.

The DatabaseManager object is the only main PODS object. It is created in the following way:

```
var dbMgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.CustDB" );
```

Note
"iAnywhere.UltraLite.DatabaseManager" is the standard URL prefix to create a DatabaseManager object. The last portion in the URL is the specific applicaiton name. Each ULPod application should has its own unique application name.

createConnectionParms method

Creates and returns an instance of the ConnectionParms object.

Syntax

```
public ConnectionParms createConnectionParms()
```

createCreationParms method

Creates and returns an instance of the CreationParms object.

Syntax

```
public CreationParms createCreationParms()
```

createDatabase method

Creates a new database.

Syntax

```
public void createDatabase(String conn_parms, String create_parms)
```

Parameters

- **conn_parms** The connection parameters used to connect to the new database.
- **create_parms** The creation parameters used to create the database.

dropDatabase method

Erases an existing database that is not currently running.

Syntax

```
public void dropDatabase(String parms)
```

Parameters

- **parms** The database identification parameters.

openConnection method

Opens a new connection to an existing database.

Syntax

```
public Connection openConnection(String parms)
```


Parameters

- **parms** The connection string.

Returns

A new connection object is returned on success. Otherwise, NULL is returned.

reOpenConnection method

Returns the persistent Connection object previously created or opened.

Syntax

```
public Connection reOpenConnection(String connName)
```

Parameters

- **connName** The connection string.

Returns

The opened Connection object.

Remarks

This method is used to maintain connections across multiple web pages.

validateDatabase method

Performs low level and index validataion on a database.

Syntax

```
public void validateDatabase(String start_parms, UInt16 type)
```

Parameters

- **start_parms** The parameters for starting the database
- **type** The type of validation. See VALIDATE_EXPRESS and VALIDATE_FULL properties in the DatabaseManager class.

AuthStatusCode variable

Readonly property; Holds the AuthStatusCode object.

Syntax

```
public AuthStatusCode AuthStatusCode;
```

directory variable

Readonly property; Returns the application directory for AvantGo.

Syntax

```
public String directory;
```

Returns

The path of the AvantGo application directory.

Remarks

This is where the downloaded schema file is placed by AvantGo.

runtimeType variable

Readonly property; Returns the runtime type: standalone or engine/client.

Syntax

```
public UInt32 runtimeType;
```

sqlCode variable

Readonly property; Provides error checking capabilities by returning the SQL code value for the success or failure of a database operation.

Syntax

```
public Int32 sqlCode;
```

SQLError variable

Readonly property; Holds the SQLError object.

Syntax

```
public SQLError SQLError;
```

SQLType variable

Readonly property; Holds the SQLType object.

Syntax

```
public SQLType SQLType;
```

UL_ENGINE_CLIENT variable

Engine version of UltraLite runtime.

Syntax

```
public const UInt32 UL_ENGINE_CLIENT;
```

UL_STANDALONE variable

Standalone version of UltraLite runtime.

Syntax

```
public const UInt32 UL_STANDALONE;
```

VALIDATE_EXPRESS variable

Validation input type: do a faster, though less thorough, validation.

Syntax

```
public const UInt16 VALIDATE_EXPRESS;
```

VALIDATE_FULL variable

Validation input type: validate tables, indexes, and database.

Syntax

```
public const UInt16 VALIDATE_FULL;
```

DatabaseSchema class

Represents the schema of an UltraLite database.

Syntax

```
public class DatabaseSchema
```

Members

All members of DatabaseSchema class, including all inherited members.

Name	Description
“getCollationName method”	Returns the name of the database's collation sequence.
“getDatabaseProperty method”	Returns the value of the specified database property.
“getDateFormat method”	Returns the date format used for string conversions.
“getDateOrder method”	Returns the date order used for string conversions.
“getNearestCentury method”	Returns the nearest century used for string conversions.
“getPrecision method”	Returns the floating-point precision used for string conversions.
“getPublicationCount method”	Returns the number of publications in the database.
“getPublicationName method”	Returns the name of the publication identified by the specified publication ID.
“getPublicationSchema method”	Returns the publication schema corresponding to the named publication.
“getTableCount method”	Returns the number of tables in the database.
“getTableName method”	Returns the name of the table identified by the specified table ID.
“getTimeFormat method”	Returns the time format used for string conversions.
“getTimestampFormat method”	Returns the timestamp format used for string conversions.
“isCaseSensitive method”	Checks whether the database is case sensitive.
“isOpen method”	Checks whether the publication schema is currently open.
“setDatabaseOption method”	Sets the value for the specified database option.

Remarks

A DatabaseSchema object is attached to a connection and is only valid while that connection is open.

getCollationName method

Returns the name of the database's collation sequence.

Syntax

```
public String getCollationName()
```

Returns

The name of the database's collation sequence.

getDatabaseProperty method

Returns the value of the specified database property.

Syntax

```
public String getDatabaseProperty(String name)
```

Parameters

- **name** The name of the database property.

Returns

The value of the property.

Remarks

For more information on recognized database properties, see "UltraLite database properties" [*UltraLite - Database Management and Reference*].

getDateFormat method

Returns the date format used for string conversions.

Syntax

```
public String getDateFormat()
```

Returns

A string describing the date format.

getDateOrder method

Returns the date order used for string conversions.

Syntax

```
public String getDateOrder()
```

Returns

A string describing the date order.

getNearestCentury method

Returns the nearest century used for string conversions.

Syntax

```
public String getNearestCentury()
```

Returns

A string describing the nearest century.

getPrecision method

Returns the floating-point precision used for string conversions.

Syntax

```
public String getPrecision()
```

Returns

A string describing the floating point precision.

getPublicationCount method

Returns the number of publications in the database.

Syntax

```
public UInt16 getPublicationCount()
```

Returns

The number of publications.

getPublicationName method

Returns the name of the publication identified by the specified publication ID.

Syntax

```
public String getPublicationName(UInt16 pubID)
```


Parameters

- **pubID** The 1-based publication ID.

Returns

The publication name.

getPublicationSchema method

Returns the publication schema corresponding to the named publication.

Syntax

```
public PublicationSchema getPublicationSchema(String name)
```

Parameters

- **name** The name of the publication.

Returns

The publication schema of the named publication.

getTableCount method

Returns the number of tables in the database.

Syntax

```
public UInt16 getTableCount()
```

Returns

The number of tables.

getTableName method

Returns the name of the table identified by the specified table ID.

Syntax

```
public String getTableName(UInt16 tableID)
```

Parameters

- **tableID** The 1-based table ID.

Returns

The table name.

getTimeFormat method

Returns the time format used for string conversions.

Syntax

```
public String getTimeFormat()
```

Returns

A string describing the time format.

getTimestampFormat method

Returns the timestamp format used for string conversions.

Syntax

```
public String getTimestampFormat()
```

Returns

A string describing the timestamp format.

isCaseSensitive method

Checks whether the database is case sensitive.

Syntax

```
public Boolean isCaseSensitive()
```

Returns

True if the database is case sensitive, otherwise false.

isOpen method

Checks whether the publication schema is currently open.

Syntax

```
public Boolean isOpen()
```

Returns

True if the publication schema is open, false otherwise.

setDatabaseOption method

Sets the value for the specified database option.

Syntax

```
public Boolean setDatabaseOption(String name, String value)
```

Parameters

- **name** The database option name.
- **value** The option value.

Returns

True on success, otherwise false.

Remarks

For more information on recognized database properties, see "UltraLite database properties" [*UltraLite - Database Management and Reference*].

IndexSchema class

Represents the schema of an UltraLite table index.

Syntax

```
public class IndexSchema
```

Members

All members of IndexSchema class, including all inherited members.

Name	Description
"getColumnCount method"	Returns the number of columns in the index.
"getColumnName method"	Gets the name of a column given its 1-based ID.
"getName method"	Returns the name of this index.
"getReferencedIndexName method"	Returns the name of the referenced primary index if this index is a foreign key.
"getReferencedTableName method"	Returns the name of the referenced primary table if this index is a foreign key.

Name	Description
“isColumnDescending method”	Checks whether the named column is used in descending order by this index.
“isForeignKey method”	Checks whether the index is a foreign key.
“isForeignKeyCheckOnCommit method”	Checks whether referential integrity for this foreign key is performed on commits or on inserts and updates.
“isForeignKeyNullable method”	Checks whether this foreign key is nullable.
“isPrimaryKey method”	Checks whether the index is the primary key.
“isUniqueIndex method”	Checks whether the index is unique.
“isUniqueKey method”	Checks whether the index is a unique key.

getColumnCount method

Returns the number of columns in the index.

Syntax

```
public UInt16 getColumnCount()
```

Returns

The number of columns as a short-type integer.

getColumnName method

Gets the name of a column given its 1-based ID.

Syntax

```
public String getColumnName(UInt16 colID)
```

Parameters

- **colID** The 1-based column ID.

Returns

The name of the column.

getName method

Returns the name of this index.

Syntax

```
public String getName()
```

Returns

The name of the index.

getReferencedIndexName method

Returns the name of the referenced primary index if this index is a foreign key.

Syntax

```
public String getReferencedIndexName()
```

Returns

The name of the referenced index.

getReferencedTableName method

Returns the name of the referenced primary table if this index is a foreign key.

Syntax

```
public String getReferencedTableName()
```

Returns

The name of referenced table.

isColumnDescending method

Checks whether the named column is used in descending order by this index.

Syntax

```
public Boolean isColumnDescending(String name)
```

Parameters

- **name** The column name.

Returns

True if column is used in descending order, or false otherwise.

isForeignKey method

Checks whether the index is a foreign key.

Syntax

```
public Boolean isForeignKey()
```

Returns

True if index is the foreign key, or false otherwise.

Remarks

Columns in a foreign key may reference a non-null unique index of another table.

isForeignKeyCheckOnCommit method

Checks whether referential integrity for this foreign key is performed on commits or on inserts and updates.

Syntax

```
public Boolean isForeignKeyCheckOnCommit()
```

Returns

True if referential integrity is checked on commits, or false if it is checked on inserts and updates.

isForeignKeyNullable method

Checks whether this foreign key is nullable.

Syntax

```
public Boolean isForeignKeyNullable()
```

Returns

True if this foreign key is nullable, or false othewise.

isPrimaryKey method

Checks whether the index is the primary key.

Syntax

```
public Boolean isPrimaryKey()
```


Returns

True if index is the primary key, or false if the index is not the primary key.

isUniqueIndex method

Checks whether the index is unique.

Syntax

```
public Boolean isUniqueIndex()
```

Returns

True if index is a unique, or false otherwise.

isUniqueKey method

Checks whether the index is a unique key.

Syntax

```
public Boolean isUniqueKey()
```

Returns

True if index is a unique key, or false otherwise.

PreparedStatement class

Represents a pre-compiled SQL statement with or without IN parameters.

Syntax

```
public class PreparedStatement
```

Members

All members of PreparedStatement class, including all inherited members.

Name	Description
"appendBytesParameter method"	Appends the specified subset of the specified array of bytes to the new value for the specified SQLType.LONGBINARY column.
"appendStringChunkParameter method"	Appends the String to the new value for the specified SQLType.LONGVARCHAR column.

Name	Description
“close method”	Closes the prepared statement.
“executeQuery method”	Executes a SQL SELECT statement as a query.
“executeStatement method”	Executes a statement that does not return a result set, such as a SQL INSERT, DELETE or UPDATE statement.
“getPlan method”	Returns a text-based description of the query execution plan.
“getResultSetSchema method”	Returns the schema describing the result set of this query statement.
“hasResultSet method”	Checks whether the prepared statement generated a result set.
“isOpen method”	Checks whether this prepared statement is currently open.
“setBooleanParameter method”	Sets the value for the specified parameter using a boolean.
“setBytesParameter method”	Sets the value for the specified parameter using an array of bytes.
“setDateParameter method”	Sets the value for the specified SQLType.DATE type parameter using a date.
“setDoubleParameter method”	Sets the value for the specified SQLType.DOUBLE type parameter using a double.
“setFloatParameter method”	Sets the value for the specified SQLType.REAL type parameter.
“setIntParameter method”	Sets a parameter to a signed 32-bit value.
“setLongParameter method”	Sets a parameter to a signed 64-bit value.
“setNullParameter method”	Sets the specified parameter to the SQL NULL value.
“setShortParameter method”	Sets a parameter to a signed 16-bit value.
“setStringParameter method”	Sets the value for the specified parameter using a String.

Name	Description
"setTimeParameter method"	Sets the value for the specified SQLType.TIME type parameter using a date.
"setTimestampParameter method"	Sets the value for the specified SQLType.TIMESTAMP type parameter using a date.
"setTimestampWithTimeZoneParameter method"	Sets the value for the specified SQLType.TIMESTAMP_WITH_TIME_ZONE type parameter using a date.
"setULongParameter method"	Sets a parameter to an unsigned 64-bit value.
"setUUIDParameter method"	Sets the value for the specified parameter using a UUID.

Remarks

This object can then be used to efficiently execute this statement multiple times.

When a prepared statement is closed, all ResultSet and ResultSetSchema objects associated with it are also closed. For resource management reasons, it is preferred that you explicitly close prepared statements when you are done with them.

appendBytesParameter method

Appends the specified subset of the specified array of bytes to the new value for the specified SQLType.LONGBINARY column.

Syntax

```
public void appendBytesParameter(
    UInt16 parmID,
    Array value,
    UInt32 srcOffset,
    UInt32 count
)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The byte chunk to append to the current new value for the parameter.
- **srcOffset** The start position in the source array.
- **count** The number of bytes to be copied.

Remarks

The bytes at position srcOffset, starting from 0 through (srcOffset + count - 1), of the value byte chunk are appended to the value for the specified parameter. When inserting, insertBegin initializes the new value to the parameter's default value.

An error with the SQLError.SQLE_INVALID_PARAMETER code is thrown and the destination is not modified when any of the following statements are true:

- The value parameter is null.
- The srcOffset parameter is negative.
- The count argument is negative.
- srcOffset + count is greater than value.length, the length of the source array.

appendStringChunkParameter method

Appends the String to the new value for the specified SQLType.LONGVARCHAR column.

Syntax

```
public void appendStringChunkParameter(UInt16 parmID, String value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The value to append to the current new value for the parameter.

Remarks

Sample:

for (i = 0; i < 100; i++) { stmt.appendStringChunkParameter(1, "012345"); }

close method

Closes the prepared statement.

Syntax

```
public void close()
```

Remarks

When a prepared statement is closed, all ResultSet and ResultSetSchema objects associated with it are also closed.

It is recommended that you set the preparedStatement object to null immediately after you close it.

executeQuery method

Executes a SQL SELECT statement as a query.

Syntax

```
public ResultSet executeQuery(String persistName)
```

Parameters

- **persistName** The name for the cross-page JavaScript object persistent. Set this value to "null" if the application only has one HTML page, so no object persistent is required.

Returns

The result set of the query as a set of rows.

executeStatement method

Executes a statement that does not return a result set, such as a SQL INSERT, DELETE or UPDATE statement.

Syntax

```
public Int32 executeStatement()
```

Returns

The number of rows affected by the statement.

Remarks

If Connection.autoCommit is true, the statement committed only if one or more rows is affected by the statement.

getPlan method

Returns a text-based description of the query execution plan.

Syntax

```
public PODSString getPlan()
```

Returns

A description of the query execution plan.

Remarks

This method is intended primarily for use during development.

For more information, see "Execution plans in UltraLite" [*UltraLite - Database Management and Reference*].

getResultSetSchema method

Returns the schema describing the result set of this query statement.

Syntax

```
public ResultSetSchema getResultSetSchema()
```

Returns

The schema of this statement's result set.

hasResultSet method

Checks whether the prepared statement generated a result set.

Syntax

```
public Boolean hasResultSet()
```

Returns

True if a result set is generated when this statement is executed, or false if a result set is not generated.

isOpen method

Checks whether this prepared statement is currently open.

Syntax

```
public Boolean isOpen()
```

Returns

True if the prepared statement is open, or false otherwise.

setBooleanParameter method

Sets the value for the specified parameter using a boolean.

Syntax

```
public void setBooleanParameter(UInt16 parmID, Boolean value)
```


Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setBytesParameter method

Sets the value for the specified parameter using an array of bytes.

Syntax

```
public void setBytesParameter(UInt16 parmID, Array value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setDateParameter method

Sets the value for the specified SQLType.DATE type parameter using a date.

Syntax

```
public void setDateParameter(UInt16 parmID, Date value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setDoubleParameter method

Sets the value for the specified SQLType.DOUBLE type parameter using a double.

Syntax

```
public void setDoubleParameter(UInt16 parmID, Double value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setFloatParameter method

Sets the value for the specified SQLType.REAL type parameter.

Syntax

```
public void setFloatParameter(UInt16 parmID, Float value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setIntParameter method

Sets a parameter to a signed 32-bit value.

Syntax

```
public void setIntParameter(UInt16 parmID, Int32 value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setLongParameter method

Sets a parameter to a signed 64-bit value.

Syntax

```
public void setLongParameter(UInt16 parmID, Int64 value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setNullParameter method

Sets the specified parameter to the SQL NULL value.

Syntax

```
public void setNullParameter(UInt16 parmID)
```


Parameters

- **parmID** The 1-based ID of the parameter.

setShortParameter method

Sets a parameter to a signed 16-bit value.

Syntax

```
public void setShortParameter(UInt16 parmID, Int16 value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setStringParameter method

Sets the value for the specified parameter using a String.

Syntax

```
public void setStringParameter(UInt16 parmID, String value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setTimeParameter method

Sets the value for the specified SQLType.TIME type parameter using a date.

Syntax

```
public void setTimeParameter(UInt16 parmID, Date value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setTimestampParameter method

Sets the value for the specified SQLType.TIMESTAMP type parameter using a date.

Syntax

```
public void setTimestampParameter(UInt16 parmID, Date value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setTimestampWithTimeZoneParameter method

Sets the value for the specified SQLType.TIMESTAMP_WITH_TIME_ZONE type parameter using a date.

Syntax

```
public void setTimestampWithTimeZoneParameter(UInt16 parmID, Date value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setULongParameter method

Sets a parameter to an unsigned 64-bit value.

Syntax

```
public void setULongParameter(UInt16 parmID, UInt64 value)
```

Parameters

- **parmID** The 1-based ID of the parameter.
- **value** The new value for the parameter.

setUUIDParameter method

Sets the value for the specified parameter using a UUID.

Syntax

```
public void setUUIDParameter(UInt16 parmID, UUID value)
```


Parameters

- **parmID** The 1-based ID of the parameter.
- **value** A UUID object as the new value for the parameter.

PublicationSchema class

Represents the schema of an UltraLite publication.

Syntax

```
public class PublicationSchema
```

Members

All members of PublicationSchema class, including all inherited members.

Name	Description
"GetName method"	Returns the name of this publication.

GetName method

Returns the name of this publication.

Syntax

```
public String GetName()
```

Returns

The publication name.

ResultSet class

Represents a result set in an UltraLite database.

Syntax

```
public class ResultSet
```

Members

All members of ResultSet class, including all inherited members.

Name	Description
“appendBytes method”	Appends the given bytes to the end of the column.
“appendStringChunk method”	Appends a string chunk to a column.
“close method”	Closes the ResultSet.
“deleteRow method”	Deletes the current row and moves to the next valid row.
“getAGDBSet method”	Bind to the AGDBSet object from the result set.
“getBoolean method”	Returns the value for the specified column as a boolean.
“getBytes method”	Returns the value for the specified column as an array of bytes.
“getBytesSection method”	Gets a binary chunk from the column.
“getDate method”	Returns the value for the specified SQLType.DATE column as a Date object.
“getDouble method”	Returns the value for the specified column as a double.
“getFloat method”	Returns the value for the specified column as a float.
“getInt method”	Returns the value for the specified column as a signed 32-bit integer.
“getLong method”	Returns the value for the specified column as a long.
“getRowCount method”	Returns the number of rows in the ResultSet object.
“getRowCountWithThreshold method”	Returns the number of rows in the result set up to a specified threshold number of rows.
“getShort method”	Returns the value for the specified column as a signed 16-bit integer.
“getString method”	Returns the value of the specified column as a String.
“getStringChunk method”	Copies a subset of the value for the specified SQLType.LONGVARCHAR column, beginning at the specified offset, to the retuned String object.
“getTime method”	Returns the value for the specified SQLType.TIME column as a Date object.
“getTimestamp method”	Returns the value for the specified SQLType.TIMESTAMP column as a Date object.

Name	Description
“getTimestampWithTimeZone method”	Returns the value for the specified SQLType.TIMESTAMP_WITH_TIME_ZONE column as a Date object.
“getULong method”	Returns the value for the specified column as an unsigned 64-bit integer.
“getUUID method”	Returns the value for the specified column as a UUID.
“isBOF method”	Checks whether the current row position is before the first row.
“isEOF method”	Checks whether the current row position is after the last row.
“isNull method”	Checks whether the value from the specified column is NULL.
“isOpen method”	Checks whether this ResultSet is currently open.
“moveAfterLast method”	Moves to a position after the last row of the ResultSet object.
“moveBeforeFirst method”	Moves to a position before the first row of the ResultSet object.
“moveFirst method”	Moves to the position of the first row of the ResultSet object.
“moveLast method”	Moves to the position of the last row of the ResultSet object.
“moveNext method”	Moves to the position of the next row or after the last row of the ResultSet object.
“movePrevious method”	Moves to the position of the previous row or before the first row of the ResultSet object.
“moveRelative method”	Moves from the current row to the offset number of rows.
“setBoolean method”	Sets the value for the specified column using a boolean.
“setBytes method”	Sets the value for the specified column using an array of bytes.
“setDate method”	Sets the value for the specified column using a Date object.
“setDouble method”	Sets the value for the specified column using a double.
“setFloat method”	Sets the value for the specified column using a float.
“setInt method”	Sets the value for the specified column using an integer.
“setLong method”	Sets the value for the specified column using an Int64.
“setNull method”	Sets a column to the SQL NULL.

Name	Description
“setShort method”	Sets the value for the specified column using a UInt16.
“setString method”	Sets the value for the specified column using a String.
“setTime method”	Sets the value for the specified column using a Date object.
“setTimestamp method”	Sets the value for the specified column using a Date object.
“setTimestampWithTimeZone method”	Sets the value for the specified column using a Date object.
“setULong method”	Sets the value for the specified column using a 64-bit long treated as an unsigned value.
“setUUID method”	Sets the value for the specified column using a UUID.
“update method”	Updates the current row.
“updateBegin method”	Selects the updates made for setting columns.
“NULL_TIMESTAMP_VAL variable”	Constant for null timestamp.
“schema variable”	Readonly property; Holds the schema of this result set.

Remarks

This class cannot be directly instantiated. Result sets are created using the PreparedStatement.executeQuery() method.

A result set is only valid while the prepared statement is open.

appendBytes method

Appends the given bytes to the end of the column.

Syntax

```
public void appendBytes(
    UInt16 cid,
    Array value,
    UInt32 srcOffset,
    UInt32 count
)
```

Parameters

- **cid** The 1-based column ID.
- **value** The value to append to the current new value for the column.

- **srcOffset** The start position in the source array.
- **count** The number of bytes to be copied.

appendStringChunk method

Appends a string chunk to a column.

Syntax

```
public void appendStringChunk(UInt16 cid, String value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The value to append to the current new value for the column.

close method

Closes the ResultSet.

Syntax

```
public void close()
```

Remarks

This method frees all resources associated with this object.

deleteRow method

Deletes the current row and moves to the next valid row.

Syntax

```
public void deleteRow()
```

getAGDBSet method

Bind to the AGDBSet object from the result set.

Syntax

```
public ResultSetAGDBSet getAGDBSet()
```

Returns

The ResultSetAGDBSet object.

getBoolean method

Returns the value for the specified column as a boolean.

Syntax

```
public Boolean getBoolean(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a boolean.

getBytes method

Returns the value for the specified column as an array of bytes.

Syntax

```
public Array getBytes(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as an array of bytes.

Remarks

Only valid for columns of type SQLType.BINARY or SQLType.LONGBINARY.

getBytesSection method

Gets a binary chunk from the column.

Syntax

```
public UInt32 getBytesSection(
    UInt16 cid,
    UInt32 srcOffset,
    Array dst,
    UInt32 dstOffset,
```



```
    UInt32 count
)
```

Parameters

- **cid** The 1-based column ID.
- **srcOffset** The zero-relative offset into the source array of bytes. The beginning of the value.
- **dst** A destination array of bytes.
- **dstOffset** The zero-relative offset into the destination array of bytes.
- **count** The number of bytes to be copied.

Returns

The actual number of bytes copied.

getDate method

Returns the value for the specified SQLType.DATE column as a Date object.

Syntax

```
public Date getDate(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a Date object.

getDouble method

Returns the value for the specified column as a double.

Syntax

```
public Double getDouble(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a double.

getFloat method

Returns the value for the specified column as a float.

Syntax

```
public Float getFloat(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a float.

getInt method

Returns the value for the specified column as a signed 32-bit integer.

Syntax

```
public Int32 getInt(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a signed 32-bit integer.

getLong method

Returns the value for the specified column as a long.

Syntax

```
public Int64 getLong(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a long.

Remarks

The value is represented in a double.

getRowCount method

Returns the number of rows in the ResultSet object.

Syntax

```
public UInt32 getRowCount()
```

Returns

The number of rows in the ResultSet object.

getRowCountWithThreshold method

Returns the number of rows in the result set up to a specified threshold number of rows.

Syntax

```
public UInt32 getRowCountWithThreshold(UInt32 threshold)
```

Parameters

- **threshold** The limit on the number of rows to count. Zero indicates no limit.

Returns

The number of rows in the result set.

getShort method

Returns the value for the specified column as a signed 16-bit integer.

Syntax

```
public Int16 getShort(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a signed 16-bit integer.

getString method

Returns the value of the specified column as a String.

Syntax

```
public String getString(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a String.

getStringChunk method

Copies a subset of the value for the specified SQLType.LONGVARCHAR column, beginning at the specified offset, to the retuned String object.

Syntax

```
public String getStringChunk(UInt16 cid, UInt32 srcOffset, UInt32 count)
```

Parameters

- **cid** The 1-based column ID.
- **srcOffset** The start position in the column value. Zero is the beginning of the value.
- **count** The number of characters to be copied.

Returns

The string with specified characters copied.

Remarks

The characters at position srcOffset (starting from 0) through srcOffset+count-1 of the value are copied into the string.

getTime method

Returns the value for the specified SQLType.TIME column as a Date object.

Syntax

```
public Date getTime(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a Date object.

getTimestamp method

Returns the value for the specified SQLType.TIMESTAMP column as a Date object.

Syntax

```
public Date getTimestamp(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a Date object.

getTimestampWithTimeZone method

Returns the value for the specified SQLType.TIMESTAMP_WITH_TIME_ZONE column as a Date object.

Syntax

```
public Date getTimestampWithTimeZone(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a Date object.

getULong method

Returns the value for the specified column as an unsigned 64-bit integer.

Syntax

```
public UInt64 getULong(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as an unsigned 64-bit integer.

Remarks

The value is represented in a Double number.

getUUID method

Returns the value for the specified column as a UUID.

Syntax

```
public UUID getUUID(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a UUID object.

Remarks

The column type must be a SQLtype.BINARY type with length 16.

isBOF method

Checks whether the current row position is before the first row.

Syntax

```
public Boolean isBOF()
```

Returns

True if before first row, or false otherwise.

isEOF method

Checks whether the current row position is after the last row.

Syntax

```
public Boolean isEOF()
```

Returns

True if after last row, or false otherwise.

isNull method

Checks whether the value from the specified column is NULL.

Syntax

```
public Boolean isNull(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

True if value is null, false otherwise.

isOpen method

Checks whether this ResultSet is currently open.

Syntax

```
public Boolean isOpen()
```

Returns

True if the ResultSet is open, or false otherwise.

moveAfterLast method

Moves to a position after the last row of the ResultSet object.

Syntax

```
public void moveAfterLast()
```

moveBeforeFirst method

Moves to a position before the first row of the ResultSet object.

Syntax

```
public void moveBeforeFirst()
```

moveFirst method

Moves to the position of the first row of the ResultSet object.

Syntax

```
public Boolean moveFirst()
```

Returns

True if successful, false otherwise. For example, the method fails if there are no rows.

moveLast method

Moves to the position of the last row of the ResultSet object.

Syntax

```
public Boolean moveLast()
```

Returns

True if successful, false otherwise. For example, the method fails if there are no rows.

moveNext method

Moves to the position of the next row or after the last row of the ResultSet object.

Syntax

```
public Boolean moveNext()
```

Returns

True if successful, or false otherwise. For example, the method fails if there are no more rows.

movePrevious method

Moves to the position of the previous row or before the first row of the ResultSet object.

Syntax

```
public Boolean movePrevious()
```

Returns

True if successful, otherwise the resulting position is BeforeFirst().

moveRelative method

Moves from the current row to the offset number of rows.

Syntax

```
public Boolean moveRelative(Int32 offset)
```


Parameters

- **offset** The number of rows to move.

Returns

True if successful, false otherwise.

setBoolean method

Sets the value for the specified column using a boolean.

Syntax

```
public void setBoolean(UInt16 cid, Boolean value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setBytes method

Sets the value for the specified column using an array of bytes.

Syntax

```
public void setBytes(UInt16 cid, Array value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

Remarks

This method is only suitable for columns of the SQLType.Binary and SQLType.LONGBINARY type.

setDate method

Sets the value for the specified column using a Date object.

Syntax

```
public void setDate(UInt16 cid, Date value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setDouble method

Sets the value for the specified column using a double.

Syntax

```
public void setDouble(UInt16 cid, Double value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setFloat method

Sets the value for the specified column using a float.

Syntax

```
public void setFloat(UInt16 cid, Float value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setInt method

Sets the value for the specified column using an integer.

Syntax

```
public void setInt(UInt16 cid, Int32 value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setLong method

Sets the value for the specified column using an Int64.

Syntax

```
public void setLong(UInt16 cid, Int64 value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value, a 64-bit long for the column.

setNull method

Sets a column to the SQL NULL.

Syntax

```
public void setNull(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

setShort method

Sets the value for the specified column using a UInt16.

Syntax

```
public void setShort(UInt16 cid, Int16 value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setString method

Sets the value for the specified column using a String.

Syntax

```
public void setString(UInt16 cid, String value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setTime method

Sets the value for the specified column using a Date object.

Syntax

```
public void setTime(UInt16 cid, Date value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setTimestamp method

Sets the value for the specified column using a Date object.

Syntax

```
public void setTimestamp(UInt16 cid, Date value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setTimestampWithTimeZone method

Sets the value for the specified column using a Date object.

Syntax

```
public void setTimestampWithTimeZone(UInt16 cid, Date value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setULong method

Sets the value for the specified column using a 64-bit long treated as an unsigned value.

Syntax

```
public void setULong(UInt16 cid, UInt64 value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value, an unsigned 64-bit long for the column.

setUUID method

Sets the value for the specified column using a UUID.

Syntax

```
public void setUUID(UInt16 cid, UUID value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

Remarks

For more information about UUIDs, see "Using UUIDs" [*MobiLink - Server Administration*].

update method

Updates the current row.

Syntax

```
public void update()
```

updateBegin method

Selects the updates made for setting columns.

Syntax

```
public void updateBegin()
```

NULL_TIMESTAMP_VAL variable

Constant for null timestamp.

Syntax

```
public const Date NULL_TIMESTAMP_VAL;
```

schema variable

Readonly property; Holds the schema of this result set.

Syntax

```
public ResultSetSchema schema;
```

Remarks

This property is only valid while its prepared statement is open.

ResultSetSchema class

Represents the schema of an UltraLite result set.

Syntax

```
public class ResultSetSchema
```

Members

All members of ResultSetSchema class, including all inherited members.

Name	Description
“getColumnCount method”	Returns the number of columns in this result set.
“getColumnID method”	Returns the 1-based column ID of the named column.
“getColumnName method”	Returns the name of a column given its 1-based column ID.
“getColumnPrecision method”	Returns the precision of the named column.
“getColumnPrecisionByColID method”	Returns the precision of the column specified by column ID.
“getColumnScale method”	Returns the scale of the named column.
“getColumnScaleByColID method”	Returns the scale of the column specified by the column ID.

Name	Description
“getColumnSize method”	Returns the size of the named column.
“getColumnSizeByColID method”	Returns the size of the column specified by the column ID.
“getColumnSQLType method”	Returns the SQL type of the named column.
“getColumnSQLTypeByColID method”	Returns the SQL type of the column specificed by column ID.
“isOpen method”	Checks whether this result set is currently open.

getColumnCount method

Returns the number of columns in this result set.

Syntax

```
public UInt16 getColumnCount()
```

Returns

The number of columns.

getColumnID method

Returns the 1-based column ID of the named column.

Syntax

```
public UInt16 getColumnID(String name)
```

Parameters

- **name** The name of the column.

Returns

The column ID.

getColumnName method

Returns the name of a column given its 1-based column ID.

Syntax

```
public String getColumnName(UInt16 colID)
```

Parameters

- **colID** The 1-based column ID.

Returns

The name of the column.

getColumnPrecision method

Returns the precision of the named column.

Syntax

```
public Int32 getColumnPrecision(String name)
```

Parameters

- **name** The name of the column.

Returns

The precision of the numeric column.

getColumnPrecisionByColID method

Returns the precision of the column specified by column ID.

Syntax

```
public Int32 getColumnPrecisionByColID(UInt16 colID)
```

Parameters

- **colID** The 1-based column ID.

Returns

The precision of the numeric column.

getColumnScale method

Returns the scale of the named column.

Syntax

```
public Int32 getColumnScale(String name)
```


Parameters

- **name** The name of the column.

Returns

The scale of the numeric column.

getColumnScaleByColID method

Returns the scale of the column specified by the column ID.

Syntax

```
public Int32 getColumnScaleByColID(UInt16 colID)
```

Parameters

- **colID** 1-based column ID.

Returns

The scale of the numeric column.

getColumnSize method

Returns the size of the named column.

Syntax

```
public UInt32 getColumnSize(String name)
```

Parameters

- **name** The name of the column.

Returns

The size of the column.

getColumnSizeByColID method

Returns the size of the column specified by the column ID.

Syntax

```
public UInt32 getColumnSizeByColID(UInt16 colID)
```


Parameters

- **colID** The 1-based column ID.

Returns

The size of the column.

getColumnSQLType method

Returns the SQL type of the named column.

Syntax

```
public UInt16 getColumnSQLType(String name)
```

Parameters

- **name** The name of the column.

Returns

A SQL type.

getColumnSQLTypeByColID method

Returns the SQL type of the column specificed by column ID.

Syntax

```
public UInt16 getColumnSQLTypeByColID(UInt16 colID)
```

Parameters

- **colID** The 1-based column ID.

Returns

A SQL type.

isOpen method

Checks whether this result set is currently open.

Syntax

```
public Boolean isOpen()
```


Returns

True if the result set is open, false otherwise.

SQLType class

Enumerates the available UltraLite SQL database types used as table column types.

Syntax

```
public class SQLType
```

Members

All members of SQLType class, including all inherited members.

Name	Description
“toString method”	Generates the string name of the specified SQL column type constant.
“BAD_INDEX variable”	Column type returned if bad column indentifier is passed to TableSchema.getColumnSQLType().
“BINARY variable”	Binary data, with a specified length.
“BIT variable”	1-bit flag.
“CHAR variable”	Character data, with a specified length.
“DATE variable”	Date information.
“DOUBLE variable”	Double precision floating point number (8 bytes).
“LONGBINARY variable”	Binary data, with variable length.
“LONGVARCHAR variable”	Character data, with variable length.
“MAX_INDEX variable”	Reserved.
“NUMERIC variable”	Exact numerical data, with a specified precision and scale.
“REAL variable”	Single precision floating point number (4 bytes).
“S_BIG variable”	Signed 64-bit integer.
“S_LONG variable”	Signed 32-bit integer.

Name	Description
“S_SHORT variable”	Signed 16-bit integer.
“ST_GEOMETRY variable”	Geo spatial data.
“TIME variable”	Time information.
“TIMESTAMP variable”	Timestamp information (date, time).
“TIMESTAMP_WITH_TIME_ZONE variable”	Timestamp zone information.
“TINY variable”	Unsigned 8-bit integer.
“U_BIG variable”	Unsigned 64-bit integer.
“U_LONG variable”	Unsigned 32-bit integer.
“U_SHORT variable”	Unsigned 16-bit integer.
“UUID variable”	UUID (Universally Unique Identifer).

Remarks

BINARY columns of length 16 are sometimes referred to as UUID columns.

The object is attached to the DatabaseManager object as the sqlType property.

toString method

Generates the string name of the specified SQL column type constant.

Syntax

```
public String toString(UInt16 code)
```

Returns

The name of the type or "BAD_SQL_TYPE" if not a recognized type.

BAD_INDEX variable

Column type returned if bad column indentifier is passed to TableSchema.getColumnSQLType().

Syntax

```
public const UInt16 BAD_INDEX;
```


BINARY variable

Binary data, with a specified length.

Syntax

```
public const UInt16 BINARY;
```

BIT variable

1-bit flag.

Syntax

```
public const UInt16 BIT;
```

CHAR variable

Character data, with a specified length.

Syntax

```
public const UInt16 CHAR;
```

DATE variable

Date information.

Syntax

```
public const UInt16 DATE;
```

DOUBLE variable

Double precision floating point number (8 bytes).

Syntax

```
public const UInt16 DOUBLE;
```

LONGBINARY variable

Binary data, with variable length.

Syntax

```
public const UInt16 LONGBINARY;
```

LONGVARCHAR variable

Character data, with variable length.

Syntax

```
public const UInt16 LONGVARCHAR;
```

MAX_INDEX variable

Reserved.

Syntax

```
public const UInt16 MAX_INDEX;
```

NUMERIC variable

Exact numerical data, with a specified precision and scale.

Syntax

```
public const UInt16 NUMERIC;
```

REAL variable

Single precision floating point number (4 bytes).

Syntax

```
public const UInt16 REAL;
```

S_BIG variable

Signed 64-bit integer.

Syntax

```
public const UInt16 S_BIG;
```


S_LONG variable

Signed 32-bit integer.

Syntax

```
public const UInt16 S_LONG;
```

S_SHORT variable

Signed 16-bit integer.

Syntax

```
public const UInt16 S_SHORT;
```

ST_GEOMETRY variable

Geo spatial data.

Syntax

```
public const UInt16 ST_GEOMETRY;
```

TIME variable

Time information.

Syntax

```
public const UInt16 TIME;
```

TIMESTAMP variable

Timestamp information (date, time).

Syntax

```
public const UInt16 TIMESTAMP;
```

TIMESTAMP_WITH_TIME_ZONE variable

Timestamp zone information.

Syntax

```
public const UInt16 TIMESTAMP_WITH_TIME_ZONE;
```

TINY variable

Unsigned 8-bit integer.

Syntax

```
public const UInt16 TINY;
```

U_BIG variable

Unsigned 64-bit integer.

Syntax

```
public const UInt16 U_BIG;
```

U_LONG variable

Unsigned 32-bit integer.

Syntax

```
public const UInt16 U_LONG;
```

U_SHORT variable

Unsigned 16-bit integer.

Syntax

```
public const UInt16 U_SHORT;
```

UUID variable

UUID (Universally Unique Identifer).

Syntax

```
public const UInt16 UUID;
```


SyncParms class

Represents synchronization parameters that define how to synchronize an UltraLite database.

Syntax

```
public class SyncParms
```

Members

All members of SyncParms class, including all inherited members.

Name	Description
"getAdditionalParms method"	Returns a string of name value pairs.
"getAuthenticationParms method"	Returns parameters provided to a custom user authentication script.
"getDownloadOnly method"	Checks if uploads are disabled when synchronizing.
"getKeepPartialDownload method"	Checks if partial downloads are to be kept after a synchronization that failed because of communication errors or user abort.
"getNewPassword method"	Returns the new MobiLink password for the user specified with setUserName.
"getPartialDownloadRetained method"	Indicates whether UltraLite applied the changes that were downloaded rather than rolling back the changes when a download fails because of a communications error during synchronization.
"getPassword method"	Returns the MobiLink password for the user specified with the setUserName method.
"getPingOnly method"	Checks whether the client is only going to ping the MobiLink server instead of performing a real synchronization.
"getPublications method"	Returns the publications to be synchronized.
"getResumePartialDownload method"	Checks if a previous partial download is to be resumed or discarded.
"getSendColumnNames method"	Checks whether the client is sending column names to the MobiLink synchronization server during synchronization.
"getSendDownloadAck method"	Instructs the MobiLink server whether the client provides a download acknowledgement.

Name	Description
“getStream method”	Returns the type of MobiLink synchronization stream to use for synchronization.
“getStreamParms method”	Returns the parameters used to configure the synchronization stream.
“getUploadOnly method”	Checks if downloads are disabled when synchronizing.
“getUserName method”	Returns the MobiLink user name.
“getVersion method”	Returns the synchronization script to be used.
“setAdditionalParms method”	Specifies a string of name value pairs "name=value;" with extra parameters.
“setAuthenticationParms method”	Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
“setDownloadOnly method”	Disables or enables uploads when synchronizing.
“setKeepPartialDownload method”	Controls whether UltraLite holds on to the partial download rather than rolling back the changes when a download fails because of a communications error during synchronization.
“setMBAServer method”	Sets the UltraLite database synchronization server to be the same as M-Business Anyehre Server.
“setMBAServerWithMoreParms method”	Sets the UltraLite database synchronization server to be the same as M-Business Anywhere Server.
“setNewPassword method”	Sets a new MobiLink password for the user specified with the setUserName method.
“setPassword method”	Sets the MobiLink password for the user specified with the setUserName method.
“setPingOnly method”	Specifies whether the client should only ping the MobiLink synchronization server instead of performing a real synchronization.
“setPublications method”	Sets the publications to be synchronized.
“setResumePartialDownload method”	Specifies whether to resume a failed download.

Name	Description
“setSendColumnNames method”	Specifies whether the client should send column names to the MobiLink synchronization server during synchronization.
“setSendDownloadAck method”	Specifies whether the client should send a download acknowledgement to the MobiLink synchronization server during synchronization.
“setStream method”	Sets the MobiLink synchronization stream to use for synchronization.
“setStreamParms method”	Sets the parameters to configure the synchronization stream.
“setUploadOnly method”	Disables or enables downloads when synchronizing.
“setUserName method”	Sets the user name that uniquely identifies the MobiLink client to the MobiLink server.
“setVersion method”	Specifies which synchronization script to use.
“STREAM_TYPE_HTTP variable”	Synchronize via HTTP.
“STREAM_TYPE_HTTPS variable”	Synchronize via HTTPS (HTTP with RSA transport-layer security).
“STREAM_TYPE_TCPIP variable”	Synchronize via TCP/IP.
“STREAM_TYPE_TLS variable”	Synchronize via TLS.

Remarks

Each connection has its own SyncParms instance.

getAdditionalParms method

Returns a string of name value pairs.

Syntax

```
public String getAdditionalParms()
```

Returns

A string of name value pairs.

Remarks

For more information, see "Additional Parameters synchronization parameter" [*UltraLite - Database Management and Reference*].

getAuthenticationParms method

Returns parameters provided to a custom user authentication script.

Syntax

```
public Array getAuthenticationParms()
```

Returns

An array of strings containing authentication parameters or null if no parameters are specified.

getDownloadOnly method

Checks if uploads are disabled when synchronizing.

Syntax

```
public Boolean getDownloadOnly()
```

Returns

True if uploads are disabled, or false if uploads are enabled.

getKeepPartialDownload method

Checks if partial downloads are to be kept after a synchronization that failed because of communication errors or user abort.

Syntax

```
public Boolean getKeepPartialDownload()
```

Returns

True if downloads are to be kept, false if partial downloads should be rolled back.

getNewPassword method

Returns the new MobiLink password for the user specified with setUserName.

Syntax

```
public String getNewPassword()
```

Returns

The new password to be associated with the MobiLink user after the next synchronization.

getPartialDownloadRetained method

Indicates whether UltraLite applied the changes that were downloaded rather than rolling back the changes when a download fails because of a communications error during synchronization.

Syntax

```
public Boolean getPartialDownloadRetained()
```

Returns

True if a download was interrupted and the partial download was retained, false if the download was not interrupted or if the partial download was rolled back.

getPassword method

Returns the MobiLink password for the user specified with the setUserName method.

Syntax

```
public String getPassword()
```

Returns

The password for the MobiLink user.

getPingOnly method

Checks whether the client is only going to ping the MobiLink server instead of performing a real synchronization.

Syntax

```
public Boolean getPingOnly()
```

Returns

True if client only pings the server, or false if client performs a synchronization.

getPublications method

Returns the publications to be synchronized.

Syntax

```
public String getPublications()
```

Returns

The publications to be synchronized

getResumePartialDownload method

Checks if a previous partial download is to be resumed or discarded.

Syntax

```
public Boolean getResumePartialDownload()
```

Returns

True if the previous partial download is to be resumed, false if the previous partial download is to be rolled back.

getSendColumnNames method

Checks whether the client is sending column names to the MobiLink synchronization server during synchronization.

Syntax

```
public Boolean getSendColumnNames()
```

Returns

True if the client sends column names, or false if the client does not send column names.

getSendDownloadAck method

Instructs the MobiLink server whether the client provides a download acknowledgement.

Syntax

```
public Boolean getSendDownloadAck()
```

Returns

True if the client provides a download acknowledgement, or false if the client does not provide a download acknowledgement.

getStream method

Returns the type of MobiLink synchronization stream to use for synchronization.

Syntax

```
public UInt16 getStream()
```

getStreamParms method

Returns the parameters used to configure the synchronization stream.

Syntax

```
public String getStreamParms(UInt16 newValue)
```

Returns

A string containing all the network protocol options used for synchronization streams.

getUploadOnly method

Checks if downloads are disabled when synchronizing.

Syntax

```
public Boolean getUploadOnly()
```

Returns

True if downloads are disabled, false if downloads are enabled.

getUserName method

Returns the MobiLink user name.

Syntax

```
public String getUserName()
```

Returns

The MobiLink user name.

getVersion method

Returns the synchronization script to be used.

Syntax

```
public String getVersion()
```

Returns

The script version string.

setAdditionalParms method

Specifies a string of name value pairs "name=value;" with extra parameters.

Syntax

```
public void setAdditionalParms(String v)
```

Parameters

- **v** A string of name=value pairs.

Remarks

For more information, see "Additional Parameters synchronization parameter" [*UltraLite - Database Management and Reference*].

setAuthenticationParms method

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Syntax

```
public void setAuthenticationParms(Array v)
```

Parameters

- **v** An array of strings, each containing an authentication parameter. Null array entries result in a synchronization error.

Remarks

Only the first 255 strings are used and each string should be no longer than 128 characters. Longer strings are truncated when sent to the MobiLink server.

setDownloadOnly method

Disables or enables uploads when synchronizing.

Syntax

```
public void setDownloadOnly(Boolean v)
```

Parameters

- **v** Set to true to disable uploads, or set to false to enable uploads.

setKeepPartialDownload method

Controls whether UltraLite holds on to the partial download rather than rolling back the changes when a download fails because of a communications error during synchronization.

Syntax

```
public void setKeepPartialDownload(Boolean v)
```

Parameters

- **v** Set to true to keep partial downloads, or false to roll back partial downloads.

Remarks

UltraLite has the ability to restart downloads that fail because of communication errors. UltraLite processes the download as it is received. If a download is interrupted, then the partial download transaction remains in the database and can be resumed during the next synchronization.

To indicate that UltraLite should save partial downloads, specify Connection.syncParms.setKeepPartialDownload(true); otherwise, the download is rolled back when an error occurs.

If a partial download was kept, then the output field connection.SyncResult.getPartialDownloadRetained()} is set to true when the connection.synchronize() exits. If getPartialDownloadRetained() is set, then you can resume a download. To do this, call connection.synchronize() with connection.syncParms.setResumePartialDownload(boolean) set to true. If another communication error occurs, you might want keep KeepPartialDownload set to true. No upload is done if a download is skipped.

The download you receive during a resumed download is as old as when the download originally began. If you need the most up to date data, then you can do another download immediately after the special resumed download completes.

When resuming a download, many of the SyncParms fields are not relevant. For example, the PublicationMask field is not used. You receive the requested publications on the initial download. The only fields that need to be set are setResumePartialDownload(boolean) and setUserName(String). The field setKeepPartialDownload(boolean) may be set if desired and functions as normal.

If you have a partial download and it is no longer needed, then you can call Connection.rollbackPartialDownload() to roll back the failed download transaction. Also if you attempt to synchronize again and do not specify ResumePartialDownload, then the partial download rolls back before the next synchronization begins.

For more information, see "How synchronization failure is handled" [*MobiLink - Getting Started*].

setMBAServer method

Sets the UltraLite database synchronization server to be the same as M-Business Anyehre Server.

Syntax

```
public void setMBAServer(String host, String port, String url_suffix)
```

Parameters

- **host** The M-Business Anywhere Server host. Set to null to let ULPod configure it automatically.
- **port** The M-Business Anywhere Server port. Set to null to let ULPod configure it automatically.
- **url_suffix** The M-Business Anywhere Server ProxyPass url_suffix, or the MobiLink Redirector url_suffix.

Remarks

Data traffic to the MobiLink server is be routed through the M-Business Server.

If the "host" and "port" values are null, ULPod automatically sets it to the current M-Business server host and port.

"url_suffix" is the same "url_suffix" set in the sync.conf file of M-Business Server. The following section of sync.conf configures M-Business Server to route HTTP database traffic with "url_suffix" to the MobiLink server running on host:port.

<IfModule mod_proxy.c> ProxyRequests On ProxyPass "url_suffix" "http://host:port" </IfModule>

For example, ProxyPass could be followed by /iaredirect/ml http://localhost:8080/ From ULPod client, user could call SyncParms.setMBAServer(null, null, "/iaredirect/ml")

setMBAServerWithMoreParms method

Sets the UltraLite database synchronization server to be the same as M-Business Anywhere Server.

Syntax

```
public void setMBAServerWithMoreParms(
    String host,
    String port,
    String url_suffix,
    String additional
)
```


Parameters

- **host** The M-Business Anywhere Server host. Set to null to let ULPod configure it automatically.
- **port** The M-Business Anywhere Server port. Set to null to let ULPod configure it automatically.
- **url_suffix** The M-Business Anywhere Server ProxyPass url_suffix, or the MobiLink Redirector url_suffix.
- **additional** Additional synchronization stream parameters besides host, port, and url_suffix.

Remarks

Data traffic to the MobiLink server is routed through the M-Business server.

If "host" and "port" values are null, ULPod automatically sets it to the current M-Business Server host and port.

"url_suffix" is the same "url_suffix" set in the sync.conf file of M-Business Server.

"additional" is for additional stream parms. Since this call overides the previous stream parm setting by setStreamParms(), this argument gives caller a chance to add whatever additional synch stream parms that is otherwise lost.

setNewPassword method

Sets a new MobiLink password for the user specified with the setUserName method.

Syntax

```
public void setNewPassword(String v)
```

Parameters

- **v** The new password for MobiLink user.

Remarks

The new password takes effect after the next synchronization.

setPassword method

Sets the MobiLink password for the user specified with the setUserName method.

Syntax

```
public void setPassword(String v)
```

Parameters

- **v** The password for MobiLink user.

Remarks

This user name and password are separate from any database user ID and password, and serves to identify and authenticate the application to the MobiLink server.

setPingOnly method

Specifies whether the client should only ping the MobiLink synchronization server instead of performing a real synchronization.

Syntax

```
public void setPingOnly(Boolean v)
```

Parameters

- **v** Set to true to only ping the server, or false to perform a synchronization.

setPublications method

Sets the publications to be synchronized.

Syntax

```
public void setPublications(String pubs)
```

Parameters

- **pubs** A comma-separated list of publication names.

setResumePartialDownload method

Specifies whether to resume a failed download.

Syntax

```
public void setResumePartialDownload(Boolean v)
```

Parameters

- **v** Set to true to resume the download, or false to roll back the previous partial download.

Remarks

Only one synchronization command (DownloadOnly, PingOnly, ResumePartialDownload, or UploadOnly) may be specified at a time.

setSendColumnNames method

Specifies whether the client should send column names to the MobiLink synchronization server during synchronization.

Syntax

```
public void setSendColumnNames(Boolean v)
```

Parameters

- **v** Set to true to send column names, or false to not send column names.

setSendDownloadAck method

Specifies whether the client should send a download acknowledgement to the MobiLink synchronization server during synchronization.

Syntax

```
public void setSendDownloadAck(Boolean v)
```

Parameters

- **v** Set to true to send a download acknowledgement, or false to tell the server that no download acknowledgement should be sent.

setStream method

Sets the MobiLink synchronization stream to use for synchronization.

Syntax

```
public void setStream(UInt16 newValue)
```

Parameters

- **newValue** The type of MobiLink synchronization stream to use for synchronization.

Remarks

Most synchronization streams require parameters to identify the MobiLink synchronization server address and control other behavior. These parameters are supplied with the setStreamParms() method.

The default stream type is TCPIP.

setStreamParms method

Sets the parameters to configure the synchronization stream.

Syntax

```
public void setStreamParms(String v)
```

Parameters

- **v** A string containing all the parameters used for synchronization streams. Parameters are specified as a semicolon-separated list of name=value pairs ("param1=value1;param2=value2").

Remarks

For information on configuring specific stream types, see "Network protocol options for UltraLite synchronization streams" [*UltraLite - Database Management and Reference*].

setUploadOnly method

Disables or enables downloads when synchronizing.

Syntax

```
public void setUploadOnly(Boolean v)
```

Parameters

- **v** Set to true to disable downloads, or false to enable downloads.

setUserName method

Sets the user name that uniquely identifies the MobiLink client to the MobiLink server.

Syntax

```
public void setUserName(String v)
```

Parameters

- **v** The MobiLink user name.

Remarks

MobiLink uses this value to determine the download content, to record the synchronization state, and to recover from interruptions during synchronization. This user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink synchronization server.

setVersion method

Specifies which synchronization script to use.

Syntax

```
public void setVersion(String v)
```

Parameters

- **v** The script version.

Remarks

Each synchronization script in the consolidated database is marked with a version string. For example, there may be two different download_cursor scripts, and each one is identified by different version strings. The version string allows an UltraLite application to choose from a set of synchronization scripts.

STREAM_TYPE_HTTP variable

Synchronize via HTTP.

Syntax

```
public const UInt32 STREAM_TYPE_HTTP;
```

Remarks

The HTTP stream uses TCP/IP as its underlying transport. UltraLite applications act as Web browsers and the MobiLink synchronization server acts as a Web server. UltraLite applications send POST requests to send data to the server and GET requests to read data from the server.

STREAM_TYPE_HTTPS variable

Synchronize via HTTPS (HTTP with RSA transport-layer security).

Syntax

```
public const UInt32 STREAM_TYPE_HTTPS;
```

Remarks

Transport-layer security is a separately-licensable component and must be ordered before you can install it. To order this component, see the card in your SQL Anywhere Studio package, or see detail?id=1015780.

STREAM_TYPE_TCPIP variable

Synchronize via TCP/IP.

Syntax

```
public const UInt32 STREAM_TYPE_TCPIP;
```

STREAM_TYPE_TLS variable

Synchronize via TLS.

Syntax

```
public const UInt32 STREAM_TYPE_TLS;
```

SyncResult class

Represents the status of the last synchronization.

Syntax

```
public class SyncResult
```

Members

All members of SyncResult class, including all inherited members.

Name	Description
"getAuthStatus method"	Returns the authorization status code for the last synchronization attempt.
"getIgnoredRows method"	Checks if any uploaded rows were ignored during the last synchronization.
"getPartialDownloadRetained method"	Checks if a download was interrupted and whether the partial download was retained.
"getStreamErrorCode method"	Returns the error code reported by the stream.
"getStreamErrorParameters method"	Returns the error parameters reported by the stream.
"getStreamErrorSystem method"	Returns the stream error system-specific code.
"getTimestamp method"	Returns the timestamp of last synchronization.
"getUploadOK method"	Checks whether the last upload synchronization was successful.

Remarks

This object cannot be directly instantiated. Each connection has its own SyncResult instance, attached as its Connection.syncResult property. A SyncResult instance is only valid while that connection is open.

getAuthStatus method

Returns the authorization status code for the last synchronization attempt.

Syntax

```
public UInt16 getAuthStatus()
```

Returns

The authorization status code.

getIgnoredRows method

Checks if any uploaded rows were ignored during the last synchronization.

Syntax

```
public Boolean getIgnoredRows()
```

Returns

True if any uploaded rows were ignored, or false otherwise.

getPartialDownloadRetained method

Checks if a download was interrupted and whether the partial download was retained.

Syntax

```
public Boolean getPartialDownloadRetained()
```

Returns

True if a download was interrupted and the partial download was retained, or false if the download was not interrupted or if the partial download was rolled back.

getStreamErrorCode method

Returns the error code reported by the stream.

Syntax

```
public UInt16 getStreamErrorCode()
```

Returns

The error code.

Remarks

For more information, see "MobiLink communication error messages" [*Error Messages*].

getStreamErrorParameters method

Returns the error parameters reported by the stream.

Syntax

```
public String getStreamErrorParameters()
```

Returns

The stream error parameters.

See also

- "getStreamErrorCode method" on page 137

getStreamErrorSystem method

Returns the stream error system-specific code.

Syntax

```
public UInt32 getStreamErrorSystem()
```

Returns

A system-specific error code.

getTimestamp method

Returns the timestamp of last synchronization.

Syntax

```
public Date getTimestamp()
```

Returns

The timestamp value of last synchronization.

getUploadOK method

Checks whether the last upload synchronization was successful.

Syntax

```
public Boolean getUploadOK()
```

Returns

True if last upload synchronization was successful, or false if last upload synchronization was unsuccessful.

TableSchema class

Represents the schema of an UltraLite table.

Syntax

```
public class TableSchema
```

Members

All members of TableSchema class, including all inherited members.

Name	Description
"getColumnCount method"	Returns the number of columns in this table.
"getColumnDefaultValue method"	Returns the default value of the named column.
"getColumnDefaultValueByColID method"	Returns the default value of the specified column.
"getColumnID method"	Returns the 1-based column ID of the named column.
"getColumnName method"	Returns the name of column identified by the specified column ID.
"getColumnPartitionSize method"	Returns the global autoincrement partition size assigned to the named column.
"getColumnPartitionSizeByColID method"	Returns the global autoincrement partition size assigned to the specified column.
"getColumnPrecision method"	Returns the precision of the named column.
"getColumnPrecisionByColID method"	Returns the precision of the specified column.
"getColumnScale method"	Returns the scale of the named column.
"getColumnScaleByColID method"	Returns the scale of the named column.
"getColumnSize method"	Returns the size of the named column.

Name	Description
“getColumnSizeByColID method”	Returns the size of the specified column.
“getColumnSQLType method”	Returns the SQL type of the named column.
“getColumnSQLTypeByColID method”	Returns the SQL type of the specified column.
“getIndex method”	Gets the schema of an index given its name.
“getIndexCount method”	Returns the number of indexes on this table.
“getIndexName method”	Returns the name of the index identified by the specified index ID.
“getName method”	Returns the name of this table.
“getOptimalIndex method”	Returns the schema of the best index to when searching for a column value.
“getPrimaryKey method”	Returns the index schema of the primary key for this table.
“getPublicationPredicate method”	Returns the publication predicate stored for a given publication name.
“getUploadUnchangedRows method”	Checks whether the database has been configured to upload rows that have not changed.
“isColumnAutoIncrement method”	Checks whether the named column's default is set to autoincrement.
“isColumnAutoIncrementByColID method”	Checks whether the specified column's default is set to autoincrement.
“isColumnCurrentDate method”	Checks whether the specified column's default is set to the current date.
“isColumnCurrentDateByColID method”	Checks whether the specified column defaults to the current date.
“isColumnCurrentTime method”	Checks whether the named column's default is set to the current time.
“isColumnCurrentTimeByColID method”	Checks whether the specified column's default is set to the current time.
“isColumnCurrentTimestamp method”	Checks whether the named column's default is set to the current timestamp.

Name	Description
“isColumnCurrentTimestampByColID method”	Checks whether the specified column's default is set to the current timestamp.
“isColumnCurrentUTCTimestamp method”	Checks whether the specified column's default is set to the current UTC timestamp.
“isColumnGlobalAutoIncrement method”	Checks whether the named column's default is set to global autoincrement.
“isColumnGlobalAutoIncrementByColID method”	Checks whether the specified column's default is set to global autoincrement.
“isColumnNewUUID method”	Checks whether the named column's default is set to a new UUID.
“isColumnNewUUIDByColID method”	Checks whether the specified column's default is set to a new UUID.
“isColumnNullable method”	Checks whether the named column is nullable.
“isColumnNullableByColID method”	Checks whether the specified column is nullable.
“isInPublication method”	Checks whether this table is contained in the named publication.
“isNeverSynchronized method”	Checks whether this table is marked as never being synchronized.
“isOpen method”	Checks whether this TableSchema is currently open.

getColumnCount method

Returns the number of columns in this table.

Syntax

```
public UInt16 getColumnCount()
```

Returns

The short integer number of columns.

getColumnDefaultValue method

Returns the default value of the named column.

Syntax

```
public String getColumnDefaultValue(String name)
```

Parameters

- **name** The name of the column.

Returns

The default value of the named column.

getColumnDefaultValueByColID method

Returns the default value of the specified column.

Syntax

```
public String getColumnDefaultValueByColID(UInt16 colID)
```

Parameters

- **colID** The 1-based column ID.

Returns

The default value of the named column.

getColumnID method

Returns the 1-based column ID of the named column.

Syntax

```
public UInt16 getColumnID(String name)
```

Parameters

- **name** The name of the column.

Returns

The 1-based column ID.

getColumnName method

Returns the name of column identified by the specified column ID.

Syntax

```
public String getColumnName(UInt16 colID)
```

Parameters

- **colID** The 1-based ID of the column.

Returns

The name of the column.

getColumnPartitionSize method

Returns the global autoincrement partition size assigned to the named column.

Syntax

```
public UInt64 getColumnPartitionSize(String name)
```

Parameters

- **name** The name of the column.

Returns

The global autoincrement partition size of the column.

getColumnPartitionSizeByColID method

Returns the global autoincrement partition size assigned to the specified column.

Syntax

```
public UInt64 getColumnPartitionSizeByColID(UInt16 colID)
```

Parameters

- **colID** The 1-based column ID.

Returns

The global autoincrement partition size of the column.

getColumnPrecision method

Returns the precision of the named column.

Syntax

```
public Int32 getColumnPrecision(String name)
```


Parameters

- **name** The name of the column.

Returns

The precision of the numeric column.

getColumnPrecisionByColID method

Returns the precision of the specified column.

Syntax

```
public Int32 getColumnPrecisionByColID(UInt16 colID)
```

Parameters

- **colID** The 1-based column ID.

Returns

The precision of the numeric column.

getColumnScale method

Returns the scale of the named column.

Syntax

```
public Int32 getColumnScale(String name)
```

Parameters

- **name** The name of the column.

Returns

The scale of the numeric column.

getColumnScaleByColID method

Returns the scale of the named column.

Syntax

```
public Int32 getColumnScaleByColID(UInt16 colID)
```


Parameters

- **colID** The 1-based column ID.

Returns

The scale of the numeric column.

getColumnSize method

Returns the size of the named column.

Syntax

```
public UInt32 getColumnSize(String name)
```

Parameters

- **name** The name of the column.

Returns

The size of the column.

getColumnSizeByColID method

Returns the size of the specified column.

Syntax

```
public UInt32 getColumnSizeByColID(UInt16 colID)
```

Parameters

- **colID** The 1-based column ID.

Returns

The size of the column.

getColumnSQLType method

Returns the SQL type of the named column.

Syntax

```
public UInt16 getColumnSQLType(String name)
```

Parameters

- **name** The name of the column.

Returns

A SQL type.

getColumnSQLTypeByColID method

Returns the SQL type of the specified column.

Syntax

```
public UInt16 getColumnSQLTypeByColID(UInt16 colID)
```

Parameters

- **colID** The 1-based column ID.

Returns

A SQL type.

getIndex method

Gets the schema of an index given its name.

Syntax

```
public IndexSchema getIndex(String name)
```

Parameters

- **name** The name of the index.

Returns

The index schema of the named index.

getIndexCount method

Returns the number of indexes on this table.

Syntax

```
public UInt16 getIndexCount()
```

Returns

The number of indexes.

getIndexName method

Returns the name of the index identified by the specified index ID.

Syntax

```
public String getIndexName(UInt16 indexID)
```

Parameters

- **indexID** The 1-based ID of the index.

Returns

The name of the index.

getName method

Returns the name of this table.

Syntax

```
public String getName()
```

Returns

The name of the table.

getOptimalIndex method

Returns the schema of the best index to when searching for a column value.

Syntax

```
public IndexSchema getOptimalIndex(String cname)
```

Parameters

- **cname** The name of the column.

Returns

The index schema for the named column.

getPrimaryKey method

Returns the index schema of the primary key for this table.

Syntax

```
public IndexSchema getPrimaryKey()
```

Returns

The index schema of the primary key.

getPublicationPredicate method

Returns the publication predicate stored for a given publication name.

Syntax

```
public String getPublicationPredicate(String pubName)
```

Parameters

- **pubName** The name of the publication.

Returns

The publication predicate.

getUploadUnchangedRows method

Checks whether the database has been configured to upload rows that have not changed.

Syntax

```
public Boolean getUploadUnchangedRows()
```

Returns

True if the table is marked to always upload all rows during synchronization, or false if the table is marked to upload only changed rows.

isColumnAutoIncrement method

Checks whether the named column's default is set to autoincrement.

Syntax

```
public Boolean isColumnAutoIncrement(String name)
```

Parameters

- **name** The name of the column.

Returns

True if the column is autoincrementing, false otherwise.

isColumnAutoIncrementByColID method

Checks whether the specified column's default is set to autoincrement.

Syntax

```
public Boolean isColumnAutoIncrementByColID(UInt16 colID)
```

Parameters

- **colID** The column ID.

Returns

True if the column is autoincrementing, false otherewise.

isColumnCurrentDate method

Checks whether the specified column's default is set to the current date.

Syntax

```
public Boolean isColumnCurrentDate(String name)
```

Parameters

- **name** The name of the column.

Returns

True if the column has a current date default, or false otherwise.

isColumnCurrentDateByColID method

Checks whether the specified column defaults to the current date.

Syntax

```
public Boolean isColumnCurrentDateByColID(UInt16 colID)
```

Parameters

- **colID** The column ID.

Returns

True if the column defaults to the current date, or false otherwise.

isColumnCurrentTime method

Checks whether the named column's default is set to the current time.

Syntax

```
public Boolean isColumnCurrentTime(String name)
```

Parameters

- **name** The name of the column.

Returns

True if the column defaults to the current time, or false otherwise.

isColumnCurrentTimeByColID method

Checks whether the specified column's default is set to the current time.

Syntax

```
public Boolean isColumnCurrentTimeByColID(UInt16 colID)
```

Parameters

- **colID** The column ID.

Returns

True if the column defaults to the current time, or false otherwise.

isColumnCurrentTimestamp method

Checks whether the named column's default is set to the current timestamp.

Syntax

```
public Boolean isColumnCurrentTimestamp(String name)
```

Parameters

- **name** The name of the column.

Returns

True if the column defaults to the current timestamp, or false otherwise.

isColumnCurrentTimestampByColID method

Checks whether the specified column's default is set to the current timestamp.

Syntax

```
public Boolean isColumnCurrentTimestampByColID(UInt16 colID)
```

Parameters

- **colID** The column ID.

Returns

True if the column defaults to the current timestamp, or false otherwise.

isColumnCurrentUTCTimestamp method

Checks whether the specified column's default is set to the current UTC timestamp.

Syntax

```
public Boolean isColumnCurrentUTCTimestamp(UInt16 colID)
```

Parameters

- **colID** The column ID.

Returns

True if the column defaults to the current UTC timestamp, or false otherwise.

isColumnGlobalAutoIncrement method

Checks whether the named column's default is set to global autoincrement.

Syntax

```
public Boolean isColumnGlobalAutoIncrement(String name)
```

Parameters

- **name** The name of the column.

Returns

True if the column is global autoincrementing, false otherwise.

isColumnGlobalAutoIncrementByColID method

Checks whether the specified column's default is set to global autoincrement.

Syntax

```
public Boolean isColumnGlobalAutoIncrementByColID(UInt16 colID)
```

Parameters

- **colID** The column ID.

Returns

True if the column is global autoincrementing, false otherwise.

isColumnNewUUID method

Checks whether the named column's default is set to a new UUID.

Syntax

```
public Boolean isColumnNewUUID(String name)
```

Parameters

- **name** The name of the column.

Returns

True if the column defaults to a new UUID, false otherwise.

isColumnNewUUIDByColID method

Checks whether the specified column's default is set to a new UUID.

Syntax

```
public Boolean isColumnNewUUIDByColID(UInt16 colID)
```

Parameters

- **colID** The column ID.

Returns

True if the column defaults to a new UUID, false otherwise.

isColumnNullable method

Checks whether the named column is nullable.

Syntax

```
public Boolean isColumnNullable(String name)
```


Parameters

- **name** The name of the column.

Returns

True if the column is nullable, false otherwise.

isColumnNullableByColID method

Checks whether the specified column is nullable.

Syntax

```
public Boolean isColumnNullableByColID(UInt16 colID)
```

Parameters

- **colID** The column ID.

Returns

True if the column is nullable, false otherwise.

isInPublication method

Checks whether this table is contained in the named publication.

Syntax

```
public Boolean isInPublication(String name)
```

Parameters

- **name** The name of the publication.

Returns

True if table in publication, false otherwise.

isNeverSynchronized method

Checks whether this table is marked as never being synchronized.

Syntax

```
public Boolean isNeverSynchronized()
```

Returns

True if the table is omitted from synchronization, or false if the table is included as a synchronizable table.

isOpen method

Checks whether this TableSchema is currently open.

Syntax

```
public Boolean isOpen()
```

Returns

True if the TableSchema is open, or false otherwise.

ULTable class

Represents a table in an UltraLite database.

Syntax

```
public class ULTable
```

Members

All members of ULTable class, including all inherited members.

Name	Description
"appendBytes method"	Appends the given bytes to the end of the column.
"appendStringChunk method"	Appends a string chunk to a column.
"close method"	Closes the table.
"deleteAllRows method"	Deletes all rows in the table.
"deleteRow method"	Deletes the current row and moves to the next valid row.
"findBegin method"	Prepares to perform a new find on this table.
"findFirst method"	Moves forward through the table from the beginning, looking for a row that exactly matches a value or full set of values in the current index.
"findFirstForColumns method"	Moves forward through the table from the beginning, looking for a row that exactly matches a value or partial set of values in the current index.
"findLast method"	Moves backward through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.

Name	Description
“findLastForColumns method”	Moves backward through the table from the end, looking for a row that exactly matches a value or partial set of values in the current index.
“findNext method”	Continues a findFirst() search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or full set of values in the current index.
“findNextForColumns method”	Continues a findFirst(int) search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or partial set of values in the current index.
“findPrevious method”	Continues a findLast() search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or full set of values in the current index.
“findPreviousForColumns method”	Continues a findLast(int) search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or partial set of values in the current index.
“getBoolean method”	Returns the value for the specified column as a boolean.
“getBytes method”	Returns the value for the specified column as an array of bytes.
“getBytesSection method”	Gets a binary chunk from the column.
“getDate method”	Returns the value for the specified SQLType.DATE column as a Date object.
“getDouble method”	Returns the value for the specified column as a double.
“getFloat method”	Returns the value for the specified column as a float.
“getInt method”	Returns the value for the specified column as a signed 32-bit integer.
“getLong method”	Returns the value for the specified column as a long.
“getRowCount method”	Returns the number of rows in the table.
“getRowCountWithThreshold method”	Returns the number of rows in the table up to a specified threshold number of rows.

Name	Description
“getShort method”	Returns the value for the specified column as a signed 16-bit integer.
“getString method”	Returns the value of the specified column as a String.
“getStringChunk method”	Copies a subset of the value for the specified SQLType.LONGVARCHAR column, beginning at the specified offset, to the retuned String object.
“getTime method”	Returns the value for the specified SQLType.TIME column as a Date object.
“getTimestamp method”	Returns the value for the specified SQLType.TIMESTAMP column as a Date object.
“getTimestampWithTimeZone method”	Returns the value for the specified SQLType.TIMESTAMP_WITH_TIME_ZONE column as a Date object.
“getULong method”	Returns the value for the specified column as an unsigned 64-bit integer.
“getUUID method”	Returns the value for the specified column as a UUID.
“insert method”	Inserts a new row with the current column values (specified using the setType methods).
“insertBegin method”	Prepares to insert a new row into this table by setting all current column values to their default values.
“isBOF method”	Checks whether the current row position is before the first row.
“isEOF method”	Checks whether the current row position is after the last row.
“isNull method”	Checks whether the value from the specified column is NULL.
“isOpen method”	Checks whether this table is currently open.
“lookupBackward method”	Moves backward through the table from the end, looking for a row that matches or is less than a value or full set of values in the current index.
“lookupBackwardForColumns method”	Moves backward through the table from the beginning, looking for a row that matches or is less than a value or partial set of values in the current index.
“lookupBegin method”	Prepares to perform a new lookup on this table.

Name	Description
“lookupForward method”	Moves forward through the table from the beginning, looking for a row that matches or is greater than a value or full set of values in the current index.
“lookupForwardForColumns method”	Moves forward through the table from the beginning, looking for a row that matches or is greater than a value or partial set of values in the current index.
“moveAfterLast method”	Moves to a position after the last row of the table.
“moveBeforeFirst method”	Moves to a position before the first row of the table.
“moveFirst method”	Moves to the position of the first row of the table.
“moveLast method”	Moves to the position of the last row of the table.
“moveNext method”	Moves to the position of the next row or after the last row of the table.
“movePrevious method”	Moves to the position of the previous row or before the first row of the table.
“moveRelative method”	Moves from the current row to the offset number of rows.
“open method”	Opens this table for data access using its primary key.
“openWithIndex method”	Opens this table for data access using the specified index.
“setBoolean method”	Sets the value for the specified column using a boolean.
“setBytes method”	Sets the value for the specified column using an array of bytes.
“setDate method”	Sets the value for the specified column using a Date object.
“setDouble method”	Sets the value for the specified column using a double.
“setFloat method”	Sets the value for the specified column using a float.
“setInt method”	Sets the value for the specified column using an integer.
“setLong method”	Sets the value for the specified column using an Int64.
“setNull method”	Sets a column to the SQL NULL.
“setShort method”	Sets the value for the specified column using a UInt16.
“setString method”	Sets the value for the specified column using a String.

Name	Description
“setTime method”	Sets the value for the specified column using a Date object.
“setTimestamp method”	Sets the value for the specified column using a Date object.
“setTimestampWithTimeZone method”	Sets the value for the specified column using a Date object.
“setToDefault method”	Sets the value for the specified column to its default value.
“setULong method”	Sets the value for the specified column using a 64-bit long treated as an unsigned value.
“setUUID method”	Sets the value for the specified column using a UUID.
“truncate method”	Deletes all rows in the table while temporarily activating stop synchronization delete.
“update method”	Updates the current row.
“updateBegin method”	Selects the updates made for setting columns.
“NULL_TIMESTAMP_VAL variable”	Constant for null timestamp.
“schema variable”	Readonly property; Holds the schema of this table.

appendBytes method

Appends the given bytes to the end of the column.

Syntax

```
public void appendBytes(
    UInt16 cid,
    Array value,
    UInt32 srcOffset,
    UInt32 count
)
```

Parameters

- **cid** the 1-based column ID.
- **value** The new name for the column.
- **srcOffset** The start position in the source array.
- **count** The number of bytes to be copied.

appendStringChunk method

Appends a string chunk to a column.

Syntax

```
public void appendStringChunk(UInt16 cid, String value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The value to append to the current new value for the column.

close method

Closes the table.

Syntax

```
public void close()
```

Remarks

This method frees all resources associated with this object.

deleteAllRows method

Deletes all rows in the table.

Syntax

```
public void deleteAllRows()
```

Remarks

In some applications, it can be useful to delete all rows from a table before downloading a new set of data into the table. Rows can be deleted from the UltraLite database without being deleted from the consolidated database using the Connection.startSynchronizationDelete method.

deleteRow method

Deletes the current row and moves to the next valid row.

Syntax

```
public void deleteRow()
```

findBegin method

Prepares to perform a new find on this table.

Syntax

```
public void findBegin()
```

Remarks

The value(s) to search for are specified by calling the appropriate setType method(s) on the columns in the index this table was opened with.

findFirst method

Moves forward through the table from the beginning, looking for a row that exactly matches a value or full set of values in the current index.

Syntax

```
public Boolean findFirst()
```

Returns

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure the cursor position is after the last row (isEOF()).

Each search must be preceded by a call to findBegin().

See also

- "findBegin method" on page 160
- "isEOF method" on page 171

findFirstForColumns method

Moves forward through the table from the beginning, looking for a row that exactly matches a value or partial set of values in the current index.

Syntax

```
public Boolean findFirstForColumns(UInt16 numColumns)
```


Parameters

- **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a numColumns value of 1.

Returns

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that exactly matches the index value. On failure the cursor position is after the last row (isEOF()).

Each search must be preceded by a call to findBegin().

See also

- "findBegin method" on page 160
- "isEOF method" on page 171

findLast method

Moves backward through the table from the end, looking for a row that exactly matches a value or full set of values in the current index.

Syntax

```
public Boolean findLast()
```

Returns

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure the cursor position is before the first row (isBOF()).

See also

- "findBegin method" on page 160
- "isBOF method" on page 171

findLastForColumns method

Moves backward through the table from the end, looking for a row that exactly matches a value or partial set of values in the current index.

Syntax

```
public Boolean findLastForColumns(UInt16 numColumns)
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a numColumns value of 1.

Returns

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row found that exactly matches the index value. On failure the cursor position is before the first row (isBOF()).

Each search must be preceded by a call to findBegin().

See also

- "findBegin method" on page 160
- "isBOF method" on page 171

findNext method

Continues a findFirst() search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or full set of values in the current index.

Syntax

```
public Boolean findNext()
```

Returns

True if successful, false otherwise.

Remarks

The cursor is left on the next row if it exactly matches the index value. On failure the cursor position is after the last row (isEOF()).

The findNext method behavior is undefined if the column values being searched for are modified during a row update.

findNextForColumns method

Continues a findFirst(int) search by moving forward through the table from the current position, looking to see if the next row exactly matches a value or partial set of values in the current index.

Syntax

```
public Boolean findNextForColumns(UInt16 numColumns)
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a numColumns value of 1.

Returns

True if successful, false otherwise.

Remarks

The cursor is left on the next row if it exactly matches the index value. On failure the cursor position is after the last row (isEOF()).

The findNext method behavior is undefined if the column values being searched for are modified during a row update.

findPrevious method

Continues a findLast() search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or full set of values in the current index.

Syntax

```
public Boolean findPrevious()
```

Returns

True if successful, false otherwise.

Remarks

The cursor is left on the previous row if it exactly matches the index value. On failure the cursor position is before the first row (isBOF()).

The findPrevious method behavior is undefined if the column values being searched for are modified during a row update.

findPreviousForColumns method

Continues a findLast(int) search by moving backward through the table from the current position, looking to see if the previous row exactly matches a value or partial set of values in the current index.

Syntax

```
public Boolean findPreviousForColumns(UInt16 numColumns)
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a numColumns value of 1.

Returns

True if successful, false otherwise.

Remarks

The cursor is left on the previous row if it exactly matches the index value. On failure the cursor position is before the first row (isBOF()).

The findPrevious method behavior is undefined if the column values being searched for are modified during a row update.

getBoolean method

Returns the value for the specified column as a boolean.

Syntax

```
public Boolean getBoolean(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a boolean.

getBytes method

Returns the value for the specified column as an array of bytes.

Syntax

```
public Array getBytes(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as an array of bytes.

Remarks

Only valid for columns of type SQLType.BINARY or SQLType.LONGBINARY, or SQLType.UUID when value is of length 16.

getBytesSection method

Gets a binary chunk from the column.

Syntax

```
public UInt32 getBytesSection(
    UInt16 cid,
    UInt32 srcOffset,
    Array dst,
    UInt32 dstOffset,
    UInt32 count
)
```

Parameters

- **cid** The 1-based column ID.
- **srcOffset** The zero-relative offset into the source array of bytes. The beginning of the value.
- **dst** A destination array of bytes.
- **dstOffset** The zero-relative offset into the destination array of bytes.
- **count** The number of bytes to be copied.

Returns

The actual number of bytes copied.

getDate method

Returns the value for the specified SQLType.DATE column as a Date object.

Syntax

```
public Date getDate(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a Date object.

getDouble method

Returns the value for the specified column as a double.

Syntax

```
public Double getDouble(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a double.

getFloat method

Returns the value for the specified column as a float.

Syntax

```
public Float getFloat(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a float.

getInt method

Returns the value for the specified column as a signed 32-bit integer.

Syntax

```
public Int32 getInt(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a signed 32-bit integer.

getLong method

Returns the value for the specified column as a long.

Syntax

```
public Int64 getLong(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a long.

getRowCount method

Returns the number of rows in the table.

Syntax

```
public UInt32 getRowCount()
```

Returns

The number of rows in the table.

getRowCountWithThreshold method

Returns the number of rows in the table up to a specified threshold number of rows.

Syntax

```
public UInt32 getRowCountWithThreshold(UInt32 threshold)
```

Parameters

- **threshold** The limit on the number of rows to count. Zero indicates no limit.

Returns

The number of rows in the table.

getShort method

Returns the value for the specified column as a signed 16-bit integer.

Syntax

```
public Int16 getShort(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a signed 16-bit integer.

getString method

Returns the value of the specified column as a String.

Syntax

```
public String getString(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a string.

getStringChunk method

Copies a subset of the value for the specified SQLType.LONGVARCHAR column, beginning at the specified offset, to the retuned String object.

Syntax

```
public String getStringChunk(UInt16 cid, UInt32 srcOffset, UInt32 count)
```

Parameters

- **cid** The 1-based column ID.
- **srcOffset** The start position in the column value. Zero is the beginning of the value.
- **count** The number of characters to be copied.

Returns

The string with specified characters copied.

Remarks

The characters at position srcOffset (starting from 0) through srcOffset+count-1 of the value are copied into the string.

getTime method

Returns the value for the specified SQLType.TIME column as a Date object.

Syntax

```
public Date getTime(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a Date object.

getTimestamp method

Returns the value for the specified SQLType.TIMESTAMP column as a Date object.

Syntax

```
public Date getTimestamp(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a Date object.

getTimestampWithTimeZone method

Returns the value for the specified SQLType.TIMESTAMP_WITH_TIME_ZONE column as a Date object.

Syntax

```
public Date getTimestampWithTimeZone(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a Date object.

getULong method

Returns the value for the specified column as an unsigned 64-bit integer.

Syntax

```
public UInt64 getULong(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as an unsigned 64-bit integer.

Remarks

The value is represented in a Double number.

getUUID method

Returns the value for the specified column as a UUID.

Syntax

```
public UUID getUUID(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

The column value as a UUID object.

Remarks

Only valid for columns typed SQLtype.UUID or SQLtype.BINARY with length 16.

insert method

Inserts a new row with the current column values (specified using the setType methods).

Syntax

```
public void insert()
```

Remarks

Each insert must be preceded by a call to insertBegin.

insertBegin method

Prepares to insert a new row into this table by setting all current column values to their default values.

Syntax

```
public void insertBegin()
```

Remarks

Call the appropriate setType method(s) to specify the non-default values that are to be inserted.

isBOF method

Checks whether the current row position is before the first row.

Syntax

```
public Boolean isBOF()
```

Returns

True if before first row, false otherwise.

isEOF method

Checks whether the current row position is after the last row.

Syntax

```
public Boolean isEOF()
```

Returns

True if after last row, false otherwise.

isNull method

Checks whether the value from the specified column is NULL.

Syntax

```
public Boolean isNull(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

Returns

True if value is null, false otherwise.

isOpen method

Checks whether this table is currently open.

Syntax

```
public Boolean isOpen()
```

Returns

True if the table is open, or false otherwise.

lookupBackward method

Moves backward through the table from the end, looking for a row that matches or is less than a value or full set of values in the current index.

Syntax

```
public Boolean lookupBackward()
```

Returns

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is less than the index value. On failure (no rows less than the value being looked for) the cursor position is before the first row (isBOF()).

Each search must be preceded by a call to lookupBegin().

lookupBackwardForColumns method

Moves backward through the table from the beginning, looking for a row that matches or is less than a value or partial set of values in the current index.

Syntax

```
public Boolean lookupBackwardForColumns(UInt16 numColumns)
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a numColumns value of 1.

Returns

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is less than the index value. On failure (no rows less than the value being looked for) the cursor position is before the first row (isBOF()).

Each search must be preceded by a call to lookupBegin().

lookupBegin method

Prepares to perform a new lookup on this table.

Syntax

```
public void lookupBegin()
```

Remarks

The value(s) to search for are specified by calling the appropriate setType method(s) on the columns in the index this table was opened with.

lookupForward method

Moves forward through the table from the beginning, looking for a row that matches or is greater than a value or full set of values in the current index.

Syntax

```
public Boolean lookupForward()
```

Returns

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure (no rows greater than the value being looked for) the cursor position is after the last row (isEOF()).

Each search must be preceded by a call to lookupBegin().

lookupForwardForColumns method

Moves forward through the table from the beginning, looking for a row that matches or is greater than a value or partial set of values in the current index.

Syntax

```
public Boolean lookupForwardForColumns(UInt16 numColumns)
```

Parameters

- **numColumns** For composite indexes, the number of columns to use in the lookup. For example, if you have a three column index, and you want to look up a value that matches based on the first column only, you should set the value for the first column, and then supply a numColumns value of 1.

Returns

True if successful, false otherwise.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is left on the first row that matches or is greater than the index value. On failure (no rows greater than the value being looked for) the cursor position is after the last row (isEOF()).

Each search must be preceded by a call to lookupBegin().

moveAfterLast method

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

Syntax

```
public void moveAfterLast()
```

moveBeforeFirst method

Moves to a position before the first row of the table.

Syntax

```
public void moveBeforeFirst()
```

moveFirst method

Moves to the position of the first row of the table.

Syntax

```
public Boolean moveFirst()
```

Returns

True on success, false otherwise.

moveLast method

Moves to the position of the last row of the table.

Syntax

```
public Boolean moveLast()
```

Returns

True on success, false otherwise.

moveNext method

Moves to the position of the next row or after the last row of the table.

Syntax

```
public Boolean moveNext()
```

Returns

True if successful, false otherwise.

movePrevious method

Moves to the position of the previous row or before the first row of the table.

Syntax

```
public Boolean movePrevious()
```

Returns

True on success, false otherwise.

moveRelative method

Moves from the current row to the offset number of rows.

Syntax

```
public Boolean moveRelative(Int32 offset)
```

Parameters

- **offset** The number of rows to move.

Returns

True on success, false otherwise.

open method

Opens this table for data access using its primary key.

Syntax

```
public void open()
```

openWithIndex method

Opens this table for data access using the specified index.

Syntax

```
public void openWithIndex(String index)
```

Parameters

- **index** The name of index to open the table with. If null, the primary key is used.

setBoolean method

Sets the value for the specified column using a boolean.

Syntax

```
public void setBoolean(UInt16 cid, Boolean value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setBytes method

Sets the value for the specified column using an array of bytes.

Syntax

```
public void setBytes(UInt16 cid, Array value)
```


Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

Remarks

Suitable for columns of type SQLType.BINARY, SQLType.LONGBINARY, or for columns of type SQLType.UUID when value is of length 16.

Sample:

var blob = new Array(3); blob[0] = 78; blob[1] = 0; blob[2] = 68; t.setBytes(1, blob);

setDate method

Sets the value for the specified column using a Date object.

Syntax

```
public void setDate(UInt16 cid, Date value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setDouble method

Sets the value for the specified column using a double.

Syntax

```
public void setDouble(UInt16 cid, Double value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setFloat method

Sets the value for the specified column using a float.

Syntax

```
public setFloat(UInt16 cid, Float value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setInt method

Sets the value for the specified column using an integer.

Syntax

```
public void setInt(UInt16 cid, Int32 value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setLong method

Sets the value for the specified column using an Int64.

Syntax

```
public void setLong(UInt16 cid, Int64 value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value, a 64-bit long for the column.

setNull method

Sets a column to the SQL NULL.

Syntax

```
public void setNull(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

setShort method

Sets the value for the specified column using a UInt16.

Syntax

```
public void setShort(UInt16 cid, Int16 value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setString method

Sets the value for the specified column using a String.

Syntax

```
public void setString(UInt16 cid, String value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

Remarks

SQLE_INVALID_PARAMETER is set if len > 32K. For large strings, call AppendStringChunk instead.

setTime method

Sets the value for the specified column using a Date object.

Syntax

```
public void setTime(UInt16 cid, Date value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setTimestamp method

Sets the value for the specified column using a Date object.

Syntax

```
public void setTimestamp(UInt16 cid, Date value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

setTimestampWithTimeZone method

Sets the value for the specified column using a Date object.

Syntax

```
public void setTimestampWithTimeZone(UInt16 cid, Date value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

Remarks

a Date object.

setToDefault method

Sets the value for the specified column to its default value.

Syntax

```
public void setToDefault(UInt16 cid)
```

Parameters

- **cid** The 1-based column ID.

setULong method

Sets the value for the specified column using a 64-bit long treated as an unsigned value.

Syntax

```
public void setULong(UInt16 cid, UInt64 value)
```


Parameters

- **cid** The 1-based column ID.
- **value** The new value, an unsigned 64-bit long for the column.

setUUID method

Sets the value for the specified column using a UUID.

Syntax

```
public void setUUID(UInt16 cid, UUID value)
```

Parameters

- **cid** The 1-based column ID.
- **value** The new value for the column.

Remarks

For more information on UUIDs, see "Using UUIDs" [*MobiLink - Server Administration*].

truncate method

Deletes all rows in the table while temporarily activating stop synchronization delete.

Syntax

```
public void truncate()
```

update method

Updates the current row.

Syntax

```
public void update()
```

Remarks

Each update must be preceded by a call to updateBegin.

updateBegin method

Selects the updates made for setting columns.

Syntax

```
public void updateBegin()
```

Remarks

Column values are modified by calling the appropriate setType method(s).

The data in the row is not actually changed until you execute the update, and that change is not permanent until it is committed.

Modifying columns in the index used to open the table affects any active searches in unpredictable ways. Columns in the primary key of the table can not be updated.

NULL_TIMESTAMP_VAL variable

Constant for null timestamp.

Syntax

```
public const Date NULL_TIMESTAMP_VAL;
```

schema variable

Readonly property; Holds the schema of this table.

Syntax

```
public TableSchema schema;
```

UUID class

Represents a UUID.

Syntax

```
public class UUID
```

Members

All members of UUID class, including all inherited members.

Name	Description
"equals method"	Compares another UUID to this UUID and returns true if they contain the same value.

Remarks

A UUID (Universally Unique Identifer) or GUID (Globally Unique Identifier) is a generated value guaranteed to be unique across all computers and databases. UUIDs are stored as SQLType.BINARY(16) or SQLType.UUID values in UltraLite databases and can be used to uniquely identify rows.

See also

- "getUUID method" on page 170
- "setUUID method" on page 181

equals method

Compares another UUID to this UUID and returns true if they contain the same value.

Syntax

```
public Boolean equals(UUID other)
```

Parameters

- **other** A UUID with which to compare.

Returns

True if this UUID is the same as the other argument, false otherwise.

Index

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

